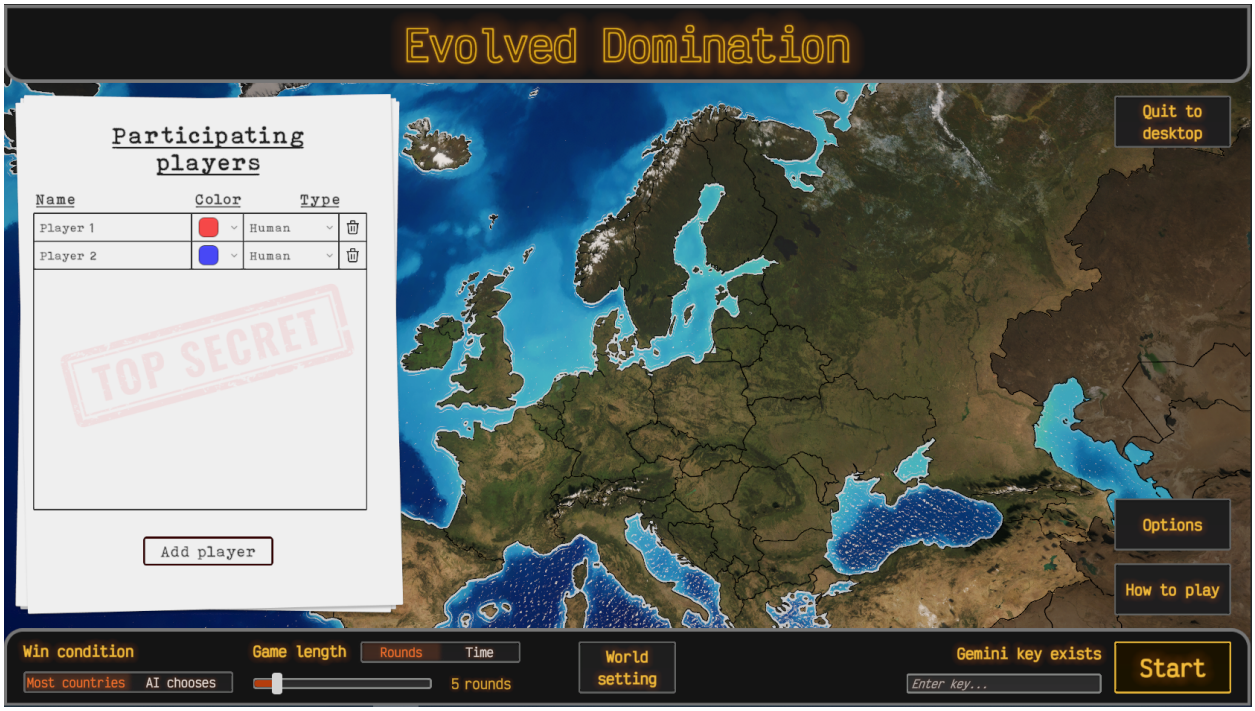
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG




# Evolved Domination

## Exploring generative AI in a turn-based video game context

Master's thesis in Computer science and engineering

Daniel Persson

Joel Båtsman Hilmersson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

# Evolved Domination

Exploring generative AI in a turn-based video game context

Daniel Persson
Joel Båtsman Hilmersson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Evolved Domination
Exploring generative AI in a turn-based video game context
Daniel Persson, Joel Båtsman Hilmersson



Supervisor: Staffan Björk, Department of Computer Science and Engineering
Examiner: Michael Heron, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screenshot of the main menu in the game *Evolved Domination*.

Typeset in LaTeX
Gothenburg, Sweden 2025

Evolved Domination
Exploring generative AI in a turn-based video game context
Daniel Persson, Joel Båtsman Hilmersson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

This master's thesis explores the integration of multiple generative AI models into a turn-based strategy video game to function as a cohesive unit. Generative AI is an interesting topic in the field of video game development since it allows for the automatic creation of content. This can, for instance, increase replayability, shorten development time and enable new game mechanics that were previously impractical. Developments of these benefits have, however, been limited due to the availability of high-quality AI models that minimize hallucinations and inconsistencies when content is generated. To examine this, the strategy video game *Evolved Domination* was constructed through a feature-focused Scrum workflow. It utilizes 12 AI model instances to generate game data, text, images, music and text-to-speech that form a cohesive experience. The findings are presented through a SWOT analysis conducted during the project, highlighting 13 strengths, 10 weaknesses, 11 opportunities and 10 threats of integrating generative AI in a turn-based context. These findings aim to help researchers and developers understand the challenges and potential benefits of generative AI in games, aiding future research.

Keywords: Generative AI, LLM, LLMGA, local AI, games, turn-based game, strategy game

# Acknowledgements

We want to express gratitude to our supervisor, Staffan Björk, for guiding us through the project. He brought many insightful ideas that inspired features for the final game. His mentorship and support have motivated us to push onward when challenges arose and encouraged further exploration in the field. The final result would not have been the same without Staffan.

Daniel Persson & Joel Båtsman Hilmersson, Gothenburg, 2025-06-17

# Contents

# List of Figures

# List of Tables

# Glossary

**AI** - Artifical Intelligence.
**AI instance** - A concept used in *Evolved Domination* to represent a specific component that uses generative AI similar to a chat with an LLM.
**API** - An Application Programming Interface allows different software to communicate.
**C#** - Programming language.
**GA** - A Game Agent capable of operating and perceiving information in a virtual environment.
**GAN** - Generative Adversarial Networks.
**Game state** - A programmatic snapshot of relevant information in the game.
**Gemini** - A family of generative AI models developed by Google.
**Google AI Studio** - A website that allows developers to experiment with different AI models.
**Google Cloud** - A platform developed by Google to host or use services.
**GUI** - Graphical User Interface. Allows users to visually interact with digital components.
**LLM** - Large Language Model. A type of AI trained on large data sets of text used to generate new text.
**LLMGA** - Large Language Model-Based Game Agent. A game agent that uses LLMs to make decisions.
**MVP** - Minimal Viable Product. The first version of a product with all necessary components present.
**NPC** - Non-Player Character usually found in games.
**PNG** - Portable Network Graphics is an image data format.
**REST** - Representational State Transfer. An interface used between two systems to communicate securely.
**STT** - Speech To Text. Technology that converts speech into text.
**SWOT** - Strength Weaknesses Opportunities Threats.
**Technical debt** - A collection of design or implementation constructs that make future changes more costly or impossible.
**TTS** - Text To Speech. Technology that can convert text into spoken voice output.
**VCS** - Version Control System.

# 1

# Introduction

Generative Artificial Intelligence (AI) has received significant attention in recent years due to its ability to generate new creative content upon request. This capability is especially interesting in the field of video game development, where such an AI can be used to dynamically create new game content during gameplay [1]. The procedural generation of in-game content can offer enriched player experiences through unique and personalized content, thereby varying playthroughs and increasing game replayability [2]. Furthermore, leveraging procedurally generated content in game development can lead to substantial savings in development time and resources compared to conventional manual content creation methods [1, 3].

Large Language Models (LLMs) are a subset of all generative AI models that specialize in natural language processing and text generation. This allows them to excel at a wide variety of tasks, ranging from simple conversations to the generation of complex states for specialized systems. However, one weakness of LLMs is their potential to generate information that is considered inaccurate [4]. This inaccurate information is often called "hallucinations", which manifest as generated responses that, despite appearing plausible, lack coherence or logical validity within the intended context [5, 6]. Therefore, finding a reliable way to avoid hallucinations while being able to consistently control the LLMs will enable LLMs to be used in critical situations where it is important for the AI to follow rules.

From an academic standpoint, there are numerous reasons why it is interesting to research how LLMs can be integrated into games effectively. For example, they can be used as Non-Playable-Characters (NPCs), a game master, procedural content generator, game mechanic, commentator or player agent [6]. Finding solutions on how generative AI can be incorporated into games can therefore make advancements in a multitude of areas concerning game design and advance the game development field as a whole.

*Generative Domination* [7] is a prototype video game, previously developed by the authors, to do research in the mentioned areas above. It is a turn-based strategy game that utilizes an LLM to generate game states for each round and works sufficiently for a prototype. However, developing the prototype further would allow new research to be carried out with greater depth than before, since time could be spent more efficiently. It also gives opportunities to study new areas of generative AI in games.

## 1.1 Purpose

The project's purpose is to continue development on *Generative Domination*, a turn-based strategy video game that utilizes an LLM to procedurally generate new game states based on a pre-defined ruleset. The game takes place on a map representing Europe, where players take turns doing in-game actions. After all players have taken their turn, the individual actions are sent to the LLM, which processes the actions according to defined rules. The LLM then responds with a new game state presented to the players. It also aims to incorporate other aspects of generative AI such as image, music and text-to-speech to enhance the game. This purpose is represented and guided by the following research question:

> *How can several generative AI models be utilized together in turn-based strategy video games to procedurally generate new game states?*

Lastly, the project is also developed to create a satisfactory player experience during gameplay. It will therefore demonstrate the feasibility of developing an enjoyable game leveraging generative AI to create unique experiences.

## 1.2 Limitations

Due to time constraints put on the project, limitations had to be considered for the general scope of the project. It was decided that the game would only take place in Europe and that no other area of the world would be playable. This is because the focus is to utilize generative AI inside the project and not map generation. Likewise, no custom map will be created for players to play on for the same reason.

Concerning the generative aspects of the project, the limitation to only using one LLM provider throughout the project was imposed. This simplifies development since it eliminates the need to test different models from different providers when developing. This limitation excludes other forms of generative AI and is only applicable to the LLM used. Moreover, to make development more effective, the decision to allow only four base actions inside the game was made. This decreases the number of pre-defined rules the LLM has to follow and results in time being spent on improving the four existing actions.

Since this project focuses on exploring how different generative models can be used inside a game, it was decided that implementing and testing these would be prioritized over increasing the fun factor of the game. While the goal was still to create an enjoyable game, certain features may be sacrificed in favor of exploring generative AI. Moreover, there are multiple possible game solutions that satisfy the research question, but only one will be explored and developed throughout the thesis.

Lastly, a choice was made not to implement online multiplayer inside the game. This choice was made to avoid having to spend time implementing complicated network code, allowing more focus and time to be spent on the core features of the project. Similarly, split-screen multiplayer was also determined to be too time-consuming to

create. Instead, players have to take turns on the same computer, making input when playing together.

## 1.3 Contribution

The report aims to contribute to the growing body of research done within game development concerning the application of generative AI inside games. Specifically, the focus will be directed towards describing the development process throughout the project, highlighting areas of improvement and lessons learned from developing the systems necessary to complete the purpose mentioned in Section 1.1. A comprehensive SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis will also be conducted during the project, providing information on the use of generative AI in video games and their future potential in the medium.

Moreover, people from other technical fields related to computer science can draw inspiration from the project on how generative AI models can be controlled and used in complex systems. This use case can be applied to multiple scenarios other than games and therefore serves as the main contribution outside the game development field.

# 2

# Background

This chapter highlights important background information around games and AI. It also shows already existing games relevant to the project.

## 2.1 Procedural Content Generation

Procedural Content Generation (PCG) is a technique that primarily uses algorithms to generate game narratives, characters, music, levels or terrain, eliminating the need for manual design [1, 8]. By using PCG, developers can create unique and varied content without requiring large efforts, while maintaining an engaging player experience. Initially, PCG was accomplished using simpler randomization algorithms, but has recently transitioned to use machine learning and generative AI approaches.

## 2.2 Generative AI

Generative AI models use a machine learning architecture to create new data using learned patterns [9]. This facilitates the ability to generate new content such as images, text and audio [1, 10]. Natural text prompts can then generate unique output based on provided instructions. Different generative models include Transformer-based, Generative Adversarial Networks (GAN) and diffusion-based [1, 11]. Text generation is often done using transformer models, while images and audio are often generated using GANs or diffusion models. Additionally, for these models to function, they need to be run either locally on a device or remotely in the cloud.

### 2.2.1 Local models

Local models are capable of running on-device using programs such as Ollama [12] and Unity Sentis [13]. This approach enables reduced latency, more control, greater data security and requires no internet connection, but gives less accurate results [14, 15]. Examples of local models are Llama [16], DeepSeek [17] and Gemma [18].

### 2.2.2 Cloud models

The alternative to running local models is to use cloud models that run on remote servers and can be accessed through a web UI or via an Application Programming

Interface (API) using an API key. This enables greater processing power and the ability to run more complex models, resulting in improved output, but has higher latency and costs [14, 15]. Examples of these are ChatGPT [19], Gemini [20] and Claude [21].

## 2.3 LLMs

Large Language Models (LLMs) simulate natural human-like conversations to solve coding, summarization and information-gathering tasks [4, 22]. These models excel at creating context-aware natural responses, making them great for maintaining coherent conversations. As the LLM space constantly evolves with new models released regularly, model comparison can be challenging. A website designed for testing new models is Chatbot Arena [23]. Based on the testing methodology explained by Chiang *et al.* [24], the website uses user-driven blind tests to determine a model score.

Examples of LLMs are OpenAI's *o3* [19] and Google's *Gemini 2.5 Pro* [20]. Based on the leaderboard provided by Chatbot Arena [23], the two LLMs perform similarly, but *Gemini 2.5 Pro* performs slightly better, depicted in Figure 2.1. *Gemini 2.5 Pro* also provides more free API features than *o3* and has a long context window of 1 048 576 tokens compared to *o3s* 200 000 tokens [25, 26]. Other research also shows that OpenAIs ChatGPT models are slightly better at conversational tasks, while Gemini models prioritize information accuracy and take advantage of Google's knowledge in the search engine field [22, 27].

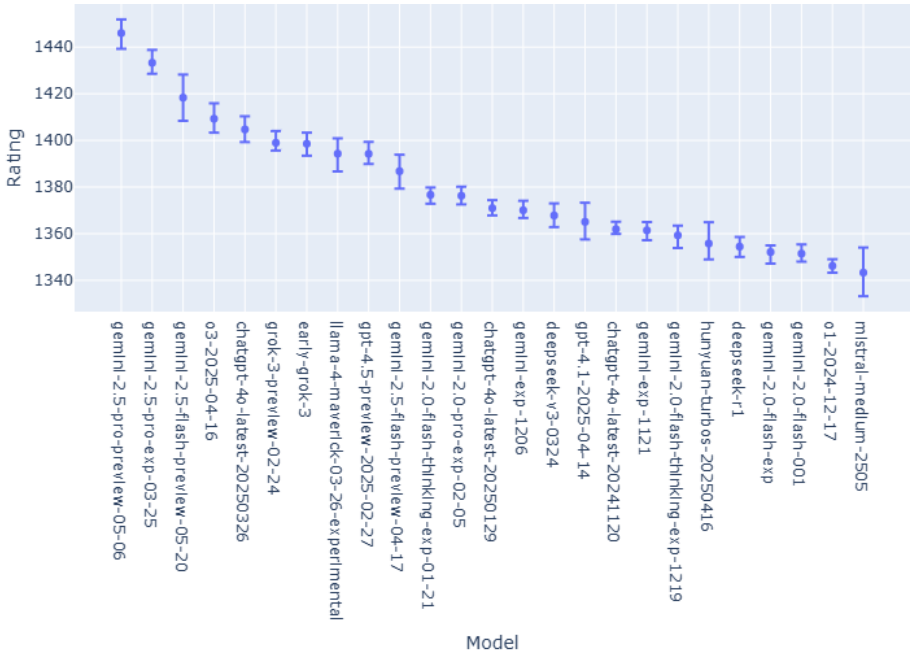



Figure 2.1: The highest scoring LLMs on the Chatbot Arena leaderboard as of 25 May 2025 [23].

Additionally, both models have a structured JSON output schema that helps keep the responses consistent. Structured output guarantees that models adhere to a predefined format when responding [28]. This is formatted as a JSON structure defined by developers when making requests following the JSON Schema standard [29]. This ensures that each response is consistent and has a predictable structure, allowing for easier integration with other APIs or systems.

LLMs usually also have other model parameter settings such as *temperature*, *top p* and *top k* [30]. Temperature, for instance, is an important parameter that controls how random the output from the LLM is [30, 31]. A lower temperature leads to more consistent outputs while higher temperatures lead to increased randomness. This can be seen as how broadly the model uses information from its training data [31].

## 2.4 AI in games

Traditional AI in games focuses on how non-playable characters can perform tasks, for example, decision-making, pathfinding and navigation. To achieve this, the early AI in games relied primarily on techniques such as A* pathfinding and finite state machines that NPCs use [32]. Throughout the years, more advanced techniques, namely behavior-based models, decision trees and reinforcement learning, have been incorporated, creating improved and more believable NPCs.

Another approach is to use LLMs that can play different roles in games, such as a game master steering the game's direction, an assistant giving hints to players or a ruleset manager controlling the game [6]. However, when an LLM simulates these roles, it inherits the same hallucination and bias problems found in these models. This can lead to a degraded player experience if not properly addressed.

## 2.5 Strategy games

Strategy games can broadly be defined as games that emphasize decision-making and strategic thinking. In an article about strategy in games, Dor [33] expresses how strategy is a broad term that can be understood through different lenses. Definitions focus either on war-based origin or gameplay mechanics that require players to plan and adapt to the playing field. Strategy games are often designed to give players multiple choices in which they must anticipate outcomes in combination with resource management. The genre can also be divided into two subcategories, real-time strategy and turn-based games.

## 2.6 Turn-based games

The core idea of turn-based games is that players take turns performing actions without relying on real-time aspects [34]. Turn-based games minimize the need for reflex-based actions but require more cognitive effort from players. One downside

mentioned in the article is that turn-based games create downtime where other players have to wait for their turn. This can be solved using an asynchronous approach where players can perform actions simultaneously, giving players less downtime.

## 2.7 Generative Domination

A turn-based strategy game, developed by the authors, incorporating an LLM, is *Generative Domination* [7], as seen in Figure 2.2. This game takes place on a map of Europe, and the players' goal is to conquer as many countries as possible. This is done through natural text and players take turns performing actions such as attacking, researching, creating rules or becoming allies. The game uses an LLM to modify the game state each round and drive the game forward. Each player's action is sent and processed by the LLM, producing unique player experiences each playthrough. With rule modifications, players can democratically steer the games' direction using rule voting. The game also has virtual players that perform randomized actions based on certain rules, which allow the game to also be played in single-player.

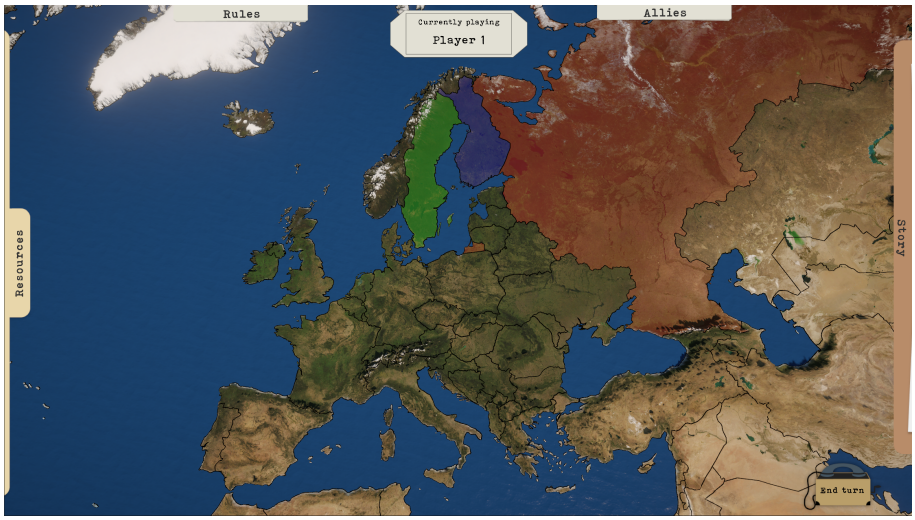



Figure 2.2: The game *Generative Domination*.

### 2.7.1 Player actions inside Generative Domination

This section will describe the in-game actions available to perform inside *Generative Domination*. It will do this by explaining shortly how they function and what the results of them are in-game.

#### Attack

The attack action lets the player attack another country to try and gain control of it. This is done by selecting a country to attack from and a country to attack towards, in combination with writing an instruction on how the attack should be carried out. Depending on whether the central LLM decides the attack is successful, the attacked country will be occupied.

#### Research

The research action allows the player to write an instruction on how they conduct research to discover and get a new resource in the game. The types of resources a player can receive from this are up to their imagination. However, the probability of the LLM approving the proposed research is based on the logic of that item being able to be researched utilizing the players current resources. The LLM can also decide to partially succeed a research action, which results in the player receiving an intermediary item to the desired resource. For instance, an intermediary item towards "advanced explosives" could be "standard explosives".

#### Ally

Unlike the previous actions, the ally action is a multi-round action that does not utilize any LLMs. An ally action in *Generative Domination* begins with one player selecting an already occupied country by another player. The player sending the ally request cannot attach any message with the request, but it will get sent to the player occupying the country next round, where they can choose to accept or reject the alliance offer. Depending on the answer, the alliance will either go into effect during the next round or be rejected.

One weakness of this system was that, immediately after the selection was made to either accept or reject the request, neither of the players could visually see in the GUI that they had officially become allies with each other. They could also not interact in any special way through in-game systems, and the only way that the LLM got to know about their alliance was through a list sent to it during round processing.

#### Rule

The rule action works by allowing players to propose new game rules that should be followed by the central LLM. The rule system works throughout multiple rounds by first allowing the player to propose a rule inside their instruction, which that round gets validated by a special validation LLM to not break existing rules. If it does not, the rule will appear in a pop-up window for all players the round after, which they need to either vote to accept or deny. If the rule gets the majority vote, it will go into the game, and players will be able to see a list of all currently active rules inside the "rules" tab shown in the upper-left part of Figure 2.2.

The negative aspects of this action were that it provided subpar feedback to players regarding whether the rule was accepted or not, and in what step it failed. *Generative Domination* did not state if it were during rule validation or rule voting and instead did not mention anything if it failed. Likewise, when a rule was accepted, that was also not clearly stated, which could lead to confusion for players.

### 2.7.2 Limitations of Generative Domination

Since *Generative Domination* was created in eight weeks, there were several limitations that were not addressed properly due to time constraints. For instance,

players had no way of knowing which color they were inside the game and instead had to remember what selection they made inside the main menu, the starting countries were fixed and always the same, and each country was initiated with the same two resources. Furthermore, there was no win condition, which resulted in players playing endlessly, no setting menu to change any settings and in general, lacking in quality-of-life features such as easy understanding of the GUI and all in-game systems. Maybe the largest limitation was the impossibility for players to input their own API key in the game, required for communication with Gemini, making the game unplayable without having a developer API key.

## 2.8 Related games

This section presents games related to the research area of the thesis. This includes different types of published games, mods and games made for AI research.

### 2.8.1 PANGeA

An example of previous research is the paper *PANGeA* [35], which created a turn-based game called *Dark Shadows* seen in Figure 2.3 that uses procedural artificial narratives. Their research highlights how generative AI in the shape of dialogue generation can be used in games to drive narration forward. They also focus on creating a system for output validation and memory management that allows for consistent content generation.


Figure 2.3: The game *Dark Shadows* used in PANGeA research [35]. CC-BY.

### 2.8.2 Snake story

Another game created for research purposes is *Snake Story* [36]. This is a co-creative storytelling game where players control a snake and collect candy to build a story. Each candy represents a dialogue option with text from an AI or inputted by the player, seen in Figure 2.4. If the candy is picked up, the story advances. The

combination of controlling the snake and creating a story makes this game unique and explores how players react to games with this mix.

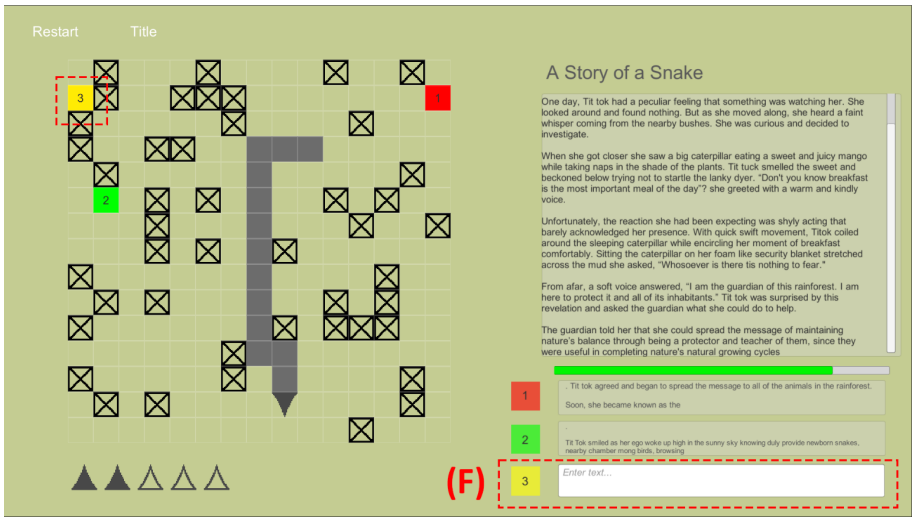



Figure 2.4: Screenshot of the game *Snake Story* with the gameplay to the left and the story to the right [36]. CC BY-SA.

### 2.8.3 Vaudeville

An example of a published game utilizing generative AI is *Vaudeville* [37], seen in Figure 2.5. This detective game lets players solve murders by interacting with AI-powered chatbots [38]. The NPCs respond to the players' natural language input using special memory and personality techniques. Each response is generated in real-time in the cloud and gives the player a unique experience.

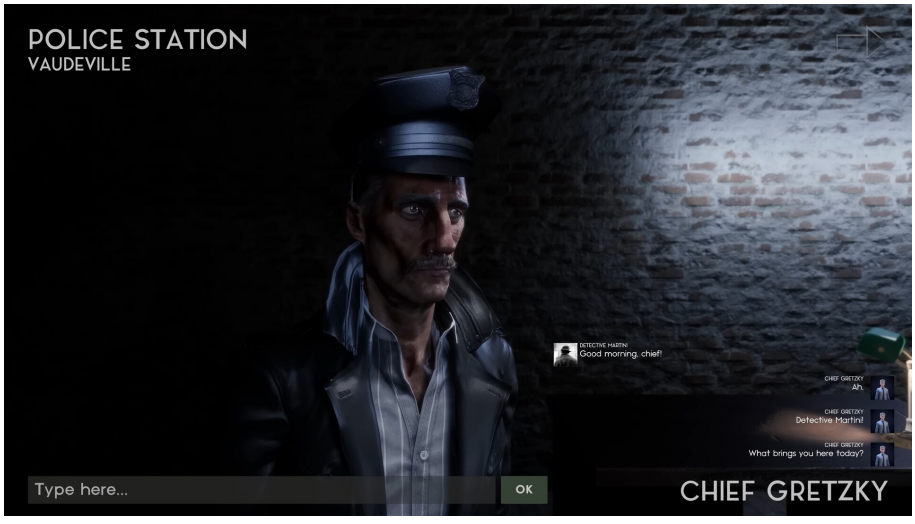



Figure 2.5: In-game footage of Vaudeville [37].

### 2.8.4 Mantella

An example of a more advanced generative AI use in NPCs is *Mantella* [39]. Developed as a mod for Skyrim and Fallout 4, this mod allows players to communicate naturally with characters using Speech-To-Text (STT) or by chatting. The NPCs also answer using Text-To-Speech (TTS) to convert generated text into speech and remember information about the player. This tries to create a more realistic experience while talking to characters in games.

### 2.8.5 Nomic

Another example that fosters dynamic player interaction is the board game *Nomic* [40]. Unlike Vaudeville or Mantella, which rely on AI for interactions, *Nomic* forces players to redefine how the game works by modifying in-game rules. Players debate additions or removal of rules to change how the game is played, enabling unique playthroughs entirely shaped by the player's decisions.

### 2.8.6 Risk Legacy

Another board game is *Risk Legacy* as seen in Figure 2.6, which evolves over multiple playthroughs and creates a unique experience each time it is played [41]. It allows players to modify rules, factions and the game board throughout the game, leading to a continuously changing narrative.


Figure 2.6: The game board Risk Legacy [41].

### 2.8.7 Diplomacy

The *Diplomacy* board game created by Allan Calhammer is a strategic game, illustrated in Figure 2.7, where players try to take control over Europe through negotiation [42]. It uses alliance building as a core feature to foster relationships between

players and strategically maneuver units on a map to conquer other countries. The constant negotiation creates an environment where trust and deception are crucial for players to win.


Figure 2.7: The game board of Diplomacy [43].

Additionally, an LLMGA, as presented in Section 3.7, called Cicero was also created for a digital version of the game to simulate a human player [44]. It uses reinforcement learning, strategic reasoning, dialogue and other techniques to act similarly to a human player. When playing against other players, the agent blended in, and only a few real human players suspected that it was an LLMGA playing, fulfilling its goal of simulating a human player. However, sometimes the agent gave away its plans or wrote messages containing grounding errors, leading to a subpar gameplay style.

### 2.8.8 Project sid

An example of using multiple game agents for cooperation is *Project sid* developed by *Altera.AL* [45]. The project uses a custom Parallel Information Aggregation via Neural Orchestration architecture called PIANO to simulate more than 1000 autonomous AI agents in the game *Minecraft*. Each agent has a role in society and follows rules such as taxation laws. The research shows that these agents can successfully cooperate, forming civilisations mimicking human behaviors.

### 2.8.9 Infinite craft

An example of a game that uses generative AI to create new content on demand is *Infinite Craft* [46] displayed in Figure 2.8. Players start with basic elements such as water, earth, wind and fire, which must be combined to create new elements. The game checks a database for previous combinations with the same elements and if not found, a cloud-based generative AI model creates a new element that gets added to the database.

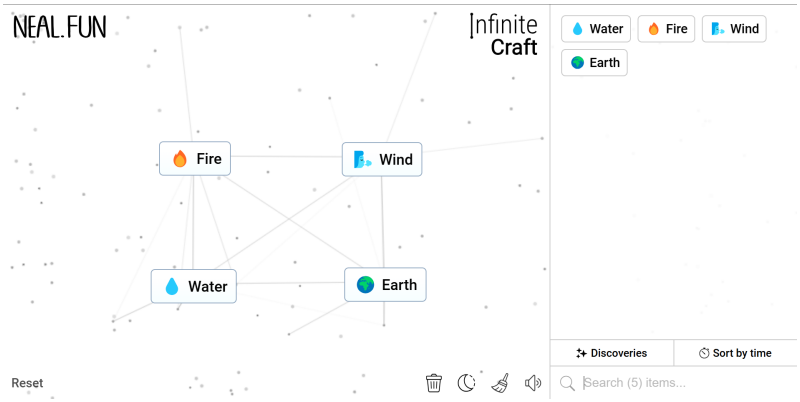



Figure 2.8: Screenshot of the *Infinite Craft* game [46].

### 2.8.10 Oasis

Another example of a game that takes advantage of generative AI is *Oasis* seen in Figure 2.9. The game uses a diffusion transformer model and has been trained on Minecraft gameplay [47]. When receiving input, the game generates a new frame showing the updated game state in real-time.


Figure 2.9: Image from the virtual simulation *Oasis* [47].

## 2.9 Tools

The section presents different tools in the areas of AI, game development and project management.

### 2.9.1 Gemini tools

A tool accompanying Gemini is called *AI studio* [48], where developers can test different types of LLM models, temperature, system instructions and configuration. This allows developers to experiment with different configurations to find appropriate models. Additionally, a tool powered by Gemini is NotebookLM [49], designed to be a peer researcher providing research aid. This is done through conversations where users can ask questions about information in academic papers to gain a deeper understanding of the text.

### 2.9.2 Game engines

The purpose of game engines is to generalize and simplify common features in video games such that components and assets can be reused between different types of games [50]. Components usually present in game engines are, for example, rendering, physics, input, animation, sound and a game loop.

According to Andrade [50], Ranaweera and Mahmoud [51] and Mohd *et al.* [52] popular engines include Unity [53], Unreal [54] and Godot [55]. Unity offers a more newcomer-friendly editor and a higher-level C# scripting language [50]. This makes Unity a popular choice among developers entering game development. Unreal, on the other hand, is known for its hyperrealistic rendering and use of the low-level C++ scripting language, which can be challenging for newer developers [52]. To minimize this, visual node-based scripting can also be used, allowing new developers to take advantage of the power of Unreal without requiring them to learn C++. Godot is a free and open-source alternative giving developers with a low budget, such as indie developers, the option to not have to pay royalties, as is the case with the other two engines [52]. It also primarily targets 2D games and has its proprietary scripting language called GDScript, making it difficult for developers who are not willing to learn a new language.

### 2.9.3 Project management tools

The goal of project management tools is to help teams that use agile practices to organize tasks and be as efficient as possible [56]. Examples of tools are Jira [57], Trello [58] and VSTS [59]. These use a card-centric approach where cards are placed on a board to represent the state of the feature and moved from column to column. Jira and Trello are popular and most appreciated by users, but all of them work with Scrum and Kanban boards and have support for advanced features [56].

### 2.9.4 Version control

A Version Control System (VCS) helps developers keep track of file changes over time [60]. This is helpful in collaborative environments and allows users to roll back to previous versions of the project. Examples of VCS software are Git [61], Apache Subversion [62] and Mercurial [63]. Git is the most popular and brings features that foster online collaboration. Services that utilize Git are, for instance, GitHub [64], GitLab [65] and Bitbucket [66].

# 3

# Theory

This chapter presents relevant theories for the project, such as different AI techniques and a SWOT analysis of *Generative Domination*.

## 3.1 Transformer models

One enabling factor for today's landscape of LLMs and generative AI has been the advent of the *Transformer* [67]. It is a model architecture built upon neural networks with an encoder-decoder structure, similar to other neural sequence transduction models [68, 69], but with a new central *attention* mechanism that outputs a weighted sum of values based on a compatibility function from the input [70]. In practice, this means that a transformer model focuses on the most relevant parts of the input when processing each word, improving its contextual understanding. By training transformer models to predict the next token in a sentence, they can answer questions asked [71] through a generation process called inference [72].

## 3.2 Reasoning models

Recently, a trend emerged in AI development to train LLMs to utilize human-like reasoning to formulate an output [73]. These types of models, built on the transformer architecture, often use an internal prompting technique similar to *chain-of-thought* [74] to generate intermediate "thoughts" [75] that are used before finalizing an output [73]. This can enable reasoning models to improve performance on a variety of tasks compared to traditional models without internal reasoning [76].

## 3.3 Chain-of-agents

One challenge faced by traditional transformer-based models is to solve tasks that require long inputs [77]. The simplified reason for this is that the attention mechanism inside transformers has a quadratic time and space complexity, making long context tasks resource-heavy to run. Chain-of-agents (CoA) solves this by splitting up the underlying data into chunks, which are processed in two main stages together with the input query [78].

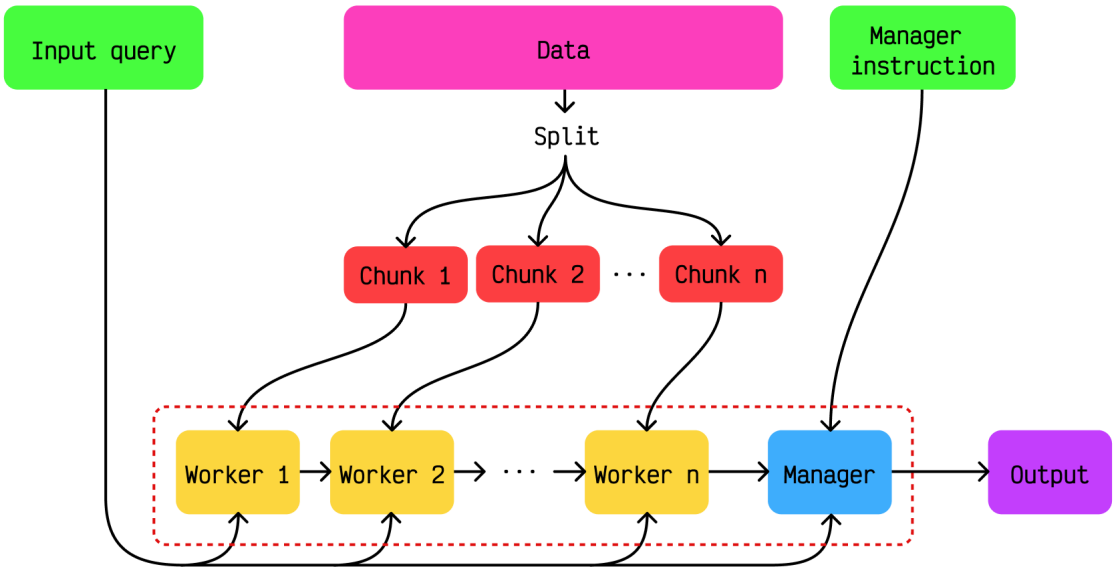



Figure 3.1: The chain-of-agent structure. The dotted red box is the agent chain.

The first stage consists of a chain of worker agents, illustrated in yellow within Figure 3.1, which each gets a chunk of the input, the input query and the output from the previous worker in the chain, if not the first worker. The job of a worker agent is to take these inputs and distill the information down to the next worker. The second stage contains one manager agent, shown in blue in Figure 3.1, which receives all accumulated knowledge from the final worker agent. Together with the input query and a manager instruction, the agent synthesizes this information to generate the final answer. When compared to other methods of handling long input tasks, CoA outperforms these [78].

## 3.4 Mixture of experts

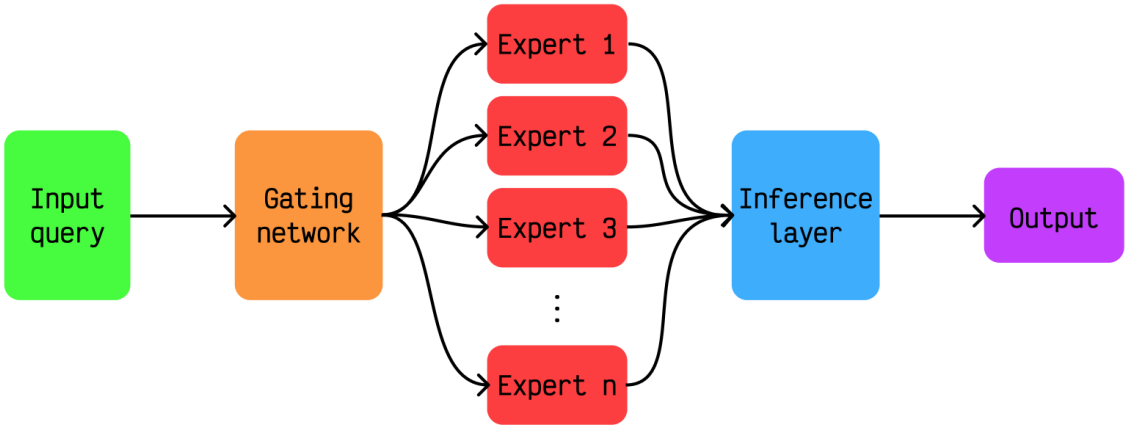



Figure 3.2: Simplified breakdown of the mixture of expert model architecture.

Mixture of Experts (MoE) is a model architecture, visualized in Figure 3.2, designed to efficiently scale LLMs to produce increased output quality [79]. It addresses the computational demands of larger models by employing specialized sub-networks referred to as experts. When an input is received, a *gating network* dynamically routes inputs to the most relevant experts depending on the type of MoE [80]. A *dense* MoE activates all experts to different degrees while a *sparse* MoE activates only a subset of experts [79]. A *sparse* MoE is therefore more computationally efficient, making it especially relevant for LLMs. MoE allows models to have increased capabilities compared to models without it, while still having the same computational cost required, making it an effective way to make models more powerful [79].

## 3.5 AI hallucinations

Generative AI models can sometimes generate responses that are undesirable and considered wrong by humans. This phenomenon is often called hallucinations and refers to when models function correctly but generate outputs outside their desired behavior [81]. Although not inherently harmful [82], generating false information that reads as plausible can be problematic in that it often can be trusted by mistake, which then can be harmful. Avoiding hallucinations is therefore preferable, and having a robust system that either detects them or handles them gracefully can help alleviate the issue if they do occur.

## 3.6 Prompt injections

Prompt injection is a method used to manipulate or override the original behavior of LLMs [5], aiming to change how a model behaves. The malicious inputs trick the LLM into interpreting the input as a new command or instruction instead of following the existing rules to generate output [83]. Consequently, the source highlights that it can lead to output that deviates from the model's original purpose and be considered a security risk.

## 3.7 LLMGAs

Large Language Model-Based Game Agents (LLMGAs) provide games with a simulated player, allowing for behaviors similar to humans [84]. These agents use custom memory, reasoning, action execution and game perception to simulate human behavior in games. Challenges for LLMGAs include proper environment grounding, knowledge discovery and emotional depth [84]. Moreover, a concrete example of an LLMGA in the game Diplomacy called Cicero is mentioned in Section 2.8.7.

## 3.8 SWOT analysis of Generative Domination

A SWOT analysis of generative AI was carried out during the *Generative Domination* project [7]. It highlights strengths, weaknesses, opportunities and threats of

generative AI inside the *Generative Domination* game. The AI offered strengths in reducing the work needed to develop dynamic and diverse storytelling, enhancing replayability by autonomously creating unique narratives that adapted to the player's actions. Weaknesses include challenges in how to effectively control the AI model, ambiguity in knowing where errors arise when issues happen, vulnerability to prompt injections and latency in LLM response times. Opportunities can be found in expanding PCG beyond narratives to include other modalities such as images and sounds, creating adaptive game experiences tailored to player styles and applying similar AI-driven game state update methods to other game genres. Threats include the potential job loss of game developers due to AI's ability to generate creative content at scale, risks of AI hallucinations providing false answers in critical situations and the presence of built-in biases in LLMs that could affect game fairness.

# 4

# Methodology

The following chapter describes different methodologies that can be utilized throughout game development projects. This includes research through design, agile methods, iterative design and more.

## 4.1 Research Through Design

Research Through Design (RtD) defines how research can be conducted by producing and experimenting with design artefacts to explore new ideas [85]. The article describes the approach as a creative and flexible process that obeys traditionally strict scientific methods. RtD aims to create more open-ended answers focusing on exploration and speculation rather than verifiable or falsifiable theories, enabling more creative solutions. However, the downside of not creating verifiable or falsifiable theories can be that it is challenging to assess the validity of the research.

## 4.2 Wicked problems

The concept of wicked problems [86] was introduced by Rittel and Webber in 1973 and defined a new set of problems separated from normal tame problems. Tame problems can be clearly defined, have a defined solution and can be solved using a straightforward problem-solving process. In contrast, wicked problems are ill-defined, deeply intertwined with other problems and often have no solution, making it challenging to handle. To distinguish wicked problems from tame problems, ten characteristics were created [86]. These include aspects such as no definition of done, no definitive formulation of a wicked problem and every problem being unique.

## 4.3 Agile working methods

Agile working methods refer to iterative software development with small, rapid cycles, where customers and developers cooperate via a well-documented, straightforward method that is straightforward to learn and is adaptive to allow for changes during development [87]. It originates from an agile development manifesto [88] and focuses on putting people at the center of development, working software over comprehensive documentation and highlighting the importance of being able to respond

to changes instead of only following a plan. The sections below demonstrate a few of these methods.

### 4.3.1 SCRUM

Scrum is a well-known agile work method that emphasizes flexibility, collaboration and iteration [89]. It utilizes a series of iterations, called sprints, during which project development takes place [89, 90]. In each sprint, work is divided into small manageable tasks picked from a backlog of defined tasks [87, 91]. These are worked on during the sprint, with progress being tracked through regular check-ins and demos. A *sprint review* and *sprint retrospective* are done at the end of a sprint to inspect outcomes and discuss working improvements before the next sprint starts [91]. Based on this feedback, Scrum adapts the project plan continuously throughout the project, focusing on quick delivery of a functional product, avoiding extensive upfront planning [89].

### 4.3.2 Kanban

Kanban is another widely adopted working method that centers around maximizing work efficiency by visualizing task flow throughout a project [92]. This is accomplished through a Kanban board, which represents phases throughout the process linearly with an "In progress" and "Done" step for each [93]. During development, a Kanban card starts from a backlog and progresses through each phase sequentially until it is finished. Tasks are continuously worked on without having a strict structure compared to Scrum [94].

### 4.3.3 Feature driven development

Feature Driven Development (FDD) is an agile method that differentiates itself from Scrum and Kanban, which focuses on the entire software development process, in that it exclusively focuses on the design and building phases of development [95]. It consists of five sequential processes [87], the first three focus on feature definitions and planning, and the last two on iterative feature development.

## 4.4 MoSCoW model

The MoSCoW model illustrates how to divide and categorize tasks into different tiers depending on the importance level of the feature. These levels are "Must-have", which are tasks that must be completed for the project to be considered successful, "Should-have", which are tasks that are non-essential but expected for the final product, "Could-have", which are tasks that have low priority and can be implemented if time allows, and "Won't-have", which are tasks that will not be in the final product but may be considered in the future [96]. The MoSCoW model is an effective way to guarantee that a project delivers a Minimal Viable Product (MVP) even if the time and difficulty level of all tasks are underestimated at the

start [97], making it especially useful when combined with agile methods that rely on task representations.

## 4.5 Iterative design

The iterative design process is a methodology mostly used in software or product development to design, test and refine a product [98]. It is a cyclic process divided into four stages that support constant improvement, as seen in Figure 4.1. The first stage is concept development and includes ideation and brainstorming. The next stage involves prototyping and designing, followed by a user testing stage where the artifact is tested on real users. Lastly, user feedback is analyzed and evaluated to identify areas for improvement and strengths.

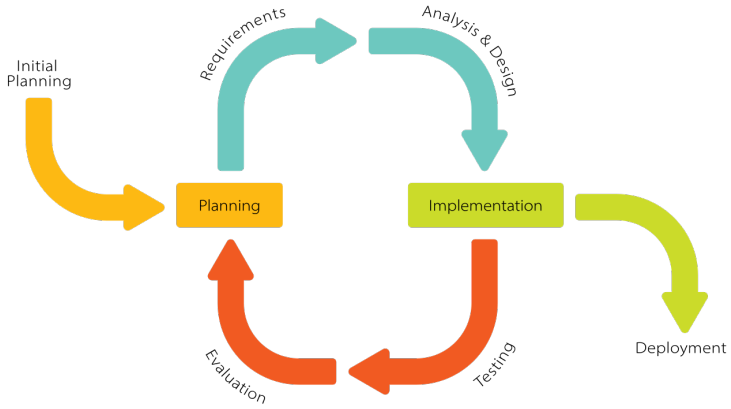



Figure 4.1: The cyclic nature of iterative design [99]. CC BY-SA 4.0.

This process enables benefits such as improved user satisfaction due to the constant feedback loop, quick iteration cycles where features can be quickly implemented and reduced risk of project collapse because of improper early testing on users [98].

### 4.5.1 Double diamond

Another design process is the double diamond process, which is divided into four phases: discover, define, develop and deliver seen in Figure 4.2 [100]. These phases leverage the designers' divergent and convergent thinking abilities. The first phase focuses on discovering user needs, ideation and gathering insights for the problem. The next phase includes interpreting needs and idea selection to create an improved problem definition. Following this, the develop phase includes finding methods on how to bring the product to fruition. The last phase is to produce and deliver the end product.

### 4.5.2 Ideation

Design ideation is the process of forming, developing and discussing ideas to come to a consensus [102]. Ideas are central to the design process and can be either abstract, visual or concrete. A few ideation methods include brainstorming, brainwriting

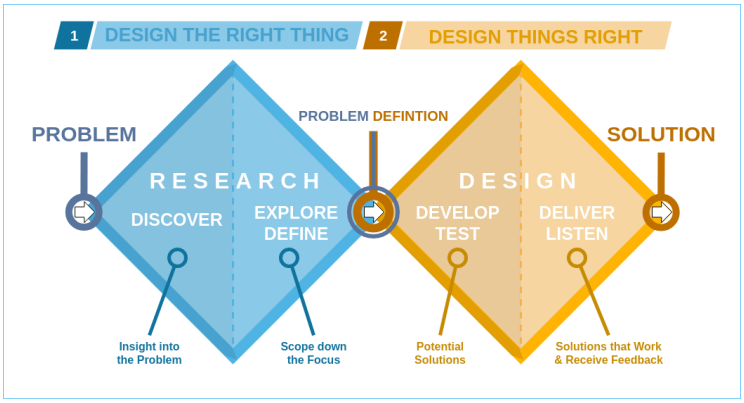



Figure 4.2: Double diamond design process. [101]. CC0.

and Crazy 8. Brainstorming aims to produce numerous ideas through collective discussion where wild ideas are expected and criticism is avoided [103]. Brainwriting can be performed in many different ways but focuses on writing down ideas silently without discussion, leading to a wider range of ideas [103]. Lastly, Crazy 8 is a type of brainwriting where designers individually generate eight ideas in eight minutes [104].

### 4.5.3 Prototyping

Prototyping aids designers to innovate, collaborate and test designs in a concrete and creative way [105]. Prototypes help in decision-making to determine whether an idea is worth pursuing or not. A specific type of prototyping is rapid prototyping, which allows for quick and inexpensive concretization of ideas. Moreover, prototypes can either have low or high fidelity, catering to different testers. Low-fidelity requires more domain knowledge and an understanding of the application. Meanwhile, high-fidelity is more developed and allows for complex interaction.

### 4.5.4 Evaluation

Evaluation is the process of methodically assessing work done by collecting and analyzing information to answer questions about the efficiency or effectiveness of the work [106]. Two common methods for evaluation are formative and summative evaluations [107]. Formative evaluations are typically used throughout the project or at the midpoint, creating an opportunity for reflection and improvement during the project. In contrast, summative evaluation is performed at the end of a project, evaluating the project in its entirety, limiting options for improvements and only reflecting on what has been done.

### 4.5.5 User testing

To gather evaluation data, user testing is often done with users to receive input on the created artifact. This can be done through a multitude of different methods, but is often divided into two categories, *qualitative* and *quantitative* testing [108].

Qualitative methods usually focus on an individual's subjective experience with the artifact, whereas quantitative methods gather multiple data points about the artifact, which are then analyzed. Below are a few different methods that can be used while user testing.

#### Interviews

Interviews are considered a qualitative information gathering method [109] and come in structured, semi-structured and unstructured formats [110]. The main difference between these formats is how the interview questions have been prepared and how much freedom the interviewer has to openly ask unprepared follow-up questions to interview answers.

#### Observation

Observations are also considered a qualitative method [109] and can be done in a "non-reactive", "reactive" and "participant" mode [111]. The non-reactive mode allows no intervention from researchers, the reactive mode allows for intervention but only as the role of an outside observer [111], while the participant mode allows the researcher to be an active member in the study [112]. Moreover, the "think aloud" process can also be utilized to further get to know the thinking process of the participant [113].

#### Questionnaire

Questionnaires are considered a quantitative information-gathering method and are a series of structured questions provided to the participant [114]. Questions can follow different formulas, such as being open questions, a Likert scale [115] and multiple choice questions. There also exist standardized questionnaires such as the Multi-Dimensional Measure of Trust (MDMT) [116] and the System Usability Scale (SUS) [117], commonly used to measure subjects in different areas.

## 4.6 Supervision

Having supervision facilitates discussions to occur about the work done within a project with the aim of refining the result of it. An experienced person often acts as the supervisor and communicates knowledge about the project area down to the supervisees who work on the project. Effective use of supervision, therefore, helps guide the supervisees throughout the project and functions as an opportunity to gain new knowledge in the project field [118]. The bond between supervisor and supervisees is often characterized by mutual respect due to their common goal of co-creation and knowledge gaining [119].

# 5

# Planning

Since this thesis originated from the authors' previous project, *Generative Domination* [7], knowledge gained could be carried over between the two. This included insights into the area of generative AI, related games and the development process as a whole. This resulted in a quicker start to the project compared to if it had started from nothing, but it also resulted in having already defined ways of working, which would be hard to change.

This chapter provides an understanding of the workflow and tools planned for the thesis based on the knowledge gained from the *Generative Domination* project. It will also present an initial time plan to work with and discuss the societal and ethical aspects of generative AI inside games.

## 5.1 Planned workflow

During the *Generative Domination* project, no strict working method was directly used to develop the game. Although this resulted in quicker overall development, it also caused problems with tracking progress throughout the project and not having a clearly defined working structure. Since the authors have worked with similar projects utilizing a Scrum-based workflow before, it was also decided to be used when working with this thesis to more effectively handle documentation, track progress and have a fixed working structure to follow.

Moreover, to prioritize the tasks within the Scrum-workflow, the MoSCoW model, mentioned in Section 4.4, was decided to be utilized since it results in a feature prioritization that ensures a minimal viable product and generally keeps the work relevant. The exact details for how the Scrum-based workflow would function and how features would be prioritized would be decided at a later stage inside Section 6.1.

## 5.2 Planned tools

To complete the project, several tools were needed to create and manage it. Inside *Generative Domination*, Gemini was used as the LLM. As discussed in Section 2.3, both Gemini and ChatGPT provide similar performance, but Gemini is free and provides a large context window up to 1 048 576 tokens. This larger context window

is useful to have as it allows the AI to parse the entire round history data when the game is played. This improved its comprehension of the game flow and remembering what had happened throughout a play session of the game. Gemini also supports structured output, which helps when the response is parsed from the AI. Because of these reasons, the Gemini models were planned to be used throughout the continued development.

Two tools related to Gemini are Google *AI Studio* and NotebookLM, mentioned in Section 2.9.1. *AI studio* proved invaluable during *Generative Domination* as it provided a testing environment to try different Gemini model versions, prompts and system instructions without having to implement code functionality first. This reduced development time and was planned to be utilized further. NotebookLM was decided to be used as a tool to help understand research papers to a greater degree. It allows for a conversation about the contents of added papers, which would help deepen the understanding.

The final use of Gemini that was decided upon was to use it as a peer reviewer to proofread text inside the report and to provide an alternative perspective. This would allow for a quicker time working with the report and more nuanced views brought up throughout it. However, it is important to emphasize that the final text would always be written and checked by the authors.

Development was decided to be done in the game engine Unity as it provides an easy-to-use interface and high-level programming language compared to other engines discussed in Section 2.9.2. The *Generative Domination* game, explained in Section 2.7, was also developed using this engine, making it an obvious choice to continue with.

To manage features and tasks in the project, Trello [58] was decided to be used. As mentioned in Section 2.9.3, it is straightforward and efficient to use, and the authors have previous experience with it. To keep track of changes in the codebase, the VCS service GitHub [64] was chosen as it allows for seamless online collaboration and is well known by the authors. It was also the VCS used for *Generative Domination*.

## 5.3 Initial time plan

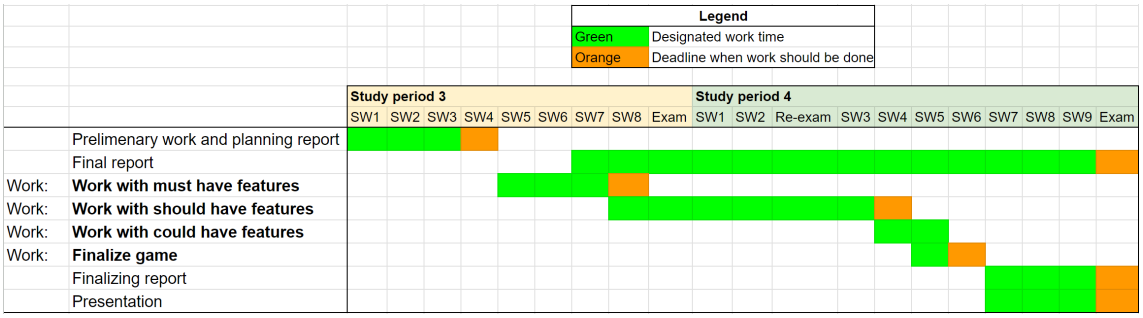



Figure 5.1: The initial time plan created.

An initial time plan was created that would contain a rough plan on how the time would be spent on the thesis. It can be seen in Figure 5.1 and contains elements such as work on the planning report, must-have, should-have and could-have features originating from the MoSCoW model, work on finalizing the report and preparation for the final presentation. Since it is an initial time plan, it was expected to change and become more detailed as the project progressed.

## 5.4 Societal and ethical aspects

Generative AI in video games poses a broad range of societal and ethical challenges, such as fairness, biases and transparency. The datasets used to train LLMs are often large and comprehensive, but can still contain certain societal biases [120, 121]. This can be manifested in the content of the game and lead to narratives that favor certain demographics or viewpoints. If this is very prominent, it can lead to bias reinforcement and stereotypes in games.

Moreover, as presented in Section 3.5, LLMs have the potential to spread misinformation due to their inherent risk of producing hallucinations that appear plausible. This causes the LLM to sometimes output factually incorrect information and may provide inaccurate responses [6]. Solutions for this might include filtering the AI output to minimize hallucinations, but this itself can introduce developer biases.

Another issue is the sustainability concern. Generative AI requires a large amount of processing power to both be trained and run, which contributes to high energy usage [6]. This can lead to an increased carbon footprint, especially if run using non-renewable energy sources.

Lastly, advancements in AI have both negative and positive societal aspects in the job industry. Generative AI has the potential to create new creative content without human intervention, which leads to increased productivity and efficiency for the company, but offloads work from humans [122, 123]. However, this can lead to job displacement and major changes in the labor market for content creators such as developers or artists and especially for people who work with routine tasks [123].

# 6

# Process

This chapter will cover the process of planning and managing the thesis project from initial pre-development stages to the final development stage. It is structured according to the timeline of the project, where each point is written in sequential order based on when it was completed.

## 6.1 Pre-development

The following sections will describe the initial work done before game development could be started, building upon *Generative Domination* [7]. This includes how the project literature study was carried out, how the Scrum-based iterative workflow would function, the categorization of initial features, the creation of a reworked time plan and a preliminary SWOT analysis that was produced.

### 6.1.1 Literature study

A literature study was conducted at the start of the project to gain a deeper understanding of the research area. As generative AI in games is a relatively new area, most research found was between 2020 and the present year. Additionally, the authors had previous knowledge in the area, aiding in knowing which papers are more relevant. Initially, proceedings from the conference *Foundations of Digital Games* (FDG) between the years 2017 and 2024 were analyzed. More specifically, proceedings in the categories regarding artificial intelligence, such as *Game Artificial Intelligence* and *Artificial and computational intelligence for games*, were examined, resulting mainly in the discovery of three relevant papers connected to the research area. It was challenging to find many papers related to generative AI in games due to its novelty, so the methods and results from these papers were mostly used to guide further research into the field. The papers by Yang *et al.* [36], Treynor and McCoy [124] and Earle *et al.* [125] included methods and results relevant to generative AI in games.

Based on this research, another method was used to search academic databases such as ACM, IEEE and Google Scholar with certain keywords. The following search terms and keywords were used: “LLM in video games”, “Generative AI in games”, “game state modification using generative AI”, “turn-based games and ruleset”. These were either created by the authors or found in other papers. This search

phase led to finding additional related papers and games that researchers have created to perform their research. Examples of these were PANGeA [35], Game AI [32] and Cicero [44].

Lastly, Gemini and *AI studio* were used to refine the current literature study by finding additional papers using the feature "search on the web" and prompting the model. Prompting was constructed to find relevant papers in a specific area, for example, "Find me ten academic papers focusing on how LLMs are used in video games". After receiving a response, the sources were examined manually to ensure relevancy and quality. Sometimes the AI responded with inadequate or inaccurate sources. Still, other times, it led to finding high-quality sources in a specific area that was missed during normal searches beneficial to the thesis. Two examples being [6] and [121].

In all cases, for longer and more complex papers, Notebook LM, as presented in Section 2.9.1, was used after reading the paper to have a conversation with the text, extract more information and get a deeper understanding of the content.

### 6.1.2 Scrum-based iterative workflow

After having decided upon a Scrum-based workflow at the planning stages of the project, further details of how to work with it had to be decided. Scrum leverages the benefits of agile development methods, discussed in Section 4.3, to enable feature-driven development. These features would be prioritized using the MoSCoW model, as mentioned in Section 5.1, to help development focus towards the project goal. The Scrum-based workflow could also be connected to Research through Design (RtD), explained in Section 4.1, in that the features can be seen as design artifacts that are produced and evaluated for each sprint, informing future sprint iterations how to best continue and gathering research data.

Another benefit of deciding upon the Scrum-based workflow is its effectiveness in helping tackle wicked problems. Since it is impossible to define a single solution for the end goal of the project, the development of the game can be seen as a wicked problem mentioned in Section 4.2. However, the iterative nature of Scrum allows the project to be responsive and change depending on the situation and therefore better handle the solution space of wicked problems [126].

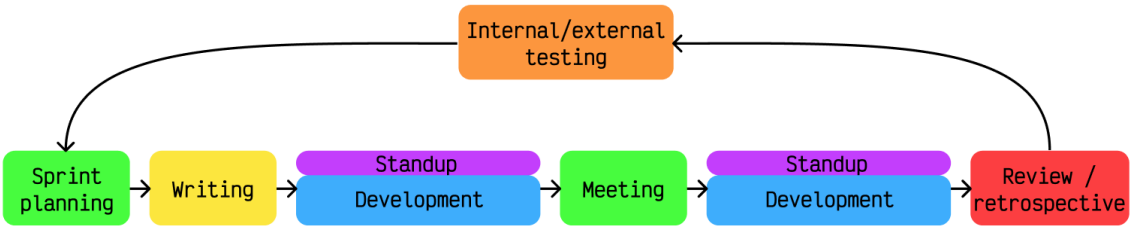



Figure 6.1: Depiction of the work process during a sprint.

Figure 6.1 visualizes a sprint in the Scrum-based iterative workflow chosen during the project. The following two sections provide a description of this cycle.

### Sprint Structure

It was decided to structure the one-week sprints to include planning, meetings, standups, review and retrospective. The sprint would begin with a planning meeting to discuss the current backlog and pick out new tasks for the sprint. In the middle of the sprint, a regular meeting would be held that includes subjects such as an agenda, meeting goals, reports, discussion points, meeting results and a list of things to do before the next meeting. At the end of a sprint, a review and retrospective meeting would be held in combination with each other to demonstrate the new features developed and evaluate the working methods used. Additionally, quick five-minute standups would be held each day, either physically or digitally, to keep each other updated. In the context of wicked problems, having these meetings would keep developers in sync and help to identify unforeseen large features or tasks before they become unmanageable.

To have a clear project management structure, it was decided that tasks in Trello should have a unique identifier containing a number and the name of the feature. When starting with a task, this identifier would be used as a branch name in GitHub, making a clear connection between tasks in Trello and code in GitHub. The different task phases in Trello would follow a sequential order and be *backlog*, *todo*, *in progress*, *testing*, *writing* and *done*. A single task would therefore start in the *backlog*, move over into the *todo* phase during sprint planning, be developed during *in progress*, moved to *testing* when implemented correctly, written about inside *writing* and finally be completely done in *done*. For a task to be written about inside the report, it was decided that both members need to agree that it is relevant to include inside the report, during the sprint end meeting. The task must then be written about during the upcoming sprint.

### Evaluation and user feedback throughout the project

To gather formative data throughout the project, it was decided that the developers would take a questionnaire at the end of each sprint to help analyze project trends. This questionnaire, more accurately described as a feedback form, was thought to be consistent throughout the project and include questions related to game quality. This could then be used to evaluate the previous sprint to better understand how newly added features would affect the game. This was thought to ensure that the workflow could remain responsive since the insights gained from each iteration could steer further development. Moreover, in the context of RtD, this evaluation was also thought to help assess the effectiveness of different solutions implemented and provide more research data.

To incorporate external perspectives on the project, the decision to conduct external playtesting was taken. This would function the same way as described above, but with the difference being that external evaluation was planned to only be conducted every other week. This could uncover inconsistencies and biases in the internal testing that may form based on extra developer knowledge and further help the project progress.

External testers were planned to be picked using convenience sampling [127] and mainly fixed throughout the project. This would help the authors observe trends in answers since the fixed testers could recall and answer based on how they experienced the game compared to prior playtesting sessions. However, other "one-off" testers would also be allowed throughout the project to gather further feedback and views from people who have not previously played the game.

### 6.1.3 Feature categorization

As described in Section 5.1 and 6.1.2, the project was divided into features categorized into must-have, should-have and could-have according to the *MoSCoW* model [96] mentioned in Section 4.4. These features were classified into one of four groups: *AI*, *Game logic*, *GUI* or *Player system* to represent different aspects of the game functionality. The group *AI* contains features related to the generative AI components, and the group *Game logic* includes core features connected to how the game operates. *GUI* encompasses all interactive visual elements and *Player system* covers features related to player interaction.

These features were created during a brainstorming and planning session based on features *Generative Domination* already had. Throughout the project, these features could change or be prioritized, and new ones added.

Table 6.1: Must-have features.

| **Must-have** | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| Expert AI:s | Turn validation | See current allies | Player color |
| Randomize country resources | Message system | API key input | Rule voting for virtual players |
| | Win condition | | Attach resources to actions |
| | Simulate rounds | | |

Table 6.2: Should-have features.

| **Should-have** | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| LLMGA exploration | Resource trading | Story history | Music |
| Modify game parameters | | Loading screen between AI turns | Improved control scheme |
| Generate AI images for narrative | | Display country resources on map | Resource trading |

Table 6.3: Could-have features.

| Could-have | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| Local LLM models | Dynamic music control | Improved navigation | Voice input |
| Generate AI narration for story | Round history | | |
| Customizable environment | Different map shaders | | |

### 6.1.4 Reworked time plan

After the feature categorization was completed, the time plan from Section 5.3 was reworked and built upon based on the progress so far. The must-have, should-have and could-have sections were changed to not overlap with each other, and external playtesting and SWOT analysis writing were added to it. The last addition was to add time for opposition preparation at the end of the project. This time plan can be seen in Figure 6.2 and would serve as a guide throughout the entire thesis project to know roughly how well the project was going.

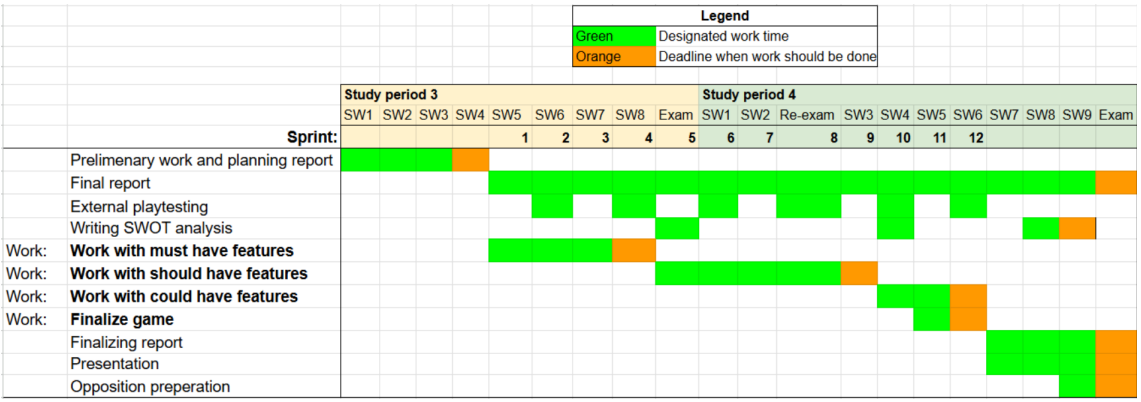



Figure 6.2: A time plan containing time estimation of important phases in the project.

## 6.2 Preliminary SWOT analysis

The SWOT analysis from the previous project *Generative Domination* resulted in general conclusions and key takeaways. Based on these and the literature study, a first version of the analysis can be done, highlighting strengths, weaknesses, opportunities and threats.

### 6.2.1 Strengths

- Reducing the work needed for generating dynamic and diverse game narratives.

- Enhanced replayability due to creating unique narratives based on player actions.
- Long context allows the LLM to keep large amounts of information available at all times.
- LLMGAs is plausible to create in turn-based video games similar to Cicero, mentioned in Section 2.8.7.

### 6.2.2 Weaknesses

- Difficulty in controlling output from the LLM
- Susceptible to prompt injections
- Requires a constant online connection for development to send API requests.
- When LLMGAs are using a message system, they might give away their plan or write messages with grounding errors.

### 6.2.3 Opportunities

- Mixture of experts is good at delegating specific tasks to experts in that area.
- Chain of thoughts helps AI models internally think and reason.
- LLMGAs have the potential to be integrated into the game.
- Can be expanded to include other modalities such as image or sound generation.
- Potentially integrate generative AI in other games.
- Latency in LLM response times.

### 6.2.4 Threats

- Potential of job loss as generative AI can generate content.
- Hallucinations of generative AI models can lead to false answers, misleading players or degrading player experience.

## 6.3 Sprint 1-2: Starting up MVP

As mentioned before, the project used *Generative Domination* as a base of development, which meant that development started from it. Since *Generative Domination* had technical debt, the focus was on fixing these issues during the first two sprints. Initial playtesting was also done for the *Generative Domination* game to create a better understanding of what should be done.

Additionally, four new features were devised called “Randomized starting countries”, “Loading screens”, “Improve main menu” and “AI model selector”. These were not

implemented during these starting sprints but were added as planned features for the project.

### 6.3.1 Player colors

One prominent problem observed inside *Generative Domination* was the confusion players had when knowing their assigned color during gameplay. As seen in Figure 6.3a, there are colors on the world map, representing each player, but no indicator inside the Graphical User Interface (GUI) to know which color corresponds to the correct player. To help with game understanding, a colored border around the top player indicator was added to signal which color the current player has. The resource panel also shows the color of each player through a colored line under the player's name. The addition of player colors to the GUI can be observed in Figure 6.3b.

(a) In game GUI inside *Generative Domination*.

(b) In game GUI after player colors was added to it.

Figure 6.3: Before and after player colors were added to the GUI.

Inside *Generative Domination*, two main classes control the core game loop. A *GameManager* and a *CanvasManager*. The *GameManager* processes and stores all data during the game and collaborates with the *CanvasManager*, which controls the GUI and takes canvas input from the players. Communication between the two classes is done mainly through two structs that hold relevant communication data. It was by adding a player-color field to one of these structs that the *CanvasManager* could color the GUI appropriately.

### 6.3.2 Randomized country resources

In *Generative Domination*, there exists a system that gives players certain country resources based on the countries they occupy. However, as quickly mentioned in Section 2.7.2, the problem was that each country would have the same resources, leading to players constantly acquiring the same resources. Therefore, to increase resource diversity, an LLM was decided to be used to generate logical resources for the countries. A new concept was also created, called an AI instance, which represents a specific component that uses generative AI. This country resource randomizer AI instance was constructed to receive a list of countries and generate two resources for each country. To implement this, a rework was first performed to create an

extensible architecture where developers could easily create new AI instances with specific configurations and system instructions. In *Generative Domination*, all the different AI requests were handled by one class called "Gemini.cs", breaking the Single Responsibility Principle [128]. Work was therefore done to move the code of all AI instances into their own separate classes and make the Gemini class more generic.

After this rework, a new AI instance class was created that inherits all functionality from the Gemini class. To enable consistent outputs from the AI, a strict response schema originating from the internal structure was added:

```
[
    {
        "CountryName": "Sweden",
        "CountryResources": [],
        "CurrentlyOccupying": []
    }
]
```

The configurations also included this system instruction to describe how the AI should respond:

> You should act as a country resource randomizer for populating each country in a list of countries with resources. You will be given a list of countries and you should come up with 2 resources per country.

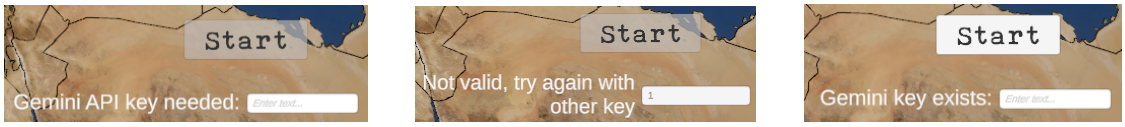
Lastly, code was added to run this directly at game launch, so resources would be ready before starting the round. Now, the system randomly generates two resources the AI deems relevant per country. When testing, this seemed to result in diverse resources and a richer player experience.

### 6.3.3 Manual API key input

To communicate between the local game and the cloud-based Gemini model, requests were sent via an Application Programming Interface (API) to Google Cloud, which provides a response. To access the Gemini API, an *API Key* is required to authenticate the client of the cloud service. Since using an API can cost money, it is often the best practice to hide or avoid uploading a key so that other users with ill intent cannot find and use it maliciously.

Inside *Generative Domination*, the API key was read from a special token file never uploaded to GitHub. This worked well during development, but if a build of the game was made, the API key had to be hard-coded into the game for it to work. This was not optimal because if the game were to be published in this state, the API key could be compromised. To avoid this issue, a system was created that would allow players to manually input their API key when playing a build of the game.

The system works by first checking if a key can be loaded from the token file used inside *Generative Domination*. The file is still used to avoid having to input the key during development. However, if not found, as in the case when the game is built, the main menu prompts the user to input a Gemini API key into a text field.




(a) No API key. (b) Incorrect API key. (c) Verified API key.

Figure 6.4: The different player models used under development from placeholder to final version.

When done, a verification system is triggered that tries to verify the API key before allowing players to start the game. Figure 6.4 shows how the field responds to different API key states inside the game.

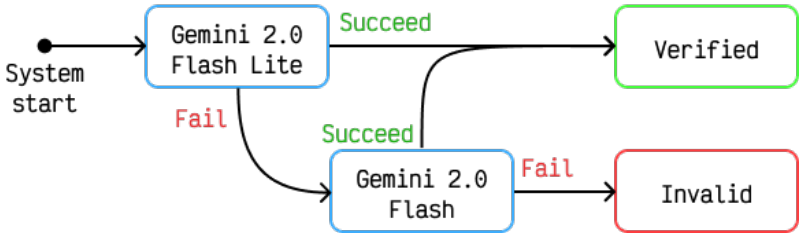



Figure 6.5: Overview of the API verification system.

As seen in Figure 6.5, the verification system functions by first sending a request to the *Gemini 2.0 Flash Lite* AI model and waiting for its response. It is Google's fastest LLM in the family of Gemini models, which is why it is used here to increase the verification speed. However, if it fails, the system also tries calling the *Gemini 2.0 Flash* model as a fallback option to give it a second chance of verifying. The reason for this is that during development, it was observed that the Lite model was sometimes unavailable due to high demand. This would then stop the verification process even if the API key was correct. Adding the second model to test reduced this possibility and made the API verification system more robust.

Lastly, there was now the possibility that the API key would not be loaded when the game started. The timing for when to randomize country resources had to be changed to do it after the API key is verified. The Gemini model was also changed from the Flash model to Flash Lite to make the process quicker.

### 6.3.4 Round-based game length

As explained in Section 2.7.2, *Generative Domination* lacked a win condition and could continue forever. Therefore, a system was created to display a winner at the end of the game after a specific number of rounds has passed. This was done by introducing a new variable indicating the maximum number of rounds the game will run for. To change the variable, a slider in the main menu was added. Additionally, a victory pop-up was created to be used when the game is over. The inspiration for this was to imitate the functionality and design of a typewriter, adhering to the old war graphical style in the game.


Figure 6.6: Victory pop-up showing a typewriter and a paper with the winning player and a leaderboard.

The typewriter and victory pop-up can be seen in Figure 6.6. The typewriter, paper and text are animated. First, the typewriter moves up from the bottom of the screen. When in position, the paper starts to appear from behind the typewriter. Additionally, the text is printed with a tiny delay between each character print, simulating the machine writing out the text by typing each character. All of these animations were accomplished using Unity coroutines and a linear interpolation to move the objects from one position to another, except the text printing, which required no movement. Sounds were also added when each character was printed, and a "ding" sound was made when the typing was done. The algorithm for calculating who won takes all players and counts which one has the highest number of occupied countries.

### 6.3.5 Deterministic players

As noted during the *Generative Domination* project [7], it was challenging to effectively test and evaluate the AI models used inside the game since it always produces a different result every gameplay session. To keep the testing consistent and only observe changes to the output, an instruction set had been written that would be manually inputted into the game before evaluating it. This resulted in a simulation of the game with consistent input for two players taking three turns before the state of the game could be evaluated.

One big problem with this method of simulating rounds, however, was the inefficiencies of manually inputting each action into the game every time the game would be evaluated. To avoid this issue, an idea to utilize the existing *virtual player* system found inside *Generative Domination* to automatically input these instructions was conceived. This was thought to be the easiest solution to the problem since it would not require a whole new system to be implemented for round simulations.

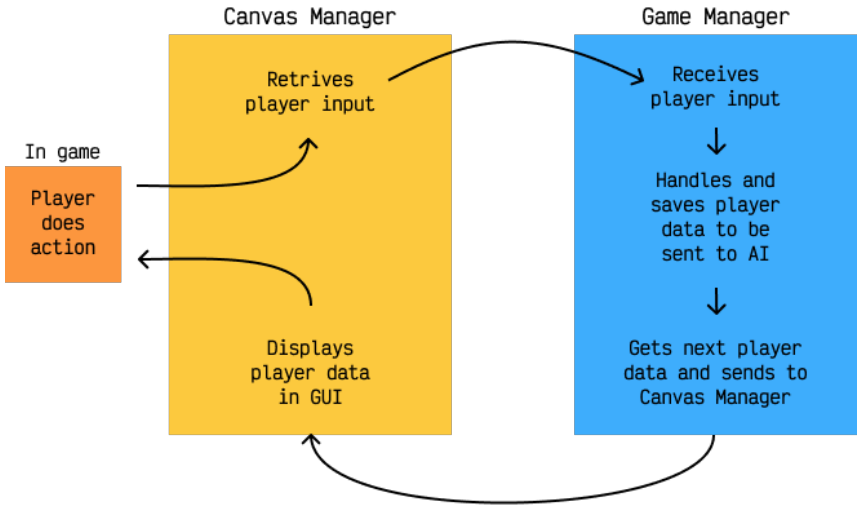



Figure 6.7: Simplification of the internal game loop during the game rounds.

During one round of *Generative Domination*, each player takes one turn where they can input their action, vote on rules and answer pending ally requests. To facilitate this, two main classes, these being the *Canvas Manager* and the *Game Manager*, work together to form a loop where the inputted data is revived, stored and reset for the next player to do the same. Figure 6.7 simplifies this code execution flow and shows the stage where the player does their action inside the game. The *virtual player* system substitutes this step with a virtual player turn, which automatically can produce all required inputs during a player turn. This system was used to create computer player bots that could be added to the game instead of real humans and would simply randomize their actions to simulate a player.

To utilize the *virtual player* system for the simulation of game rounds, a new type of player, called a *deterministic player*, was created. In comparison with the already existing computer player that uses the same system, a deterministic player has predefined actions and instructions automatically selected based on which round it is. The pre-defined data was gathered through a playtesting session where five people played a game for seven rounds before finishing. A log file of all player inputs throughout all seven rounds was then saved and used as round data for the deterministic players. Gathering the data this way allowed for a realistic data set of a real play session to be used as a basis for further game testing and evaluation in an automated setting.

After the system was implemented, an improvement to the virtual player system was made to allow it to handle rule voting. Before this, virtual players could only attack, research and answer ally requests, but after this update, rules could be voted on. Moreover, a bug was also noticed inside the virtual player system when instructions for an action were being set. Instead of it being set correctly, it was being overwritten with an empty string, resulting in only the action going to the AI model. This was fixed such that it sends both the action and its corresponding instruction to the LLM.

## 6.4 Sprint 3-4: Finishing MVP

These sprints focused on finishing up the remaining must-have features. The largest feature of these was to analyze the internal AI system and identify what improvements could be made. It was also decided to change the name from *Generative Domination* to *Evolved Domination* since the game will both evolve from the original and relate to the in-game story evolving over time. It was also decided to call the central LLM responsible for generating new game states the Game Master AI instance.

During these sprints, two noticeable bugs were found in the game. First, a bug resulted in the round history not being included to the Game Master when sending requests, making the AI unaware of data from previous rounds. The bug was found during playtesting when a player tried to perform the same action as in the previous round. The Game Master AI responded to the story with a message indicating that it did not know what action to perform. This gave a clue that something was wrong. The bug was fixed by attaching the round history when sending requests, enabling the use of game state history again.

Additionally, playtesting also discovered another bug where the attached resources were the same for all players. This bug was discovered when the Game Master responded with a story that pointed out errors in the input given to the AI. Part of the story where this is mentioned can be seen below. This discovery identified a coding error that could have remained undetected much longer, showing that the AI can also detect bugs.

> *While 'Prototype Warplane Wings' were listed as 'UsedResources' in the input, this appears to be an error as Player 3 does not possess this resource and it is illogical for a ground attack. We will disregard this and assume Iron Ore was the intended resource for this ground offensive.*

Since the must-have features were almost done, a session was also held planning new should-have and could-have features. Features included GUI changes such as "Show country on hover", "GUI layout per player" and a settings panel. Other features included "Improved context action menu", "Response from rule and ally system", "See country resources" and "Upgraded world shader". Finally, a feature where the AI would select who wins the game was also planned, trying to further incorporate AI decision-making. Moreover, a local AI model called *Gemma 3* also launched and was noted down for further exploration.

### 6.4.1 Investigation of the internal AI solution

As mentioned in Section 5.2, *Generative Domination* used Google's Gemini AI to serve as the central LLM inside the game. More specifically, the model *Gemini 2.0 Flash* was utilized in conjunction with a system instruction to generate new game states. The system instruction explains in detail how the game works and how the model should generate its response, similar to how a rulebook explains board games to players.

This system worked sufficiently, but mistakes were sometimes made. These included inconsistencies between the generated story and the generated game state, meaning that the story could state one thing, but the game state would not update accordingly. One example of this was when a player conquered a country, the story sometimes stated that it was successful, but according to the generated game state, it was not. Another issue was that the LLM sometimes allowed one player to conquer multiple countries in the same turn, which is not permitted according to the system instruction.

With these problems in mind, an investigation for a better system was conducted. Firstly, the model *Gemini 2.0 Flash Thinking* was tested to try out how a reasoning model, defined in Section 3.2, would work. A testing tool *AI studio* was used, mentioned in Section 2.9.1, selected the thinking model, provided the system instruction and gave the AI the chat history of a real game. After processing, the thinking model produced two outputs: the thinking process and the new game state. The thought process outlined the reasoning done by the model and showed that some parts of the system instructions were vague. This indicates that the instruction would need further explanation and clarification to have the potential to result in a more consistent game state. One thing to note was that the model provided an answer slower than a regular model, averaging 15 to 20 seconds instead of 8 to 14 seconds, potentially resulting in longer waiting times in the game.

Secondly, to utilize *Gemini 2.0 Flash Thinking* inside the game, a change in the code was required for how the response is parsed, as the reasoning model does not support structured JSON output. Wanting to retain support for the other model, the code was modified to support both. To improve redundancy, in case the thinking model failed or returned a malformed response, a second request could be sent to the previous model to guarantee a correctly structured response, illustrated in Figure 6.8.

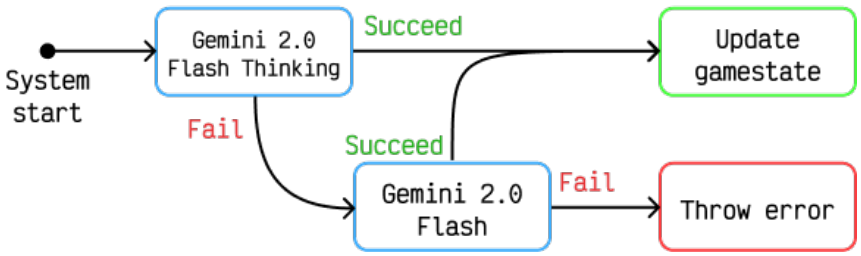



Figure 6.8: Flow of the request sent to the AI. On failure, it defaults to *Gemini 2.0 Flash*.

Thirdly, several improvements were made to the system instruction, such as clearer output format, country definition, clarifications on populating player resources used in the story and more resource use in the story. A self-check was also added at the bottom to force the model to go through the instructions and check if all rules were followed in the generation of the new game state. The reasoning part seemed to understand this and performed self-checks afterwards to ensure coherent game states.

Lastly, to evaluate this, a couple of playtests were done, which yielded positive results. The model produced game states with higher accuracy, and no large inconsistencies were found where players did not receive a conquered country or could conquer multiple countries.

### 6.4.2 Turn validation

To stop players from ending their turn before a certain task has been completed, a turn validation system was created. The goal of the system was to check if the player has selected an action if there are no remaining ally requests or rules to vote on. When players end their turn, the system checks if these three tasks are done. If not, a pop-up, as seen in Figure 6.9, appears beside the end turn button and explains to the player what needs to be done to proceed with the turn. Animations for this pop-up were also created to make the pop-up appear from the bottom and disappear downwards.


Figure 6.9: Screenshot highlighting the turn validation pop-up.

The system works by using a dictionary in the *CanvasManager.cs* class with the key being one of the three tasks and the value being a boolean if the task is done or not. The initial approach thought of using a polling technique where each of the three attributes is constantly checked in an update loop. This was considered inefficient as it had to do processing, although no values in the dictionary were changed. This led to an event-driven approach where the dictionary and accompanying GUI element were only updated when a value changed. The dictionary was moved to another class called *TurnValidation.cs*, containing a function that updates the dictionary and GUI. The function is called from the *CanvasManager.cs* class at select places where tasks change. The three changes observed were if the action instruction is empty, the list of rules to vote on is empty and if the GUI element holding ally request is empty.

### 6.4.3 See and message allies

One of the four core actions players can take is to send an ally request to another player to become allies with each other. This feature was implemented inside *Generative Domination* as explained in Section 2.7.1, but had a few major flaws due to an incomplete part inside the system. Players could send requests to each other and answer them, but after becoming allies, there was no way in the GUI to see with whom they became allies. This essentially deemed the ally system useless for players since it did nothing observable for them other than wasting their time by

sending requests. Internally, however, there existed a system in which all players had their own list of other players whom they were allied to and were sent to the AI controlling the game flow.


(a) State of "Allies" panel from Generative Domination.

(b) The updated "Allies" panel view.

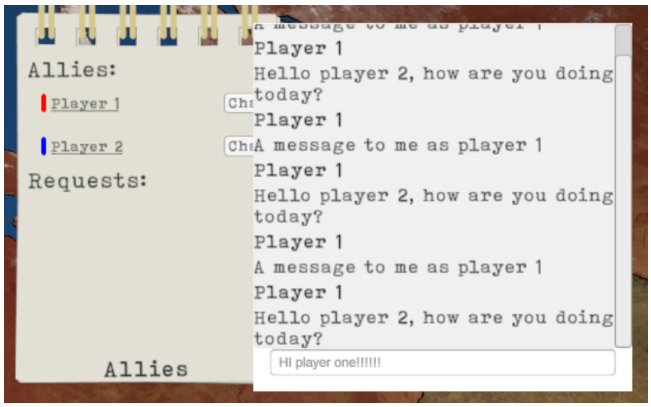
Figure 6.10: The "Allies" panel before and after it was updated to show existing allied players.

Thus, the initial step in enhancing the system was to display the list of allied players already handled internally to the player. Figure 6.10a shows how the *Allies* panel looked before this change, where only ally requests were visible and would disappear after an answer had been given. Figure 6.10b shows the same panel after the change was made, where allied players are shown above the incoming requests. To implement this change, the existing internal list of allied players was sent to the *CanvasManager*, which in turn makes the correct players visible inside the *Allies* panel. The GUI elements are created when players start the game and are simply enabled or disabled depending on the need.

To further increase the ally system's usefulness, a new chat feature was thought up that would allow players to send messages directly to other players they had become allies with. The message system would work on a round-based schedule where messages sent during one round would be received during the next. This mirrored the functionality of the existing ally requests and rule-proposing systems and made it fair for all players, independent of their turn order during a round. Figure 6.11 shows four images over the evolution of the chat system during development from initial rough GUI testing to final design. The GUI components are set up during the start of the game, with the chat box being filled with messages when a player opens the chat.

To sustain this feature, a new internal postal service was created that would receive messages sent by each player during a round and then route all messages to their


(a) Early Figma mockup of chat box.




(b) First working implementation of player chat.


(c) Final graphical elements.


(d) Added colors to player names in chat.

Figure 6.11: Different development versions of the chat UI.

intended recipients at the end of the round. Every player would then have a *Mailbox* to which the messages were sent and stored inside for later use when information is displayed by the *CanvasManager*. The intended consequence of this is that the *Mailbox* stores all chats the player has with other players during a game. The message flow throughout *Evolved Domination* is described in Figure 6.12, where messages go through the *CanvasManager* and the *GameManager* before ending up inside the *PostalSystem*. There, messages are stored until the end of a round, after which they are routed to their corresponding receivers.

### 6.4.4 Refactoring ally request system

When the new message system was implemented, there were thoughts of it being able to be used throughout *Evolved Domination* as a replacement for the current ally request system and rule proposal system. The ally request system from *Generative Domination* worked by allowing the *GameManager* to handle routing and keeping track of active requests during gameplay inside two lists, which were updated accordingly. This worked but resulted in unnecessary complexity inside the *GameManager*. By switching to a message-based ally request system, the *PostalSystem* could be made to handle ally requests instead of the *GameManager*. This

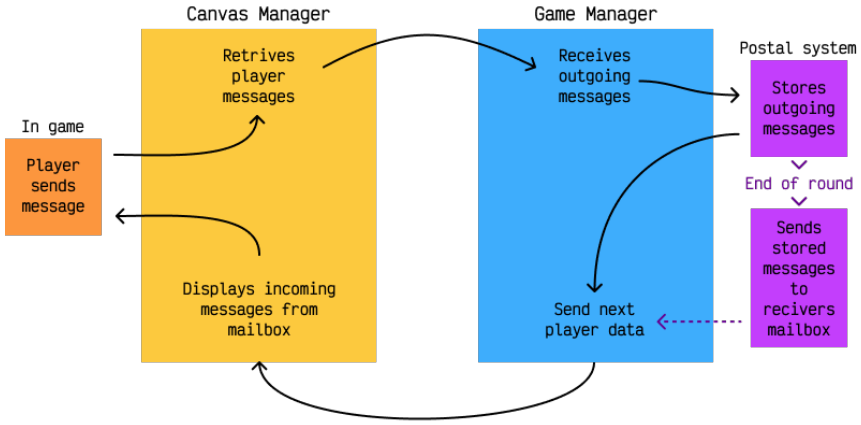



Figure 6.12: Shows the internal flow of the message system.

reduced code complexity and brought the two similar systems together into one cohesive unit.

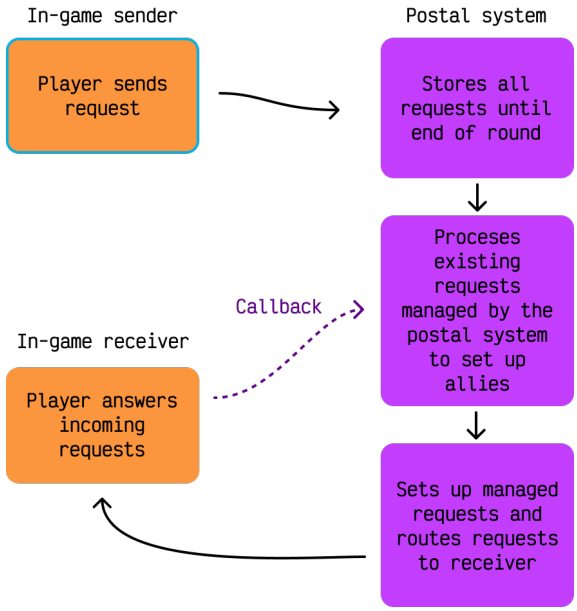



Figure 6.13: Flow of ally request utilizing the new message sending system.

The system worked similarly to the message sending system in that *AllyMessages* were sent under a newly created common *IMessage* interface to the *PostalSystem*. There, all *IMessages* would be stored until the end of the round before getting routed to the receiver of the message. When routing the IMessages, if it noticed an *AllyMessage*, it would inject a callback function into the message, which would modify a boolean value. The *PostalSystem* would then save the modifiable variable and *AllyMessage* inside itself so that the boolean could later be used to determine whether the ally request was accepted or not.

Next turn, when the receiver answers the request, the callback function is used to set the boolean value inside the *PostalSystem* accordingly. Then, before routing all messages at the end of a round, the *PostalSystem* would go through all callback

variables answered during the round and set the players as allies if the request was accepted. The overarching *AllyMessage* flow is depicted in Figure 6.13.


Figure 6.14: An approved ally request using the message sending system.

One noticeable benefit of this approach compared to the old request system was that the player answering the request has time to re-select their response before ending the turn. Before, if selecting "approve" or "deny", the request would disappear directly. With the new system, the answer would instead become highlighted, allowing the answer to be changed afterwards. Figure 6.14 shows the visuals of an approved ally request and exemplifies how a player can easily click deny and change their answer after having clicked approve.

### 6.4.5 Attach resources to actions

The resource system from *Generative Domination* worked by allowing players to see resources they had in the resource panel and use them by mentioning them in their turn instruction. This works but requires players to explicitly write the resource they want to use, which was inefficient. Therefore, a more intuitive approach to specifying resources used inside the instruction was developed. The system allows players to drag resources from the resource panel and drop them in an area on the instruction card, attaching them to the action. To achieve this, several aspects needed development.

First, a drag-and-drop system was created to allow resources to be picked up and moved. This was straightforward, as when the mouse was held down, it would attach to the position of the cursor until the mouse was released. Next, a drop area containing a custom script inside the instruction letter was constructed that would allow players to attach a maximum of three resources to the action. When a resource was dropped on this target, a resource tag was created that displayed the

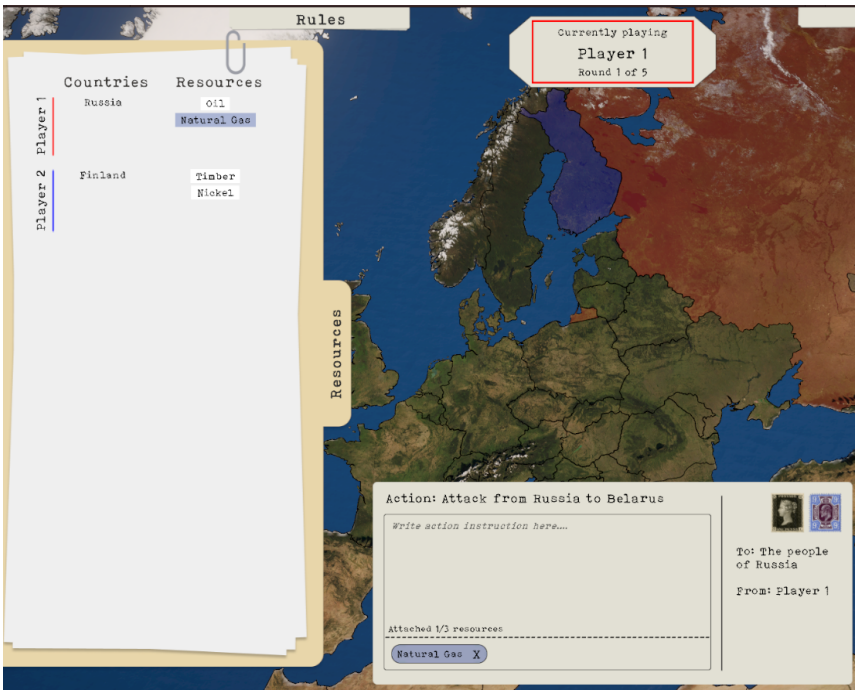



Figure 6.15: Screenshot showing natural gas in blue as an attached resource. The drop area for attached resources can be seen at the bottom of the instruction letter.

resource name and a delete button. Additionally, the resource in the resource panel would change color to the same as the resource tag to indicate that it was attached, as seen in Figure 6.15.

To get the LLM game master to understand how to parse the attached resources, the JSON input data had to be updated to reflect the changes. A new property called "UsedResources" was added to the player object, where attached resources are added to, as seen below. Along with this, the system instruction for the game master needed to be updated, describing that the used resources must be incorporated into the story and not be removed from the player resource list.

```
{
    "Players": [
        {
            "Name": "string",  // Player's name
            "Action": "string",  // Current action taken by the
             player
            "Instruction": "string",  // Details for how the action
              is performed
            "UsedResources": ["string"],  // Resources used by the
             player
            "Resources": ["string"],  // Player's resources
            "CurrentlyOccupying": ["string"],  // Countries
             occupied by the player
            "AlliedPlayers": ["string"]  // Players allied with the
              player
        }
    ]
}
```

Lastly, players were limited to solely dragging and attaching resources they own and not others. Hindering them from using other players' resources in their actions. The final system is showcased in Figure 6.15.

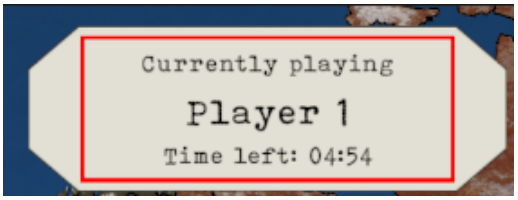
### 6.4.6 Time-based game length

Expanding on the round-based game length, a new time-based game length was added. The time-based length lets players choose how long they want to play the game for before ending the game and displaying the victory pop-up. Implementation started with refactoring the code to allow for multiple different types of win conditions. This meant introducing a new enumeration type called "WinConditionEnum" with the option of round or time-based that would be set in the main menu and received by the game manager upon game start.

After this, the slider UI element in the main menu was copied and configured to change the time limit instead of the number of rounds. To switch between the two options, a custom toggle button was created, allowing players to change from time-based to round-based or vice versa. As the UI style of the main menu was subject to change, no large efforts were put into making the UI polished, as seen in Figure 6.16a.

Lastly, the logic for a timer was added to the game manager class, including displaying the victory pop-up after the timer reached zero and all players had finished their turn. This timer was also displayed in the game and updated for each second that passed, as seen in Figure 6.16b. When finished, the text displays "Time's up!".


(a) Win condition design with a toggle button and a new slider. The "Win condition" name was the original name for this feature before it got changed to "Game length".




(b) Timer displayed in the game.

Figure 6.16: The implementation of time-based game length in the game.

## 6.5 First SWOT revision

After finishing all must-have features, a revision of the SWOT analysis was conducted, incorporating the new findings that had been noted down during the first sprints.

### 6.5.1 Strengths

- Different models can be used for different tasks. For example, simpler models for validating rules and API keys.
- Reasoning gives a higher quality output, especially important when following game rules.

### 6.5.2 Weaknesses

- If an online API is used constantly, players need to have internet access to play the game.
- Developers need to develop games with redundancy in mind. What if the API fails?
- Hard to know who is responsible for the error is when the AI generates incorrect game states.
- Harder to notice bugs as the AI can cover up some of them.
- Reasoning models are slower than non-reasoning.
- Local AI only works properly if players have sufficient hardware

### 6.5.3 Opportunities

- Reasoning can help with more complex tasks.
- The AI can point out if the input received has any errors or the system instruction.
- Local models can be used on powerful systems.

### 6.5.4 Threats

- When using external services such as Google Gemini, the game depends on this and has to change if the services are changed. One example is the list of models that keeps evolving.

## 6.6 Sprint 5-6: Starting should have features

After finishing the MVP features, the development of the should-have features started. Since the AI features “Local AI”, “LLMGA” and “Modifiable game parameters” were deemed important for the project as a whole, these should-have features were chosen to be developed during these sprints. Additional non-AI features were also chosen to be developed this time period.

An important change during this sprint was that a variable called *action succeeded* was stopped being used inside the game code during these sprints that were previously included in output from the Game Master AI. When doing playtesting, the

*action succeeded* variable was causing problems as players received a country they should not have received, according to the story. The story said that the action "Attack Sweden with a creative attack" failed to conquer Sweden, but the variable *action succeeded* was true, indicating to the game a successful attack on Sweden. This might be because the AI thought that it made a creative attack and therefore did a successful action. Because of the potential for error, it was decided to stop using the *action succeeded* variable from the output from the Game Master AI.

Moreover, during development, the API sometimes stopped working, leading to the game not functioning correctly and all AI instances failing. This led to a new feature being planned, named "Feedback when API fails" that was thought to handle these situations gracefully.

### 6.6.1 Local AI model exploration

During development, Google's *Gemma 3* AI model [129] was released at this time. The difference between Gemma and Gemini class of models is that Gemma models are available for people to run locally on their machine, while Gemini is mainly available through cloud services. Since running AI models locally offers several benefits such as increased privacy, constant availability and potentially lower latency, it was investigated as part of this thesis.

For this, a Unity package called *LLM for Unity* [130] was found that allows developers to deploy local AI models within Unity and run inference, which is the process of generating content, in real-time inside games. After downloading, a testing scene was created that allowed for the testing of different model sizes and configurations. Here, it was found that the version of *llama.cpp* [131] used in the package was not updated to support *Gemma 3* models yet. This effectively meant that it was impossible to continue in Unity at this time since *llama.cpp* is the underlying subsystem running the model inference.

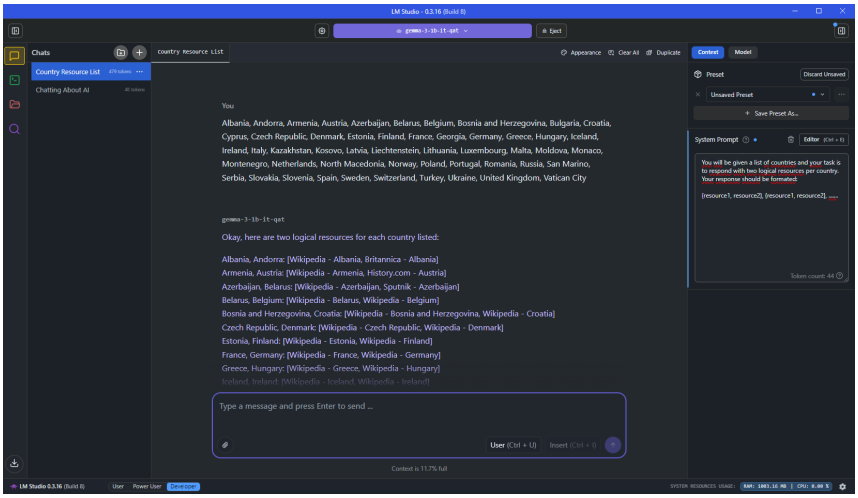



Figure 6.17: The *LM Studio* application running the Gemma 1B model.

Instead, another program was found, called *LM Studio* [132] and depicted in Figure6.17, which utilized an updated version of *llama.cpp* that would allow for *Gemma*

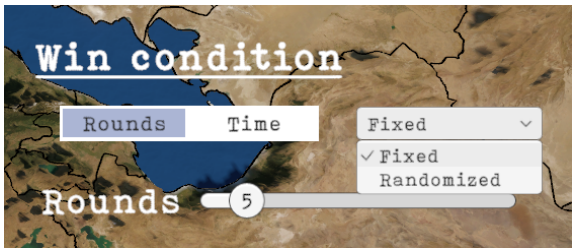



Figure 6.18: The main menu option allowing players to start with randomized countries.

*3* to run. *Gemma 3* comes in different model sizes, with larger models being more demanding to run while having higher quality results. Because the game was partially developed on conventional laptops and not only on higher-performance desktops, the smaller 1B and 4B parameter model sizes were investigated. From this investigation, it was found that using either of the models would be too slow on laptops for any meaningful generation of the AI systems currently implemented.

However, even if they were fast enough, the models still did not perform well in following the system instruction provided. For instance, when provided with the country resource randomizing instruction discussed in Section 6.3.2. Instead of generating two resources per given country as stated in the instruction, the model most often generated zero to three resources, with the resource not being closely connected to the country. This essentially rendered the models useless when in these situations.

One thing realized was that smaller models could possibly be used for generating normal text without following any complex instructions. Such a situation was not pursued further at this moment, but was noted down.

### 6.6.2 Randomized starting countries

Up to this point, to ensure consistent testing, each player has always started the game with the same pre-selected country. This was done to test features more consistently, but also resulted in the game being less varied. A randomization method was implemented in code from *Generative Domination* but was unutilized for the reasons stated. To enable this feature to work, a drop-down field inside the main menu was created where players can select the country selection method as seen in Figure 6.18. If “Randomized” is selected, then starting countries would be randomized, and if “Fixed” is chosen, the countries would be selected as before.

### 6.6.3 Continue game after winning round

On the victory pop-up, discussed in Section 6.3.4, there is an option to hide it, to continue playing. Because the existing functionality was quickly implemented, it resulted in the victory pop-up appearing after every completed round. To avoid this annoyance, simple logic was added to allow the game to continue seamlessly if the victory pop-up is hidden. If players decide to finish the game and get the


Figure 6.19: The "Finish game" button that appears if the game is continued from the victory pop-up.

victory pop-up to assign a new winner, they can click on a "Finish game" button, shown in Figure 6.19, that now appears when the victory pop-up is hidden. This will make the screen appear after the round has been finished, and a new winner will be calculated based on the current score of all players.

### 6.6.4 Converting rule system to use messages

To improve the code quality and consistency in the game, it was decided to convert the existing rule system to use the new message system. The rule system from *Generative Domination* relied on multiple lists such as proposed rules, up for voting rules and active rules with accompanying logic for these, adding unnecessary complexity to the *GameManager* class. The new message system allows extracting this functionality and moving it to the postal system, aligning messages, ally requests and rule voting to use the same system.

The old rule system from *Generative Domination* worked in three stages. First, a player submits a proposed rule, which is added to a proposed rules list. Then, when generating a new round, all rules in that list are sent to a rule validator AI that determines if any of the new rules breaks any already existing rule. If a rule is clear, it is added to the up-for-voting rules to be voted on in the next round. After this, during player turns, the rule progresses to the next stage where it is presented to the players as a pop-up, forcing players to directly vote before proceeding with the turn. The last stage includes counting all the votes from the players and adding the rule to the game if a majority voted for it.

Similarly to the migration of the old ally system to this system, a new message type *RuleMessage* was created. The class implemented the common *IMessage* interface and contained the message, sender/receiver, response function, handled flag and a last vote variable. To allow votes from players to be collected, a callback function was injected into the message, setting the vote to be approved or dismissed, indicated in Figure 6.20. A difference from the ally request and message system was how the messages needed to be sent internally. The rule system was supposed to send out voting requests to all players, not just to one. Consequently, the postal system was adjusted to accommodate this, sending one copy of the rule message to each player. The internal flow of the rule system is illustrated in Figure 6.20.

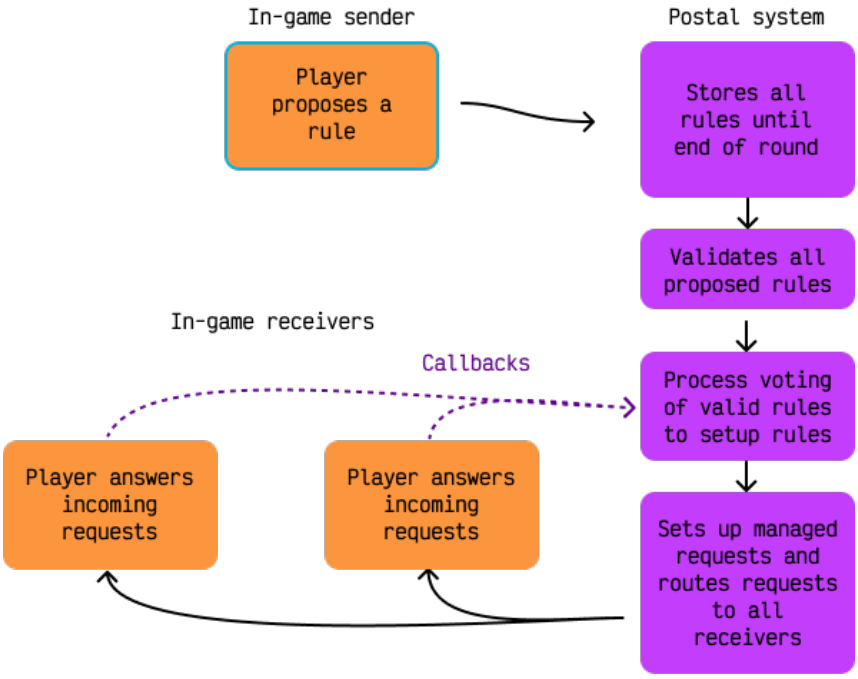



Figure 6.20: A flow diagram of how the rule system works.

All the rule logic, such as validating rules or collecting the results, was then moved from the *GameManager* class to the *PostalSystem* class, reducing the size and complexity of the *GameManager* class. Similarly to the mentioned benefit to the ally request system, last in Section 6.4.4, was that using the new message system allowed players to change their answer at any time during their turn instead of only having one chance to vote per rule at the start of their turn.

Lastly, three smaller changes were made. Rule validation was moved to be done directly after a rule action was performed to receive an answer before sending the game state to the LLM. Before, all rules were validated simultaneously as the new game state was generated. If a rule was marked invalid by the rule validator, this state and the reason why the rule was rejected were added to the generated story through an attachment to the player instruction. An example of this is "Every research should succeed. This rule is invalid because it breaks the fundamentals of the game". Next, rule voting for virtual players was fixed, making computer players have an 80% of voting yes on a rule and deterministic players always voting yes.

### 6.6.5 GUI layout per player

Inside *Generative Domination*, the GUI layout state was the same for all players. This meant that if Player 1 ended their turn with the resource panel open, then Player 2 would have it directly open from the start. To avoid similar situations where the *Allied* panel gets revealed by mistake, a feature was added to save the open and closed state of the GUI panels for each player when they end their turn. Then, when a player starts their turn, the saved state for that player can be used

to set up all panels to that state. This resulted in a system where each player could have their own layout, which would be saved and restored each time their turn came.

To prevent the display of other players' information during the loading screen between rounds, all panels except the resource panel close. The resource panel remains visible to allow players to easily see the standings and observe changes when the new game state is returned from the LLM.

### 6.6.6 Modify game parameters

To allow the LLM to have more ways of interacting with the game than providing a new game state, a new system was devised to expose game parameters that the AI could manipulate. The functionality would be used when new user-created rules are applied to the game, changing either the current round, winning round and adding or removing player resources.

To achieve this, a Gemini functionality named function calls [133] was used to receive output from the LLM in a structured way. Function calls are used when developers want the LLM to choose a suitable function from a large set of functions, along with parameters. For instance, one function was "SetMaxNrOfAttachedResources" with an integer as a parameter to specify the maximum number of attached resources. This was added to allow the LLM to adjust how many resources can be attached to actions.

A new AI instance called *RuleApplicator* was created for this use case, which received an array of all newly created rules for the round and the following system instruction:

> You will receive an array of strings that represent a rule each. Your task is to output an array of function calls that fit the rules.

After processing, the model returns with an array of function calls if it finds any suitable rules that change the game parameters mentioned earlier. If not, the model returns nothing or a special function call named "doNothing".

### 6.6.7 Addition of an LLMGA

To include another more advanced generative AI instance in the game, an LLM-based Game Agent (LLMGA) was implemented into the game. The purpose was to simulate a player capable of performing every action that a normal human player can perform, making it more engaging while playing against virtual players.

The work started by planning the requirements needed for an LLMGA to function as a simulation for a real player. These requirements would be that the game agent should have access to the same information as a real player would and that it needs to have a relatively short response time to avoid unnecessary waiting. It would also need to conform to the existing *virtual player* system that the game utilizes to avoid having separate handling logic. The difference between this *AI Player* and the already existing *Computer Player*, mentioned in Section 6.3.5, would be that

the AI player has an increased action set and is controlled by an LLM rather than just randomizing each action.

The virtual player system had, up to this point, been designed to always generate the player action instantly during the corresponding player's turn. This was well suited for the existing types of virtual players since, in the case of a *computer player*, it would randomize the actions directly and for the *deterministic player*, pick the correct action from predefined turn data. However, for an LLMGA where the response has to be generated, it will take multiple seconds before the AI model can generate its full response. This is disadvantageous since it means having to wait longer for each virtual player to make their turn, which in turn can be boring.

One solution to this problem would be to utilize a quicker model, such as *Gemini 2.0 Flash Lite* to decrease the generation time for the response, but that comes with the disadvantage of having to utilize a less capable model to simulate a human. Another solution that would allow for the use of a more powerful model while still minimizing waiting times was to start the LLMGA turn generation as soon as possible. The response could then be handled internally so that when the LLMGA turn happens, it can retrieve data from the presumably already finished response without having any noticeable waiting time in-game. Since the LLMGA needed to handle multiple things it would handle at once, running it with a more powerful model was desirable. This, in combination with the low waiting times, motivated the latter solution to be chosen.

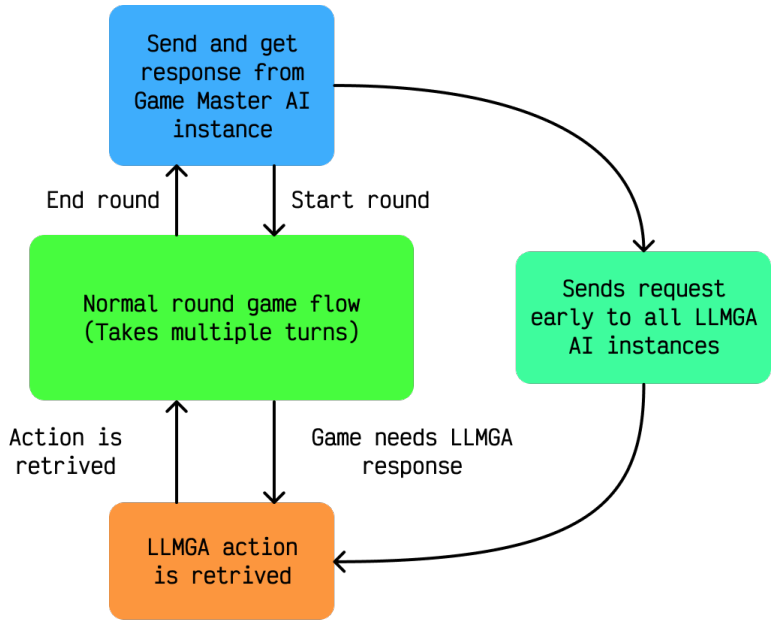



Figure 6.21: The internal flow of the LLMGA system. Requests get sent directly after the response from the Game Master instance is received.

Figure 6.21 shows the internal flow of how data gets sent to the LLMGA right after the previous round response from the Game Master AI. Since the normal game flow usually takes a few turns before the LLMGA response is needed, it usually has finished generating and can just return its response without any delay.

To speed up development and avoid having to write the complete system instruction for the LLMGA from scratch, Google's newly released *Gemini 2.5 Pro* model was used inside *AI Studio* to generate a draft of the LLMGA instruction. This worked effectively, and after some extra prompting to add additional instructions that were first not included, resulted in the system instruction that can be read in Appendix B.2. Further manual tweaks were also made to ensure all necessary instructions were in place.

Likewise, *Gemini 2.5 Pro* was used to help write the template code for the structures used for input and output to the LLMGA. This was achieved by first presenting the model with an example from the Game Master AI instance of how the code corresponds to the described JSON structure inside the system instruction. Then the model was promoted to perform the reverse, taking the LLMGA system instruction JSON description, found inside Appendix B.2, and converted it into code. After making a few tweaks to this code and making sure to set it up correctly, the AI-generated code worked and succeeded in speeding up development time.

At this point, it was possible to send a request to the LLMGA and receive an acceptable response, but it was not fully completed. In addition to the data being sent to the model at every round request, through the JSON data below, static data that would not change from round to round was injected into the system instruction at game start. This included information on how the model should think about game length and what resources every country has. Furthermore, player-created rules are also injected into the system instruction for each API call to maintain live data. This is done similarly to the normal Game Master AI instance, where player-created rules are also injected into the system instruction for each API call.

```
{
    "MyPlayer": {
        "Name": "string",
        "Resources": ["string"],
        "CurrentlyOccupying": ["string"],
        "AlliedPlayers": ["string"]
    },
    "OtherPlayers": [
        {
            "Name": "string",
            "Resources": ["string"],
            "CurrentlyOccupying": ["string"]
        }
    ],
    "TurnNumber": 1,
    "GeneratedStory": "string",
    "RulesToVoteOn": ["string"],
    "PendingAllyRequests": [
      {
        "RequestFrom": "string",
        "Message": "string"
      }
    ],
    "AlliedPlayerChats": [
        {
```

```
            "ChatWith": "string",
            "Messages": ["string"]
        }
    ]
}
```

Since it is still possible for the LLMGA to not have finished its response when it is their turn, a loading screen must be able to appear to show that it is in the process of generating an answer. To do this, the current in-game loading screen was improved to show moving loading indicators at the edges of the screen and have dynamic text in the middle indicating its current process. Figure 6.22a displays the old loading screen, which had a static text message on it, and Figure 6.22b shows the newly added loading screen with white moving loading indicators around the edges and the dynamic text saying it is Player 4's move it is currently loading.


(a) The original simple loading screen inside the game.


(b) The improved loading screen with white moving indicators on all sides.

Figure 6.22: The old and new loading screen compared to each other.

With the loading screen, a few tests were conducted to observe how the LLMGA would act. Overall, it did a satisfactory job playing the game and worked on a basis of mostly using the attack or research actions. It sometimes sent ally requests to others and managed to keep a coherent chat with allied players. If it were asked through the chat functionality, it also proposed rules, but it rarely did it by itself.

Two interesting problems related to the LLMGA were also discovered during testing. Firstly, when the LLMGA lost all its territories, it just stopped making actions and instead only responded with "Game Over" as an instruction. The fix to this was to add a section inside the LLMGA system instruction explaining that it should try to gather resources and take over a random country when facing such a situation. This worked to a satisfactory degree, although it often got stuck doing the research action multiple times before doing an attack. This was, however, deemed acceptable and was kept.

Secondly, when chatting with an AI player through the ally messaging system created in Section 6.4.3, it was noted that the LLMGA did not understand that messages were sent to players between rounds and therefore takes time to be received. This results in the LLMGA assuming that messages are received instantly and therefore occasionally states information that becomes old before being received. Though multiple changes and explanations inside the system instruction were made to fix

this knowledge gap, the problem still persists in the final version of the LLMGA, but to a lesser degree.

## 6.7 Sprint 7-8: Working with should have features

These two sprints focused on finishing the most important should-have features. The main objective was to improve the player experience based on the internal player tests. Features such as "feedback from rule and ally system", "show country resources" and "improved context menu" were therefore developed. Additionally, the features "Settings menu" and "AI selects winner" established in sprint three were also developed.

When playtests were done with people who had never played the game before, it was noticed that they had a difficult time understanding how the game worked. A "How to play page" feature was therefore planned to be developed later.

### 6.7.1 Upgraded world shaders

One area where the game had less focus put on it was the world map. To create the terrain inside *Generative Domination*, a terrain generation system created by Sebastian Lague was modified and used as the main ground terrain. The system originates from inside Lagues *Geographical Adventures* game [134], where satellite images and other height map data provided by NASA were utilized to create a virtual 3D globe players could fly around and explore. For *Generative Domination*, this system was modified by changing the way vertices get mapped to instead generate the terrain on a flat 2D plane, with mountain height being the only 3D component. Furthermore, before the vertices created the final 3D mesh, the vertex positions were also "rotated" around an imaginary sphere so that Europe would be in the center of the 2D plane when the final mesh is created. This results in less map distortion from Europe otherwise being north up close to the north pole. Lague also had an ocean shader inside his game that utilized an ocean depth map from NASA to create different colors based on water depth, but was not implemented inside *Generative Domination* due to time constraints. Instead, a simple colored plane was used together with a normal map to simulate waves.

To improve the world terrain for *Evolved Domination*, two things were identified to be improved upon. The first thing was to update and integrate Lagues ocean shader, and the second was to make countries outside of Europe darker in color to better show which countries are available to attack and possess inside the game.

The challenge of updating Lagues ocean shader, which is shown in Figure 6.23, was that the shader code was written for Unity's Built-in render pipeline instead of the Universal Render Pipeline (URP) used inside *Evolved Domination*. This resulted in the code not being compatible and instead had to be converted to work within the URP pipeline. To do this, *Gemini 2.5 Pro* was prompted to convert Lagues original shader code into code compatible with the URP pipeline. The given response was not

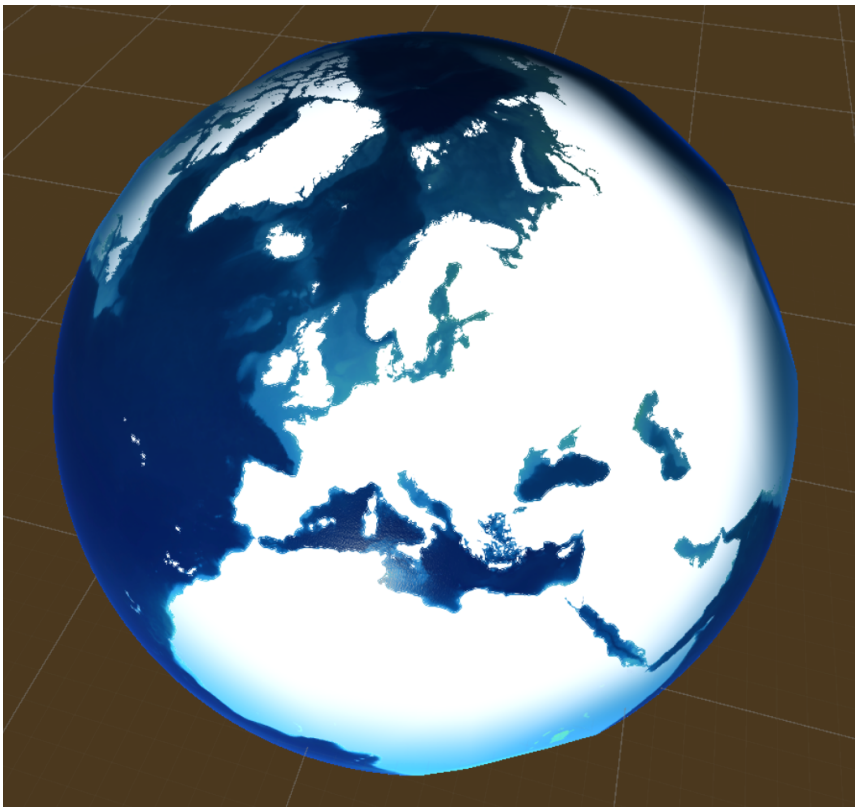


Figure 6.23: Sebastian Lagues ocean shader from his Geographical Adventures game.

a successful conversion after the first iteration, but after more prompting attempts while giving the AI feedback on errors, it successfully managed to convert the code. Figure 6.24 shows a plane with the newly converted shader applied to it inside *Evolved Domination*.

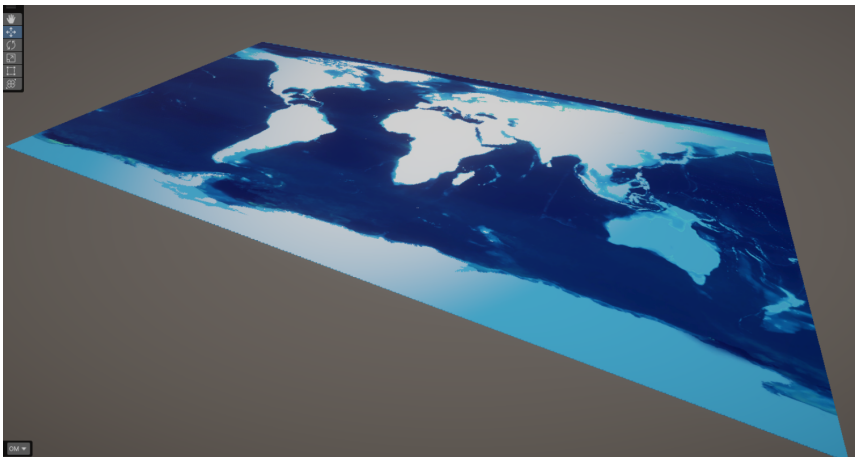
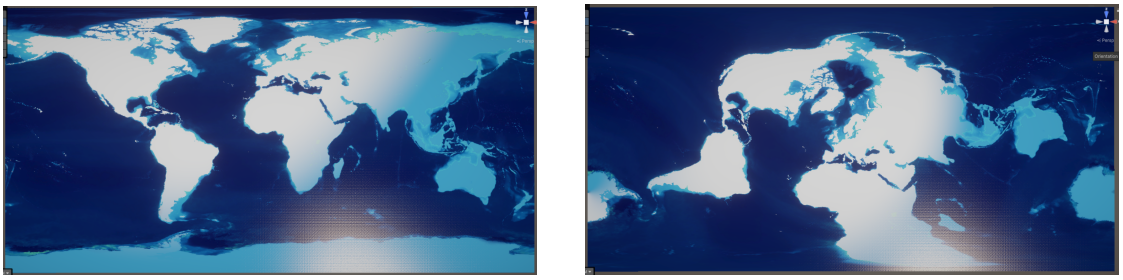


Figure 6.24: The URP converted ocean shader applied to a plane.

The last part that needed to be done was to rotate the ocean similarly to how the terrain had been rotated inside *Generative Domination*. This was done using the same underlying rotation technique used to rotate the terrain during mesh creation.



(a) Ocean shader before it was “rotated”.

(b) Ocean shader after it was “rotated”.

Figure 6.25: The ocean shader applied to a flat plane before and after map rotation centered on Europe.

It projects all map points onto a virtual sphere and rotates them in X and Y directions before reprojecting them back to the map. The before and after images of the rotation can be seen in Figure 6.25 and show how Europe has been centered on the map.

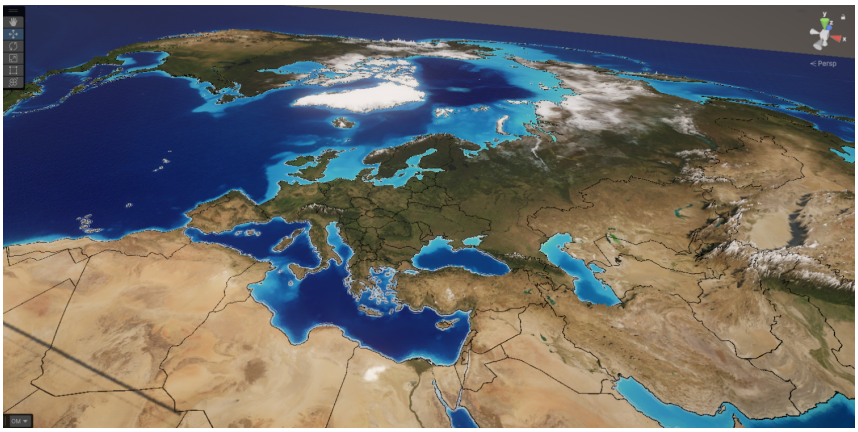
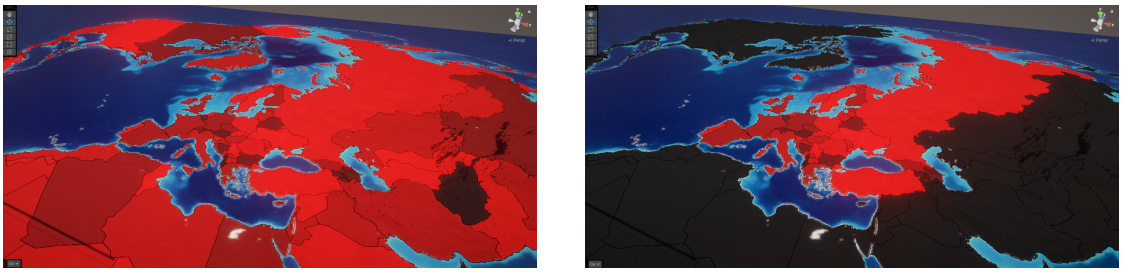


Figure 6.26: Terrain and water shader used together to form *Evolved Domination's* world map.

However, when this was used together with the terrain generation, the shoreline did not match up with the terrain correctly even when both systems were rotated by the same amount. After having compared the HLSL rotation code for the ocean shader and the C# equivalent for the terrain vertex rotation, an error was found inside the C# code that caused it to rotate the X and Y rotations in the wrong order. This caused the mismatch between the two. This was fixed with the resulting world map being shown in Figure 6.26.

With the ocean shader done, focus then turned to making countries outside Europe darker. To do this, a compute shader was created that takes the original country index texture map displayed in Figure 6.27a and creates a new index texture where all countries outside of Europe are dark, shown in Figure 6.27b. The country index texture map is a special texture where each country is colored according to an index value that represents the country. By sampling the color of the texture on a specific pixel, it is possible to read the encoded color representing the country index and,



(a) Original country index texture. (b) Generated Europe country index texture.

Figure 6.27: The original country index map together with the Europe index map generated by a compute shader.

in turn, deduce what country that pixel represents. The newly created texture is therefore used inside the terrain shader to determine which countries are outside Europe, since their color index has been set to 0, and color them darker. The newly created country index map also replaces the original index map when coloring the countries after which player occupies them, but that has no visual change. The final in-game result of all these changes can be seen in Figure 6.28, with a before and after comparison being shown in Figure 6.29.

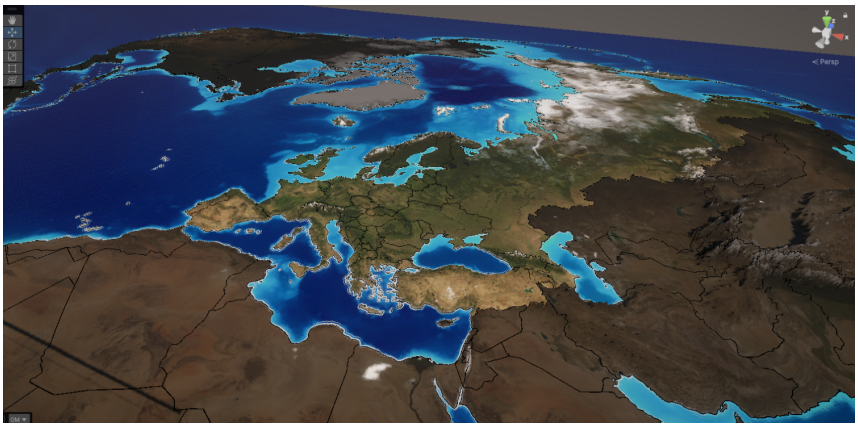


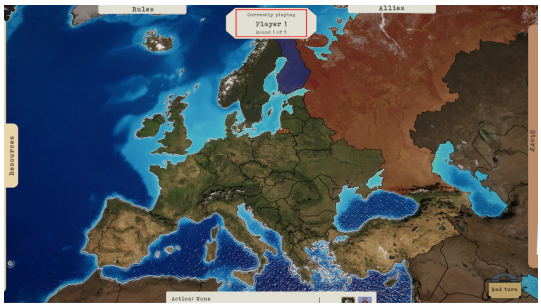
Figure 6.28: The final terrain and ocean shader together.

### 6.7.2 Feedback from rule and ally system

Before this point during development, when rules or ally requests were rejected, the answer was not sent back to the players, causing confusion about whether their request was successful. To improve this, two new message types were created to be used by the message system. For rules, the rejection message was designed to be sent out to all players after a rule did not pass voting. For the ally system, it only requires the player who sent the request to receive a rejection message. This was integrated into the postal system to be handled as messages.

The UI was also updated to support this, seen in Figure 6.30. In the rule and ally panel, a rejection header was added to be visible only if a rejection message has been

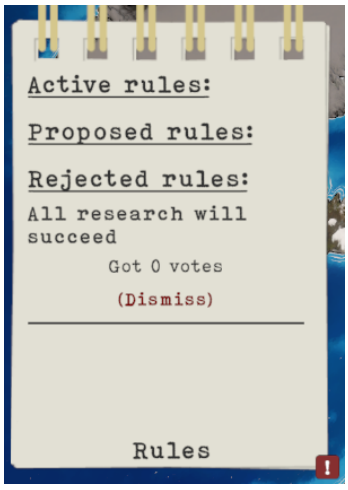
(a) In-game view before all shader updates.



(b) In-game view after all shader updates.

Figure 6.29: Before and after images displaying the visual change of the upgraded shaders used in *Evolved Domination*.

received containing the rejection message. Notification badges were also added to display if a rejection message had been received.




(a) Rule panel showing a rejected rule.


(b) Ally panel showing a rejected ally request.

Figure 6.30: Rejection messages shown for rule and ally panel.

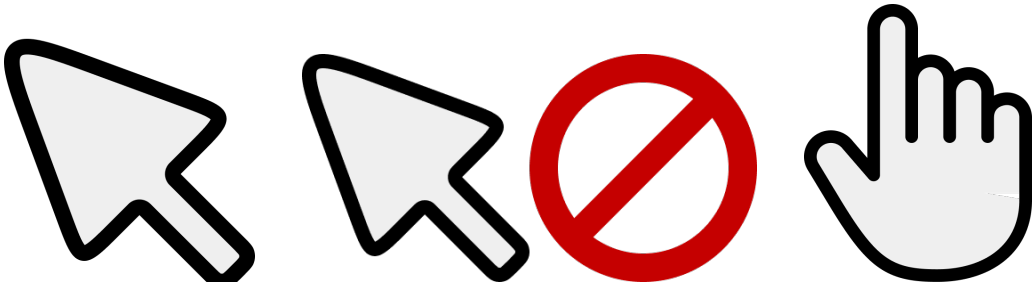
Lastly, a new input field was added to the Game Master AI request to allow for custom events to be injected into the story. This was used to make the Game Master aware of rules that did not pass voting and mention that in the story, giving players feedback in the story of how the voting went. How the Game Master should interpret the events was added to the system instruction as listed below.

> **Events**
>
> You will receive a list of events that occurred in the game last round. These events should be included in the story and can be used to create context for the players' actions.
>
> - Example: "Rule 'Every research action will succeed' was rejected" should be included in the story.

### 6.7.3 Show country names when hovered over and custom mouse cursors


Figure 6.31: Country name tag that appears when the mouse is hovering over Sweden.

To make it easier for players to know which country they are currently hovering their mouse over, a feature was added where a mouse-attached country name tag appears when a country is hovered over. The underlying technique to achieve this uses a similar raycasting approach to the one described in Section 6.7.1, where the country index map is sampled to determine which country index it is. Then, by utilizing the index, it is possible to obtain the country name from an internal list and show it in the graphical representation of the name tag attached to the mouse. Figure 6.31 shows the name tag that appears when hovering over countries.



(a) Normal mouse cursor.

(b) Deny mouse cursor.

(c) Pointer cursor.

Figure 6.32: The different mouse cursors added to the game at this time.

In addition to the country name tags that show up near the mouse, the cursor graphics were also changed and expanded upon. To do this, three new cursor designs were added, as shown in Figure 6.32. Figure 6.32a replaces the operating system default cursor and shows most of the time, Figure 6.32b shows when hovering over the darker countries not available to click on and Figure 6.32c is a pointer hand displayed over UI buttons. The reason for these additions was to make the game easier to understand and show hints to the user about what can and cannot be clicked on.

### 6.7.4 Background music

Since the idea of having dynamic music that reacts to the game was enticing and something planned from the start, a search was done to find how generative AI can be used to generate different types of music. Initially, the idea was a system that would take the story as input and generate music fitting to it each round. Both local models and API for such music generative were researched, but nothing suitable for the project was found. Either the local model was too advanced to run locally, or the API services would require a fee for each generated song which was not wanted. Among this research, a website called *Riffusion* [135] was discovered capable of generating high-quality AI-generated songs both with and without vocals.

Although *Riffusion* did not provide an API to be used directly in the game, the website allowed unlimited generations of songs. To generate music, a prompt was needed to instruct the model on what style, mood, dynamics and more it should use. Instead of generating music dynamically at runtime, the musical approach shifted to creating a large database of generated songs beforehand and using it during the game. Then, an AI could leverage function calls, as described in Section 6.6.6, to select songs from the database based on input received from the player about what music they desire. The selection was done by giving each function a description containing mood, dynamics, instruments, style and tempo. The AI would then respond with approximately five song names to be played in the game.

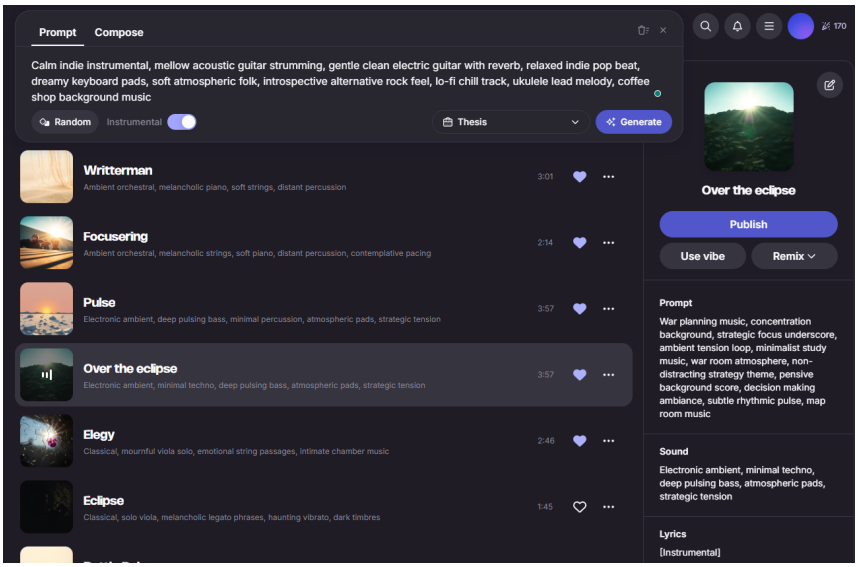



Figure 6.33: The *Riffusion* website used to generate songs by prompting.

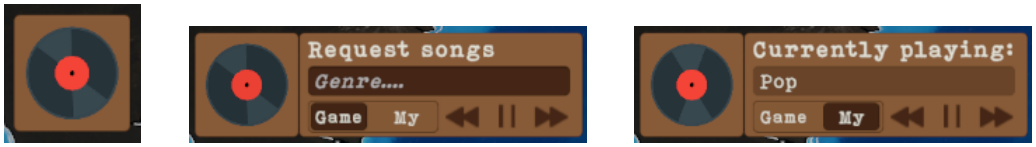
A systematic process was needed to generate all songs. First, a list of 20 genres was created manually as a starting point to try and cover as much music as possible. To use these genres in *Riffusion*, tags that describe each genre were created by prompting Gemini in *AI studio* with the prompt: "Your goal is to give me a list of tags that I can use to generate music. I will provide a list of genres that you should use". Each genre and respective list of tags were inputted one by one in *Riffusion* as exemplified in Figure 6.33, resulting in the generation of two songs. If deemed high quality, the songs were downloaded and saved locally for use in-game.

As mentioned earlier, to allow the music selector AI to make function calls, each song needed a description containing mood, dynamics, instruments, style and tempo. To decide these, *Gemini 2.5 Pro* was used inside *AI studio* by first giving it the song and then following up with the prompt: "Analyze the song by listening to the music file I provided, give me the mood, dynamics, instruments, style and tempo of this song". A structured output was then used to ensure all aspects were covered, as seen in a JSON format below, when the model responded. Parameters for the original and suggested file names were also present to create more descriptive names than those given by *Riffusion*.

```
{
    "songs": [
        {
            "moods": ["string"],
            "dynamics": ["string"],
            "instrumentsUsed": ["string"],
            "style": ["string"],
            "tempo": ["string"],
            "originalFileName": "string",
            "suggestedFileName": "string"
        }
    ]
}
```

A music database with this data was created and read by the game upon start. A new AI instance called *MusicSelector* was then created that uses this data. It generates an array of function calls containing name and description corresponding to each song in the database.

Furthermore, a user interface was created to alter the currently playing music seen in Figure 6.34. The interface allows players to control playback, such as go to the previous song, play/pause and go to the next song. The input field lets players write a description of what music they want to play. Along with this they can switch from music selected by the game and their music by pressing the toggle button.




(a) Closed music panel.

(b) Open music panel.

(c) Playing player chosen music.

Figure 6.34: The different views for the music UI.

Additionally, music for the main menu was added. To create the music tracks, *Riffusion* was once again used together with the following prompt:

> Generate a very slow-paced song with 80s FM synths. It should be mythical, tactical and calculated. Film noir style. No fast-paced snare / kick drums. 60 bpm, minimalistic. Female vocals, being lost inside

> a game called “Evolved Domination” (should be named in the lyrics), writing and writing all day and night, to come up with a cunning plan to take over Europe. Possible action, Attack, Research, Ally and propose rules (these should be accounted for in the chorus). Many possibilities, hard to know what to do.

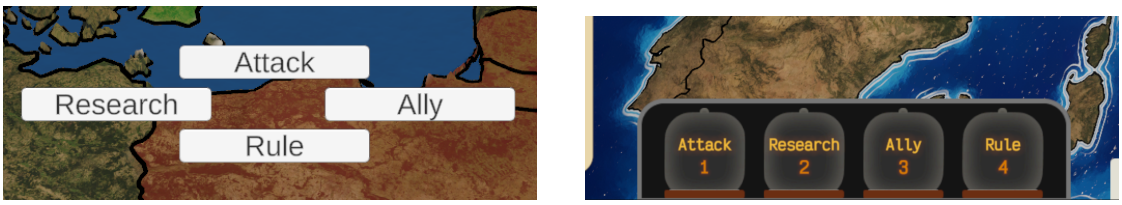
This resulted in a song with relatively good lyrics, seen in appedinx E.1, although both the song and lyrics did not quite fit the wanted theme of the game. *Gemini 2.5 Pro* was therefore used inside *AI studio* to improve the worst parts of the lyrics in combination with manual editing to change the lyrics to a satisfactory state, seen in Appendix E.2. A new song, fitting the new lyrics, was then prompted for inside *Riffusion* with the prompt:

> Female vocals! Melancholy, acoustic guitars, string instruments mysterious, film noir, danceable, interesting, dynamic

After many song generations later, the best was selected and added into the game as a main menu music track. A second instrumental track fitting the lyrics-filled song was also generated so that the instrumental and lyric song could be played in a loop.

### 6.7.5 Improved context action menu

One of the last elements left in a rudimentary state when the *Generative Domination* project was finished was its context action menu used in-game. The context action menu was used for selecting the four in-game base actions and worked by first clicking on an origin country, then selecting what action to take and then, depending if it was an *Attack* or *Ally* action, also clicking on which country to attack or ally towards. The problem with this system was that it was inconsistent between actions and unclear in what selection state it was in. It also inadvertently resulted in it being impossible to become allies with players who had lost all their territories, which was not optimal.




(a) Original context action menu. (b) New action menu.

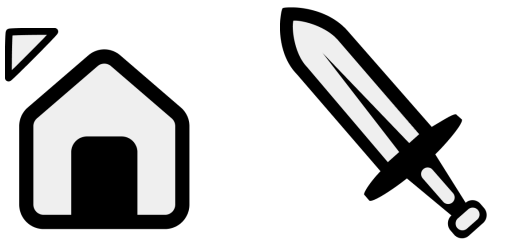
Figure 6.35: The old context action menu compared to the new fixed action menu.

Based on ideas from the project supervisor, a new action menu system was constructed that, instead of only appearing when a country was clicked, was docked in the main GUI and always visible to the player. Figure 6.35 shows the difference between the visuals of the old and new action menu. The benefits of this approach were that the different actions now had the possibility to function differently and not be forced to all use the limited way the old context menu functioned. It also

allowed for keyboard controls to be bound to the actions, which makes them quicker to access than before. Below follows a description of the changes made to each action for the new action menu.


Figure 6.36: The attack pop-up panel when selecting countries to attack.

As mentioned, the previous *Attack* action worked by first clicking a country that the player should attack from, then the old context menu would appear in which the attack option would be selected, and only after that could the country that should be attacked be selected. With the new action panel design, seen in Figure 6.35b, the player can press the attack button and then simply select which country to attack from and to. The standard order is still to select a country to attack from first and then which country the player will attack to. However, either by pressing the TAB key or pressing inside the attack pop-up panel shown in Figure 6.36, it is possible to change the order of the country input. Two new mouse cursors were also made, displayed in Figure 6.37, which makes it easier to know if the player is deciding on the origin point of the attack or the country to be attacked.



(a) The cursor shown when selecting where to attack from.

(b) The cursor shown when selecting country to attack.

Figure 6.37: The cursors visible when selecting countries to attack from and to.

Also mentioned earlier, the previous *Ally* action started by clicking any random country to get the old context menu to appear, then the user presses the *Ally* option, and only after that, selects a country owned by another player. With the new ally action pop-up panel visible in Figure 6.38, it is possible to directly select a

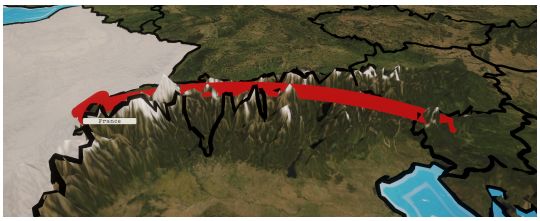
player to send an ally request to after having pressed the *Ally* option. The benefits of this were that it was possible to remove players who were unavailable to ally with. Examples are the current player itself or players that are already allies with the current player. It also allows players to ally with others who have lost all their countries, since there is no country selection as it was before. This update of allying towards players instead of countries required changes in the Game Master and LLMGA system instructions to get those systems to know how to handle the new internal command syntax.


Figure 6.38: The ally pop-up panel.

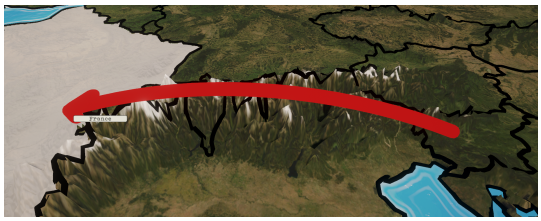
The *Research* and *Rule* actions were previously selected by clicking on a random country and then selecting either of the two actions. With the new action menu, after directly pressing one of the options inside the action menu, the user is taken directly to the instruction letter. This therefore improved the flow and lessened the confusion of having to click a random country first.

Additionally, to allow more dynamic gameplay, a feature was also added that would allow players to click and drag from a country to select an attack vector towards another country that would be attacked. The feature was implemented by dividing the action into three steps. The first being when the button is first pressed, the second being when the mouse is dragged, and the last being when the mouse button is released. These steps represent selecting where to attack from, deciding if the player drags the mouse or if they only click in place and selecting where to attack towards.

To visually show the attack vector currently inputted, an arching arrow originating from the start point towards the current mouse position was added to the game. It was implemented by prompting *Gemini 2.5 Pro* to utilize Unity's line-renderer component inside the generated code, with the constraint being that the script responsible would only get calls during the three steps mentioned in the section above. This worked well, and after a few code iterations, the desired functionality was achieved. The only problem left was that the line sometimes rendered behind higher mountains and hid the arrow, as exemplified in Figure 6.39a. To solve this, instead of using Unity's standard unlit shader, a new shader was created that allowed



(a) The attack arrow when rendered with Unity's standard unlit shader.



(b) The attack arrow when rendered with shader that causes it to render in front of everything.

Figure 6.39: The cursors visible when selecting countries to attack from and to.

the arrow to always render in front of every world object as seen in Figure 6.39b. With this, players could now also choose to drag their mouse from one country to another, as displayed in Figure 6.40, to directly set up the attack action, making it a quicker and more intuitive process.


Figure 6.40: Attack arrow inside the game.

Lastly, the graphical image of the "End Turn" button was updated to be clearer than the phone icon previously used. Before introducing the new visual style with the new action menu, it had been discussed to use a letterbox to represent the instruction letter being posted at the end of the player's turn. However, after having implemented the new visual style for the action menu, an "End Turn" button adhering to the new style was also discussed. In the end, it was decided that both designs would be created and included until further feedback from external user testers could help decide which final design to go with. The two new button designs are shown in Figure 6.41 and were implemented into the game for further feedback and decision.

### 6.7.6 Highlighted player names in the story

During development, it was apparent that players usually skimmed through the story each time new features were tested and did not read the story each time. To make this easier, the players' names were highlighted in the text in the same color as their

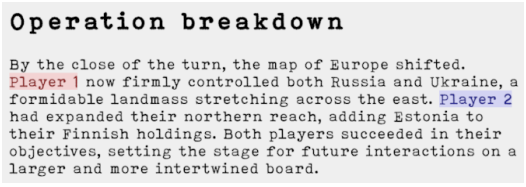

(a) The postal box end turn button graphics.


(b) The end turn button adhering to the new visual style.

Figure 6.41: The two different end turn button designs.

player color. The highlighting can be seen in Figure 6.42. Implementation in the game was done by encapsulating each player's name with an HTML tag specifying the player's color to be rendered for that player's name before presenting it. The feature successfully made finding what happened for each player easier by making the story more accessible to skim through.


(a) Without highlighting.




(b) With highlighting.

Figure 6.42: Showing different versions of the generated story.

### 6.7.7 Settings menu

*Generative Domination* had hard-coded models for the different AI systems, which could not be changed during run-time and made it impossible to play the game if a used model became deprecated. The existing system in *Evolved Domination* therefore requires developers to change the values in the code and rebuild the game, which is not preferred. Instead, a solution to change these models at runtime was decided to be incorporated into the game. This was done by creating the UI elements to select predefined models from a dropdown for each AI system and having a text box to choose custom models, seen in Figure 6.43.

To allow all AI systems to use the chosen model, a centralized location was needed where these were stored. Each system would then read from this location and get its specific model when needed. A new enumerated type called *ExpertAI* was created to define each system with values such as "GameMaster", "LLMGA", "RuleValidator" and "MusicSelector". A static dictionary was then added to the class *GeminiModelId.cs*, which uses the ExpertAI enumerator as a key and the model identifier as a


Figure 6.43: The settings menu displaying the ability to change which model each AI system uses and a volume slider to control playback volume.

string. This dictionary is then read from when making requests in the different AI systems and also updated when the settings panel closes.

Additionally, a slider to control music volume was added to the setting panel to allow players to adjust the listening volume. An internal audio mixer was created to support this with two groups: music and sound effects. This allows for individual volume control and allows for assigning sounds to different channels, as well as future-proofing when more types of audio are added.

### 6.7.8 Show country resources

Since country resources were randomized at game start and LLMGAs' knowledge of these, it was decided to allow human players to also view these resources. To do this, the code responsible for moving the old context action menu, discussed in Section 6.7.5, was repurposed to move a newly created country resource panel, shown in Figure 6.44 and added upon so that the panel becomes smaller when zooming out on the map, and larger when zooming in close.

The country name and the two randomized country resources are added inside the panel when a country is clicked and the panel appears. If the "Attack" button is clicked, the panel will close and automatically select that country for attacking, waiting only for the player to select where to attack from. This required communication between the country resource panel and the action menu panel to be implemented. If the player clicks outside the panel or on the "Close" button, the panel closes.


Figure 6.44: The country resource panel.

### 6.7.9 AI selects winner

At the moment when this feature was created, the winning player inside *Evolved Domination* was only determined by counting which player occupied the most countries by the game's end. An idea was thought of to have the AI decide who wins based on the entire history of the game. A new AI instance was created to respond with the winning player, a motivation for why that player won, a score for each player and why they got that score. The system instruction used for this can be found in Appendix B.8 and describes how the AI should take the history of the entire game as input and determine a winner.

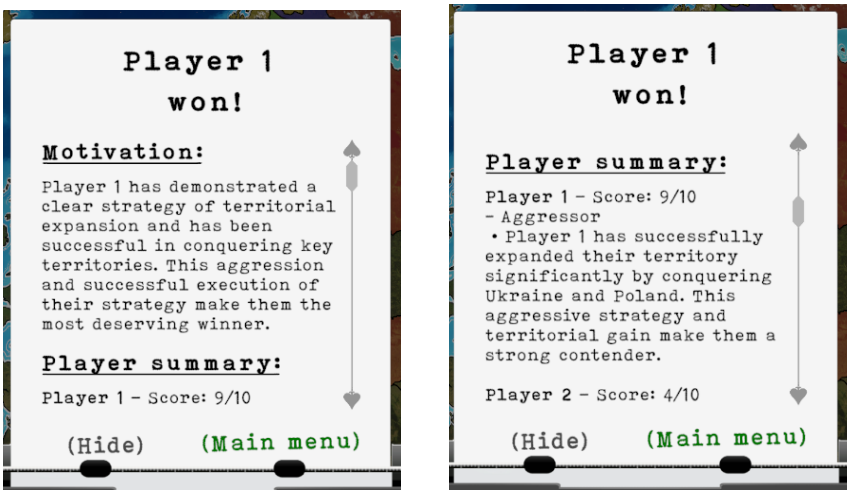
To create the visual elements, the current victory pop-up was modified to support both the most countries and the AI-selected winner. The result can be seen in Figure 6.45.

## 6.8 Second SWOT revision

Following the completion of the should-have feature development phase, a second SWOT revision was done, highlighting new findings. This revision does not include any additions from the preliminary or first revision.

### 6.8.1 Strengths

- Function calls can be used as a decision-maker to call different functionalities in the game.
- Structured output has made receiving consistent output from the AI much easier.




(a) First part showing the winning player and a motivation.

(b) Second part describing player types and score.

Figure 6.45: The two different win conditions.

- Parallelizing the AI players helps speed up the game and avoid waiting times.
- A settings panel to choose AI models to use for the different systems was created. This is to reduce the risk of a broken game in the future.
- LLMGAs can send messages to each other without the player's knowledge to strategize.

### 6.8.2 Weaknesses

- The game needs to send a lot of requests to different components in the game.
- Hard to test LLMGAs. It was possible with the Game Master AI, but with the LLMGA it is even harder since it is making moves on its own, which cannot easily be controlled.
- It is hard to configure the difficulty level of LLMGA. They are often quite overpowered in the game
- Having an LLMGA that simulates a player makes it more difficult to add new features for the player, since it also needs to be implemented for the LLMGA.
- The generated story sometimes contains text in different languages not displayable in the game.

### 6.8.3 Opportunities

- It is possible to accomplish tasks with generative AI that are impossible otherwise.
- Local models were not viable when tested because of inconsistent results, but can be viable in the future to not rely on external services.

- It is possible to save time using AI tools for writing code and system instructions for other AI models.
- New AI models in the future could bring more opportunities. Such as better Game Master (LLM), image generation, TTS and music generation.
- Local models provide more data security as data is not sent through the network.

### 6.8.4 Threats

- Too many API calls can be costly.
- It can be hard to precisely control AI models in terms of how they should behave.
- Cloud AI models can steal user data from players.
- The game industry might shift towards more AI-generated content in the game and rely on fewer humans to do the work.
- Creating images or text via API can run into safety issues if the language is too brutal.

## 6.9 Sprint 9-10: Starting to allow could-have features

These two sprints focused on improving the main menu design and started to polish the game with the introduction of could-have features. Additional redundancy for the different AI instances was also decided to be added. This allows the game to function even if some requests to external services stop working.

### 6.9.1 Feedback when API requests fail

To ensure that the game did not crash if the API for a specific model is down, the concept of a fallback request was used. For instance, if the Game Master API request fails, the game would retry the request with the same data, hoping that the fallback model is available and provides a response. If this request also fails, a retry mechanism or default values should ideally exist.

Implementing such a system required the authors to create a new method in the base AI class *Gemini.cs*. The method first performed a request with the specified model for that instance, and on failure, it tried again with a fallback model. If that were to fail, it throws an exception that needs to be caught by the calling method. When caught, the calling method would then return the default values specific to each AI instance, if applicable. For example, if the country resource randomizer failed, each country would randomly get two resources from a predefined array of five resources. This allowed the game to continue even if a request fails.

The only AI instance not using default values at this time was the Game Master due to its importance. Simple algorithms could not produce the same level of output that the AI could produce, making it difficult to create default values for. Instead, a retry mechanism was created to manually send the request again after pressing the "Retry" button in the UI shown in Figure 6.46. Additionally, the UI also displays the error of why the request failed as text in the center of the screen to allow for more feedback to players.

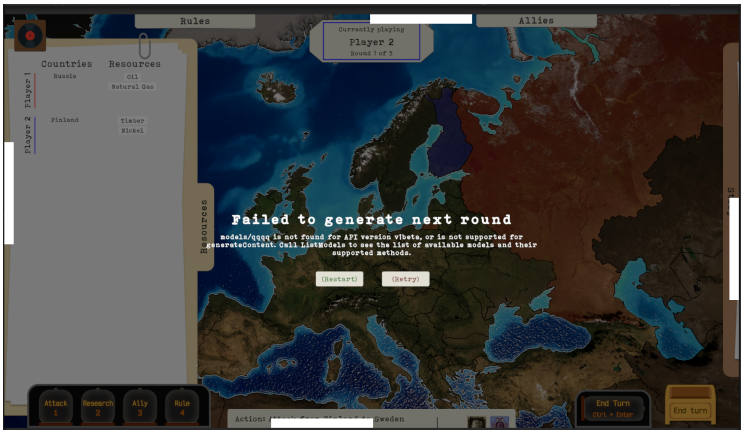



Figure 6.46: Image showing the UI when a game master request fails.

### 6.9.2 Victory music

At this point during development, no winning sound was played on the victory pop-up. To change this and make a victory feel more exciting, music was added when showing the screen. *Riffusion* was used to generate 18 victory songs with the following prompts:

> Winning music. Trumpet fanfare, uplifting music. Old war style. brassy, loud and melodically direct announcement of success.
>
> Trumpet fanfare, brassy, loud and melodically direct announcement of success, Majestic & Grand

Out of these, only five were deemed high enough quality and had a fitting victory theme. The others were rejected because they were slow, calm and minimal in the beginning, not contributing to a celebratory feeling.

### 6.9.3 New main menu

After recently having updated some of the more unfinished graphical elements inside the game, such as the action menu updated in Section 6.7.5, the unfinished main menu, seen in Figure 6.47, was also finally revised. At this stage, the menu still had the old name "Generative Domination" and included options randomly placed to the right of the screen.

The new menu design process began with the creation of a few mockups in the design program Figma [136] of how the menu possibly could look. These were done quickly


Figure 6.47: The main menu as it was before updating the menu.

in low fidelity, following the prototyping ideas from Section 4.5.3, by drawing simple shapes of how the menu could be structured. A planned "World Setting" function was added to the design to be implemented later in the process. The mockups are shown in Figure 6.48 and helped inform the next stage of designing.

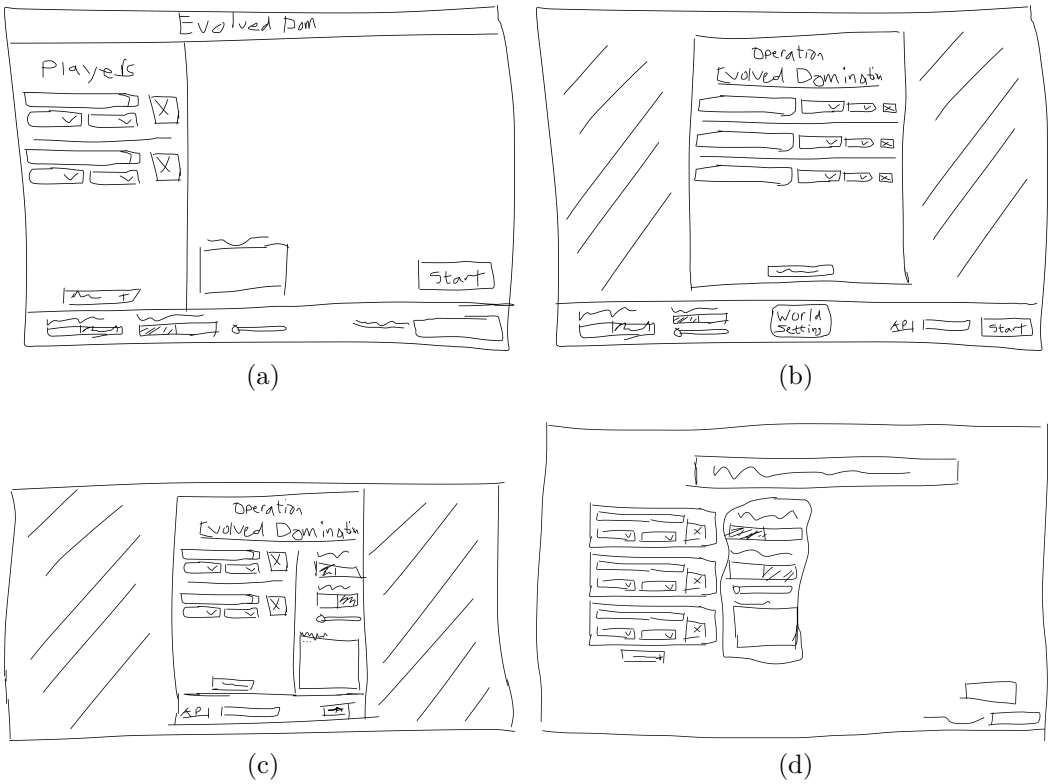
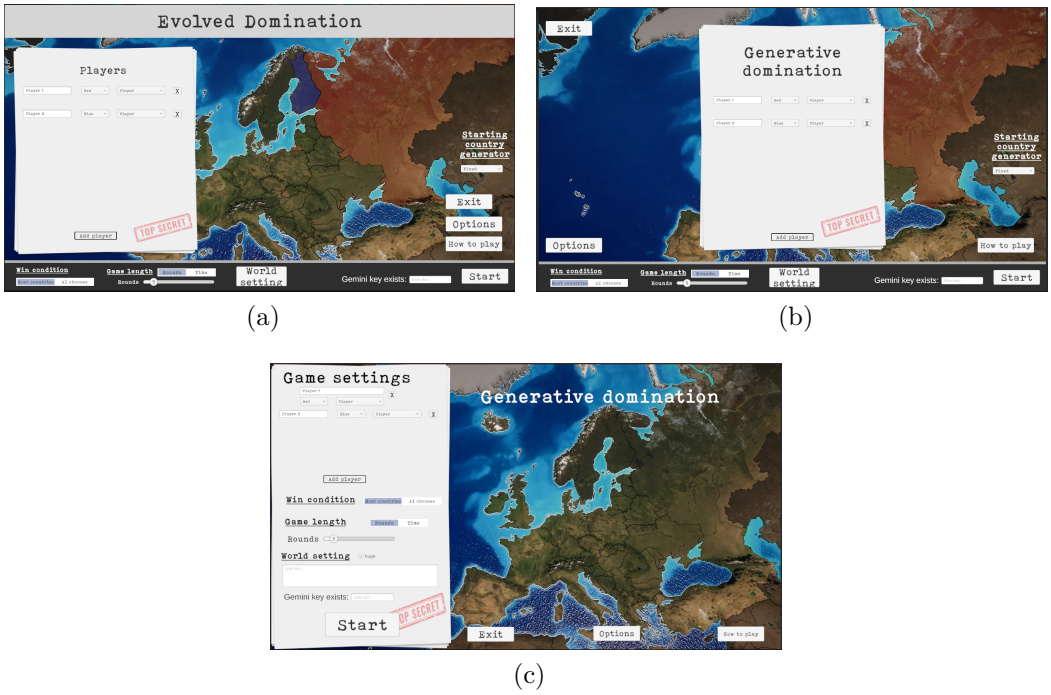



Figure 6.48: Low-fidelity Figma mockups of the new potential main menu designs.

The next steps were to quickly prototype the most interesting Figma [136] mockups from before inside Unity. This was done by modifying the existing menu to drag the existing components to their new prototype positions and, in certain cases, adding

additional elements to complete the layout. The mockup design shown in Figure 6.48a became the design in Figure 6.49a, the design from Figure 6.48b became the prototype in Figure 6.49b and the mockup in Figure 6.48c underwent minor changes to become Figure 6.49c.




(a) (b)

(c)

Figure 6.49: Higher fidelity prototypes quickly made inside Unity.

Out of these prototypes, the one shown in Figure 6.49a was selected for further development as it did not cover the Europe map in the background as the one in 6.49b and was not filled with too many options inside the paper menu panel which the design in 6.49c was. The visual style from the new action menu, written about in Section 6.7.5 was used as an inspiration for the final main menu design displayed in Figure 6.50. Things to note on the new main menu are that the "Start" button is displayed in red before the API key has been verified and becomes yellow if it succeeds. The "Starting country generator" option was planned to be removed and therefore shown in its old style, and buttons for "Options" and a planned "How to play" page were added.

Together with this visual update of the main menu, the settings menu was also updated to fit this new design language, including more orange accents. This new design is shown in Figure 6.51.

Lastly, three more mouse cursors were added when an object is grabbed and dragged inside the game and when hovering over a text field. When hovering over objects marked with a certain tag, the cursor will be switched to the corresponding cursor.


Figure 6.50: The updated main menu inside *Evolved Domination*.

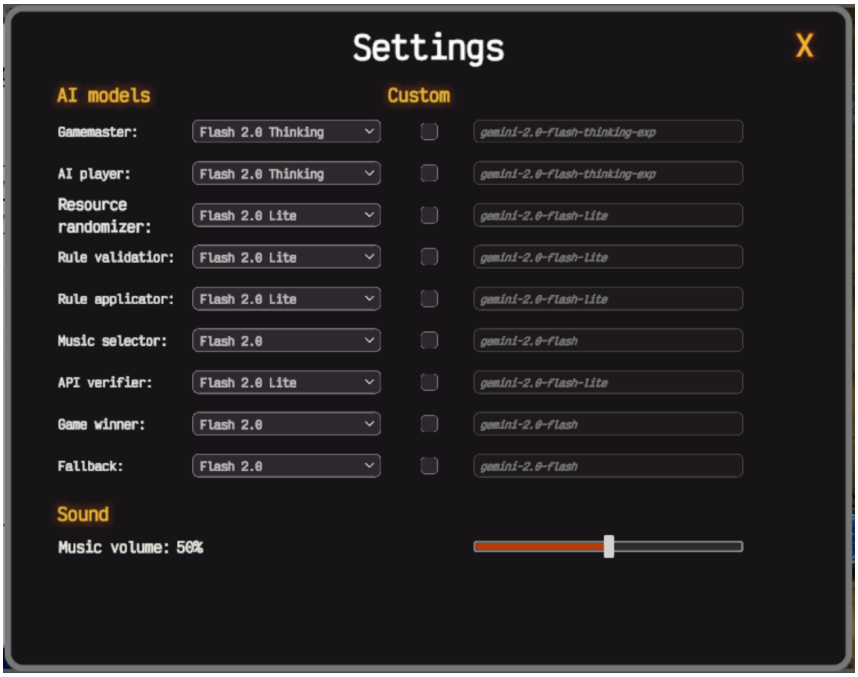
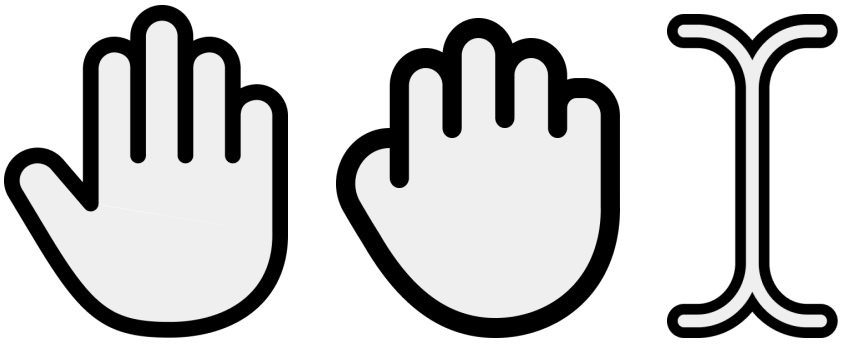



Figure 6.51: The updated settings menu after the main menu had been updated.

### 6.9.4 How to play page

As mentioned in the previous section, a "How to play" page was added to the game, which helps new players get introduced to the most important aspects of the game. The how-to-play page was first written in a separate text document with these main headers: *controls*, *actions (attack, research, ally and rule)*, *all panels*, *attach resources* and *music selection*. When the text was finished, it was transferred to the game with highlights for emphasized elements. Images were also added to support the text and visually show the players how elements look in-game. The final design is shown in Figure 6.53.



(a) Open hand cursor. (b) Grabbing hand cursor. (c) Text carot cursor.

Figure 6.52: The grabbing cursors and the text cursor.

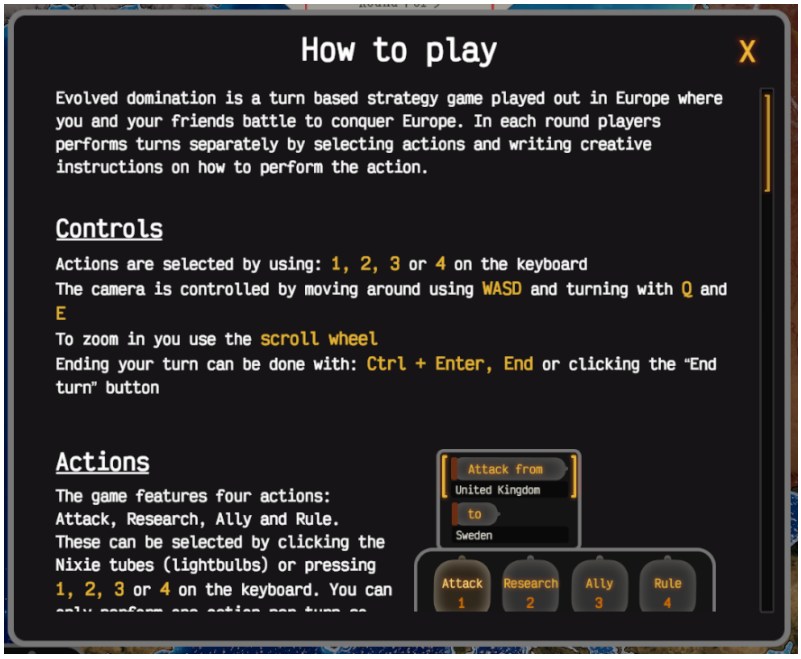



Figure 6.53: How to play pop-up inside the game.

## 6.10 Sprint 11-12: Finishing development

As the development time concluded after these sprints, they focused on adding the last features and polish to the final game. The focus was put on getting the last large features done such as local instruction generation, image generation, Text-To-Speech (TTS) and world setting. The two features, image generation and TTS, had been postponed until this point due to waiting for more advanced models, and the others were deemed important enough to be implemented.

Two other smaller changes were also made during these sprints. First, the model temperature for the country resource randomizer was changed to a value of 1.7 instead of 1 to increase resource randomness. The other change included upgrading the model for the Game Master and LLMGA instances to use Google's then newly

released *Gemini 2.5 Flash* instead of *Gemini 2.0 Flash Thinking*.

### 6.10.1 AI generated instructions

Building upon the local AI exploration in Section 6.6.1, it was decided to use local AI to help players write the action instructions in-game due to them being relatively short. Implementation was done by first updating the *LLMUnity* package to the latest version, which now had native support for running Google's Gemma models. Since *Evolved Domination's* target hardware is laptops, the model chosen was *Gemma 3 1B* to get a reasonable inference speed.

After running tests with the examples provided in the LLMUnity package, the necessary scripts *LLM.cs* and *LLMCharacter.cs* were added inside the game. To call the model, a new *InstructionPanel.cs* script was added. In the file, a simple system instruction was created to tell the model what to do and how to respond:

> You will be given the players information and state of the game and you will generate an appropriate instruction to perform the action.

This instruction gave a broad range of output unsuitable for an action instruction. The output included starting phrases such as "Okay, lets attack!" or a question asking "What kind of attack would you want to execute?". To improve this, the system instruction was reformulated and the following player information was added: *occupied countries*, *resources* and the performed *action*. The new system instruction, seen below, was more detailed, highlighting available actions and clearly describing how it should behave. This gave a more suitable response from the local LLM.

> The game is a turn based strategy war game about conquering Europe where you can attack other countries, research new technologies with current resources and send ally requests to other players. You will be given an action a player has performed and what resources they have access to. Your goal is to act as the player and generate a creative instruction of how the action is performed. The instruction should be creative and written in first person. You don't need to use all resources. Limit the response to 50 words and give me only the creative instruction, DO NOT recognize or answer the user directly. If you receive a starting instruction, use it as a basis when you generate your response!

When this was done, the UI was modified to add a button on the instruction letter. The button is shown in Figure 6.54 and changes to the text "Generating ..." when the local model is generating or printing a response. The printing is done word by word while the model is generating. While a structured output from the model was tested to ensure consistency, it was deemed not useful as the response needed to be fully generated before it could be parsed, therefore, disrupting the word-by-word printing used. When printing, typewriter sounds are played for each word that is printed and a bell sound is played when done.


Figure 6.54: The instruction letter including the button “Help me write”.

### 6.10.2 Image generation

Image generation began with identifying which model to use. First local models, such as *Stable diffusion* [137], were researched but ultimately found too demanding to be run locally on a laptop. After that, cloud models such as *Gemini 2.0 Native Image Generation* and *Google Imagen 3* were tested using an API [25]. When prompting the models, the instruction was “Generate an image that captures the events in this story. This should not be a map or contain text” followed by attaching the story for a specific round in the game.

*Gemini 2.0 Native Image Generation* was free to use but mostly generated images of a map or images that contained text, as seen in Figure 6.55a. Different prompts were tested but still mostly generated undesired results. Later this model was also restricted for use inside the EU, making it impossible to use.

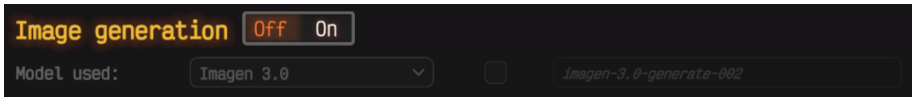
*Imagen 3*, on the other hand, was a paid model with the cost of $0.03 per image but produced significantly better images as seen in Figure 6.55b. It still generated images containing maps or text but to a lesser degree. Since the quality was higher and no better options were found, it was decided to use this model inside the game.

(a) Native image generation using *Gemini 2.0 Native Image Generation* showing a map over Europe.

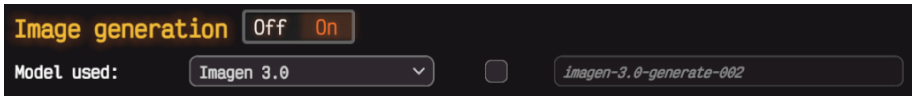
(b) Image using *Google Imagen 3* depicting a war with cannons on a frozen field.

Figure 6.55: Initial images generated during the research phase.

The in-game implementation of this feature started with creating a new class called *ImageGeneration.cs* with all request logic, since the package *UGemini* used for all other AI instances did not support this model. The class handles constructing the request body containing the prompt used, sending the request using a REST API, receiving a response, parsing the response and lastly converting it to a sprite. The parsing is done by taking the base 64 encoded image data received from the model response, converting it to a byte array, using that to create a texture and then converting that to a sprite. This was a relatively straightforward process, with the majority of complexity originating from the conversion. Since the model cost money for each use, the feature was implemented as an opt-in feature where players had to manually enable it. This was done in the settings panel seen in Figure 6.56. The setting section also allows selecting a preferred model to use when generating images if Google releases future image models.




(a) Image generation off with model settings partially visible.




(b) Image generation on with the ability to select preferred model.

Figure 6.56: Section in settings panel showing image generation turned on or off.

To incorporate the image generation into the game loop, it was decided to have it shown after the story has been generated when players are reading the story. This meant that the story could be read normally without interruption and that the image generation worked as an addition to give players a more multimodal story. When the image is generated, it animates outwards from behind the briefcase to the right and rotates a small amount to the right.

Initially during model testing, it was attempted to input the entire story as described earlier, but the model sometimes generated unwanted images, such as random maps or text. Therefore, it was decided to let the Game Master AI generate an image generation prompt in addition to the normal story and game state it generated before. To achieve this, the response structure for the game master was changed to include the field "StoryImageGenerationPrompt" with the following description in the system instruction.

> - Select the single most engaging/interesting action/conflict inside the story and generate a detailed image generation prompt for it.
> - Give a detailed description of the event while using the overarching format: "A adjective noun in a setting with characters."
> - The prompt must be good for the Imagen image generation model.

The result can be seen in Figure 6.57, where the image appears to the right of the story pop-up and can also be expanded by clicking on it. The internal prompt generated from the game master for this image was: *"Massive, steampunk-style*

*cannons firing volleys of potatoes over a border towards a castle, under a clear sky, with soldiers in the background.".*


(a) Generated image to the right of the story panel.


(b) Expanded image shown over the story panel.

Figure 6.57: UI implementation of the generated images.

In addition to adding the image to the pop-up, the image was also added to the right story panel so players could see the generated image even after closing the pop-up, seen in Figure 6.58.


Figure 6.58: The generated image attached to the right story panel as a photo.

Lastly, a toggle option was added inside the Unity editor that enabled the game to save all generated images as PNGs to the computer. This way, every image paid for

when generated is saved and could be viewed outside the game. A few examples of images that have been saved can be seen in Appendix D.

### 6.10.3 World setting

The world setting feature was developed to change the environment where the game takes place. For example, if the world setting is "The game should take place on the alien planet" the game should use this as context when different AI instances generate content.


Figure 6.59: The world setting pop-up.

To implement this, a pop-up was first created in the main menu, seen in Figure 6.59, where players can input their desired instruction. To open the pop-up, the world setting button in the redesigned main menu was used. After that, the world settings were added as a variable in the *GeminiConfiguration.cs* class for it to be injected into five different AI instances shown in Figure 6.60.

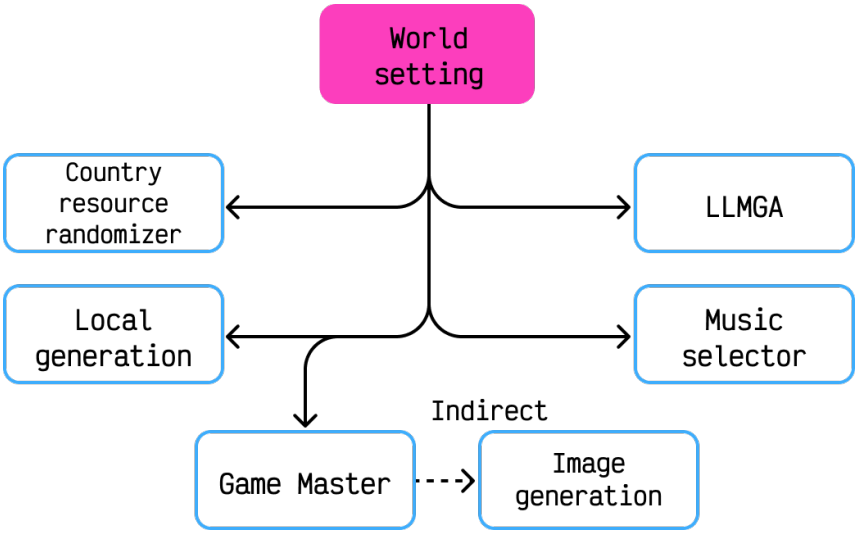



Figure 6.60: The different AI instances the world setting gets injected into.

The world setting was injected by first adding a description of how the five different AI instances should behave and a mark, "#||#", where the injection should happen inside the system instructions. These additions can be seen below for the Game Master, LLMGA, local AI, country resource randomizer and the music selector.

**Game Master:**

> **World Setting**
>
> The game is set in a fictional version of Europe, where players control countries and engage in strategic actions. The world is characterized by a mix of historical and fantastical elements, with unique resources, cultures, and conflicts.
>
> The player-provided world setting below should guide the narrative tone, environmental details and thematic elements of the story. However, if any aspect of the world setting conflicts with the game's rules or mechanics, the game rules take precedence. For example, if the world setting suggests that countries can be partially conquered, the base rule that "Players take control of the entire country, not parts of it" still applies. When generating an image prompt you should also take this world setting into consideration. If the player-provided world setting is empty, you should ignore it and not take it into consideration when generating a response.
>
> **Player provided world setting**
>
> #||#

**LLMGA:**

> **Additional world information**
> The following information is provided to help you understand the world setting. Use it in the generated response to be involved in the world and make the actions fit it. If empty, simply ignore it.
>
> * #||#

**Local AI:**

> The world setting is: "GeminiConfiguration.GetWorldSetting()"

**Resource randomizer:**

> **World Setting Consideration**
> Below is the optional player provided world setting. When generating resources for countries, align your choices with this setting's themes, environmental factors, and technological level. If a world setting is provided:
> - Generate resources that would logically exist within the described world
> - Consider any unusual environments, magic systems, or technology levels mentioned
> - Ensure the resources fit the narrative tone (fantasy, sci-fi, historical,

etc.)
- Create resources that might drive conflict or cooperation based on the setting

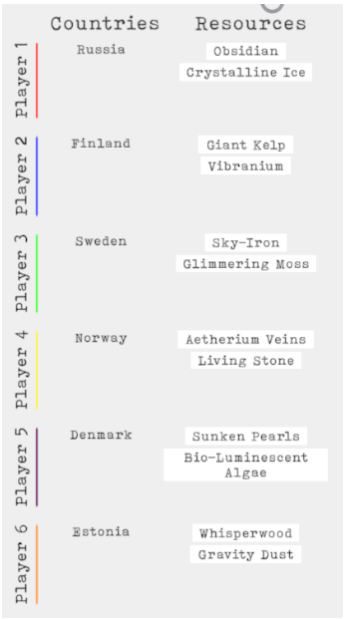
**Player provided world setting**
#||#

**Music:**

**World Setting Consideration**
Below is the optional player provided world setting. When generating songs, align your choices with this setting's themes, environmental factors, and technological level. Ignore this setting if no world setting is provided.

**World setting**
#||#

When all system instructions were modified, the game was tested. First, the world setting "The game should be played out on an alien planet called Strativarius where no humans live". This resulted in some alien resources and a story that fit well with the world setting, seen in Figure 6.61. When testing, this ability to completely change the game was found to be exciting and useful for players who wanted to tailor their experience to a specific scenario.




(a) Generated resources.


(b) Part of generated story.

Figure 6.61: Resources and story generated with the world setting "The game should be played out on an alien planet called Strativarius where no humans live".

### 6.10.4 Text-to-speech

Before development started on *Evolved Domination*, Google had released a video [138] over their upcoming Text-To-Speech (TTS) model that would allow developers enhanced control over the expression the generated voice would have. This extra control arguably made the voice sound more realistic than traditional TTS systems, where the system often speaks in a continuous monotone style. This sparked the concept of an engaging, realistic in-game TTS system that could read out loud the in-game story in an engaging, realistic way and be a part of the game. Google stated that this model would be generally available to developers in early 2025, but that turned out to be incorrect, since it was not released publicly before development stopped in May.

Therefore, throughout the development of *Evolved Domination*, when other TTS models were released similar to the one Google showcased in its video, they were tested to see if they could be utilized instead. The first of these models tested was *Sesame's* demo of their conversational TTS model [139]. Their website showcased a great example of their model in action, but when they released it publicly, they decided to release the smallest version of the model. This model was tested inside the popular AI model provider Hugging Face [140] and was found to be much worse than the example found on *Sesame's* website. This, therefore, resulted in it not being used for *Evolved Domination*.

The second model tested was *Nari Labs Dia* TTS model [141]. Likewise to *Sesame*, *Nari Labs* had a website showcasing its model with comparisons to other models with even better quality than *Sesame's* model. Since the model was not able to run on Hugging Face directly, the Dia model had to be downloaded locally and run via Python to generate testing voices. Compared to the example voices on the website, this local model produced voices of the same quality. However, for a sentence to be generated, it took upwards of 300 seconds for the model to respond since it was running on a laptop. This was deemed too slow for any real use, and in combination with having to run the model in Python compared to C# inside Unity, it was decided not to use this model either.

After still not having found a suitable TTS model to use towards the end of development, research was done into Google's existing TTS solutions. It was discovered that there existed a solution called "Google Cloud Text-to-Speech" [142] that could be accessed through a paid API that was still free to use as long as no more than 1 000 000 text characters are converted to voice per month. This was not optimal since it required players to set up a paid account inside Google Cloud just to use this API, and was more cumbersome than getting an API key for the already used Gemini API. It also does not allow for expressive voice controls such as the previous models mentioned and is instead a traditional TTS system where the voice generated reads the text in a monotone way. However, because there was no time left to wait before this feature had to be made, it was decided that this API would be used.

To implement this API, a C# package called *Google.Cloud.TextToSpeech.V1* [143] was utilized, which abstracts the direct calls to Google to make integration easier.

One difference between the Gemini API compared to this TTS API is that it requires more information than an API key to access its features. The recommended approach was to set up OAuth 2.0 credentials [144] and use the API through it. However, since it was intended that players themselves could access the APIs with their own credentials, this way of authentication would be too complicated to implement effectively. Instead, a solution was implemented, where players would have to download a JSON credential file from Google Cloud and input its filepath into the game. The content of the file was then read and used to authenticate the API use.

For faster development, *Gemini 2.5 Pro* was prompted inside *AI Studio* to provide a script that inputs a given text to the cloud TTS model and takes its audio data response and generates an audio clip that can be played inside Unity. The returned script worked but required the voice name and language code to be passed to the API in conjunction with the text that would be converted to sound.

Since the goal was to create a voice that exceeded the limitations of a typical single-voice TTS system, a system was created to allow different voices to be heard with different dialects depending on where the story takes place. To accomplish this, a new Gemini AI instance called "Voice Selector" was created to take each paragraph from the entire story and assign it a voice and language dialect. It would do this similarly to the music selector using function calls to select voices stored inside a voice database and uses its data when sending requests to the API. Further details about the voice selector AI instance can be found in Section 7.2.11, where descriptions of input and output are found.

Since the API is paid if more than 1 000 000 characters are sent per month, a conservative approach was taken to limit the number of characters sent during gameplay. This was done by adding a button to the story pop-up panel, displayed in Figure 6.62, where players can press it to begin generating voices for that page. This ensures that the system only generates audio for the pages the player wants to listen to. Once the audio is created, it is saved for later playback, eliminating the need for repeated generation requests.

In total, this means that when the TTS button is first pressed and a new story has been generated, it decides on voices for all paragraphs inside the story. It then generates a voice for every currently shown paragraph inside the pop-up panel. If the player presses the TTS button again on another page, it only generates the voice audio since the type of voice has already been decided. This logic flow is shown in Figure 6.63.

Up until this point, the story on the pop-up page was limited to showing only one story paragraph on each page. This was acceptable for the TTS system since only one paragraph of audio would be generated, but it resulted in many pages for the player to go through. Sometimes, a paragraph could also be too long to show on a single page, which causes the text to go outside the intended text area. To fix both of these issues, paragraph splitting was added to the story pop-up panel.

Paragraph splitting was constructed so that the page is filled continuously with one paragraph at a time until it either reaches a buffer zone around the intended page


Figure 6.62: The visual look of the TTS button inside the story pop-up.

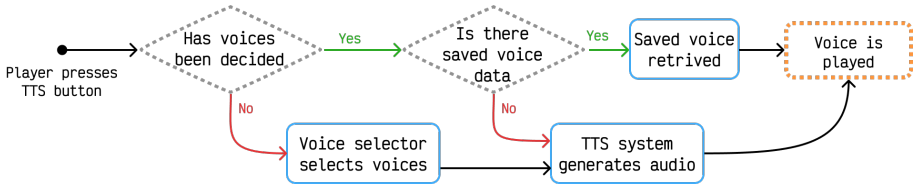



Figure 6.63: Internal logic flow of the TTS system.

splitting point or overflows the buffers completely. Figure 6.64 shows these buffers and the intended splitting point between them. The reason why there are buffers around the splitting point is to avoid a paragraph being split between pages if only a few words fall on either of the pages. This makes TTS easier to understand due to the impossibility of reading only a few disconnected words from a paragraph that has been split.

Lastly, before finishing the feature, an option was added inside the settings menu that allowed players to input the path to the JSON credentials file. Similarly to the API key that gets verified before a player can start the game, the JSON credential file also gets verified so that it is known that the TTS system will work. If it successfully verifies the credentials, the verifying button to the right of the text field will say “Success!”. Furthermore, additional audio options were added that allow for more precise settings of the music volume, TTS volume and a sound effects volume.

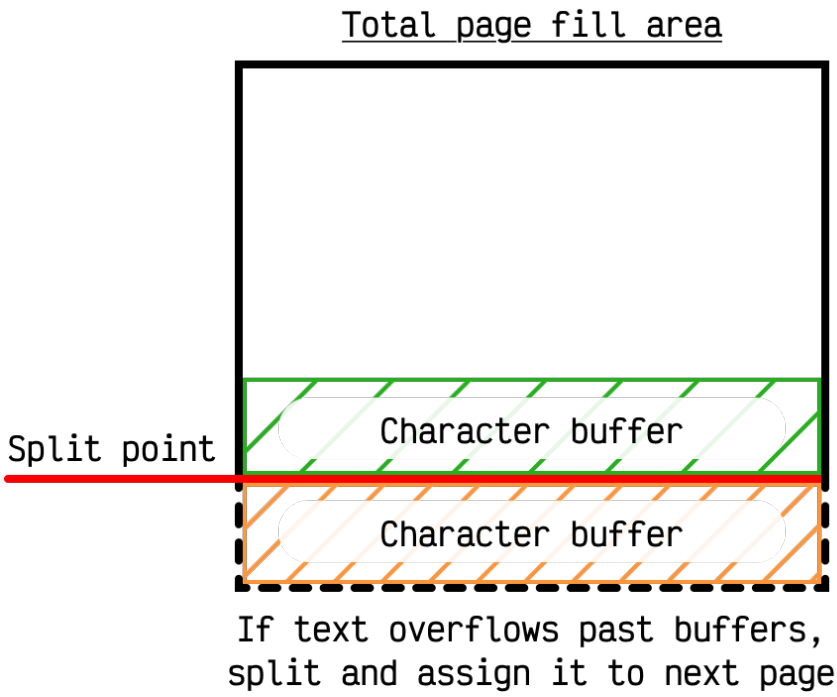


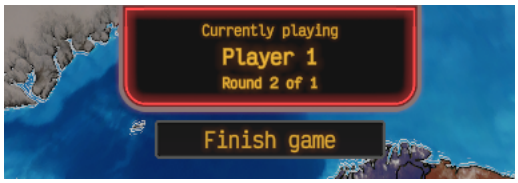

Figure 6.64: Page splitting logic exemplified with character buffers.

Sound effects include the typing sound made by the local instruction writer AI or the victory pop-up typewriter sound.


Figure 6.65: Settings panel with added TTS and sound options.

### 6.10.5 Small GUI changes

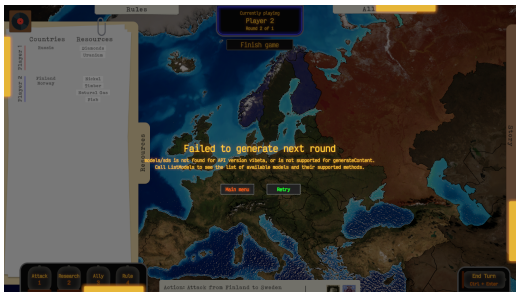
The last developments to the project included a few last GUI changes that polish the game to make it look cohesive. Figure 6.66 shows these updates that were made.



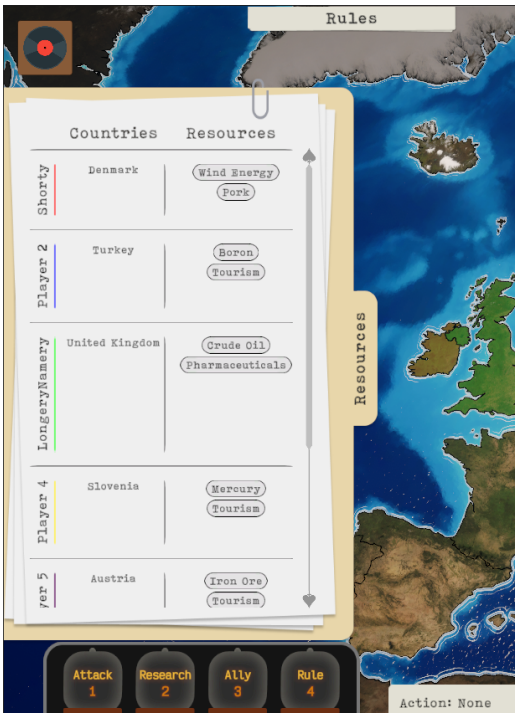

(a) The new current player panel changed to be visually similar to the action menu.


(b) The setting panel that got its name changed to “Options”.


(c) Updated colors and typeface for the loading screen.




(d) Updated color and typeface of the error screen.




(e) The new graphical style of the resource panel made to fit between added GUI elements in *Evolved Domination*.

Figure 6.66: The last GUI changes aimed at making the application cohesive throughout.

# 7

# Results

Results are presented in four different parts. First, the game *Evolved Domination* is presented from a player perspective, followed by two sections of technical details. Finally, a SWOT analysis is presented highlighting strengths, weaknesses, opportunities and threats for the thesis.

## 7.1 Evolved Domination from a player perspective

*Evolved Domination* is a turn-based strategy game that takes place on the map of Europe, where players battle to conquer different countries. Players perform turns separately in each round by selecting actions and writing creative instructions on how they are performed. This will be used to procedurally generate the next story and game state using an LLM.

Below, a walkthrough of a normal round in the game will be presented, highlighting all functionality that the players interact with.

### 7.1.1 Main menu

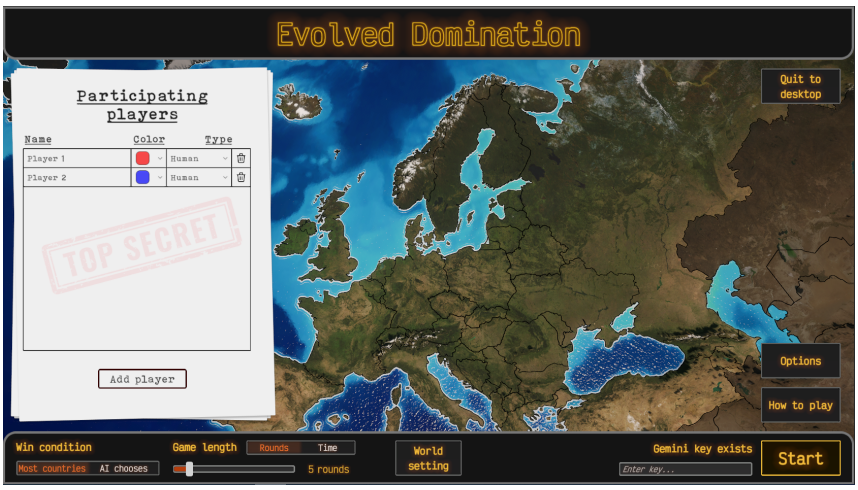



Figure 7.1: The final main menu design in *Evolved Domination*.

When *Evolved Domination* is started, the game boots up to the main menu seen in Figure 7.1. Here, the player can select how many players there should be in the game, what color they are and which type of player it is. Selectable types include *Human*, *Computer* and *AI*. The win condition can be switched between "Most countries" and "AI chooses", which have been explained in sections 6.3.4 and 6.7.9 respectively. Likewise, the toggle option for round-based and time-based game length is also explained in the corresponding sections 6.3.4 and 6.4.6 which switches between the two modes. The slider underneath allows players to choose either the number of rounds or the game duration in minutes before the victory pop-up is displayed. The last setting that can be changed is the world setting developed in Section 6.10.3. It lets users type in a world setting that the game will try to apply when it generates in-game information.

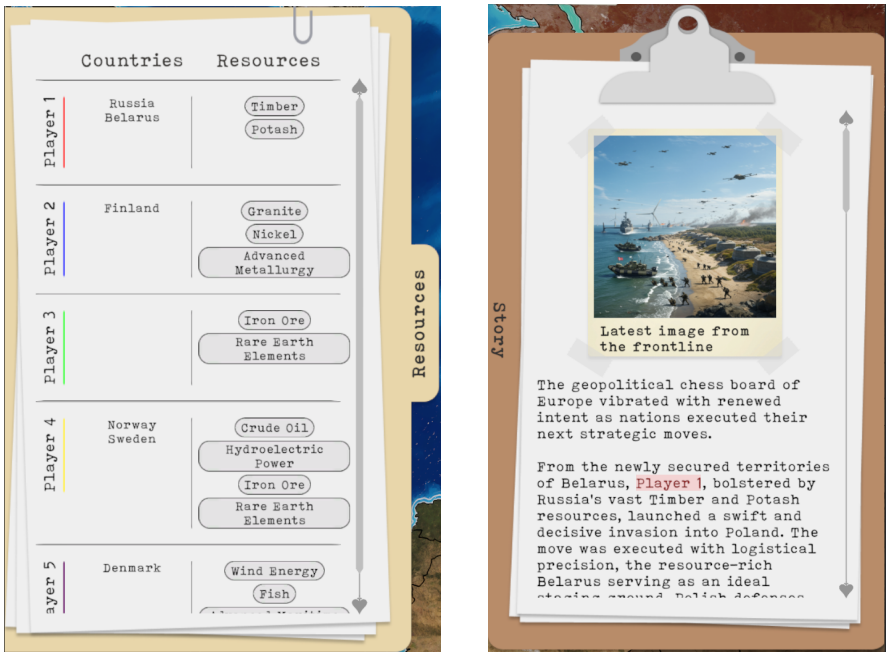
After these settings have been decided upon, the player needs to verify their Gemini API key before starting the game. This is done through the API validator in the bottom right corner of Figure 7.1. If the key is invalid, the text above the input field will display a "Try again" message, and if it passes, the start button becomes yellow instead of an initial red color. If the game is not started, there are also buttons that lead to the settings menu written about in Section 7.1.9 and the "How to play" panel described in Section 6.9.4. These are additional panels where the player can change settings or learn how *Evolved Domination* functions.

### 7.1.2 Game view

When in the game view seen in Figure 7.2, players can move the camera around in the world by using the keys: *W*, *A*, *S* and *D*. Additionally, camera movement can be performed while holding down the right mouse button and moving the mouse around. Camera rotation can be done by pressing either *Q* or *E*, and rotates dependently left or right. Players can also use the scroll wheel on the mouse to zoom in and get close to the map.


Figure 7.2: Screenshot showing the game view.

Inside the game view there are six UI panels serving different purposes. The resource panel depicted in Figure 7.3a displays all players' occupied countries and current resources. The story panel seen in Figure 7.3b shows the latest story generated.




(a) The resource panel displaying player data such as occupied countries and resources.

(b) Story panel shown on the right side in the game.

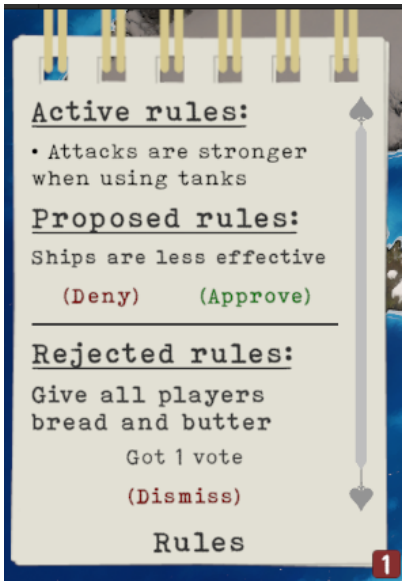
Figure 7.3: Showing the resource and story panel in the game view.

The third panel is the instruction letter seen in Figure 7.4, located at the bottom of the screen. When players have selected an action, this automatically appears, allowing players to write their instruction that fits their action in the input field. If needed, players can also press "Help me write" to have the local LLM, explained in Section 7.2.9, write an instruction based on the selected action, resources and existing text in the input field. Additionally, the two actions *Attack* and *Research* allow players to attach resources to the action by dragging them from the resource panel to the bottom part of the instruction letter.

Panels four and five are used for the rule or the ally system respectively. The rule panel seen in Figure 7.5a lists all currently active rules, proposed rules this round and rejected rule if there are any. When voting on proposed rules, players either select "Deny" or "Approve", leading to a drawn circle shown around the answer. This allows them to change the answer afterwards if desired. Rejected rules can also be dismissed by pressing the button "Dismiss" and a red notification icon is shown if there is any new information in the panel.

The ally panel, shown in Figure 7.5b, works similarly, but instead of showing rules, it presents ally information. It displays current allies with chat buttons for each ally, explained in Section 7.1.4. Furthermore, it also lists current ally requests that need answering and rejected requests that can be dismissed.


Figure 7.4: The instruction letter showing current action, input field, attached resources and a "Help me write" button.




(a) Rule panel showing active, proposed and rejected rules.

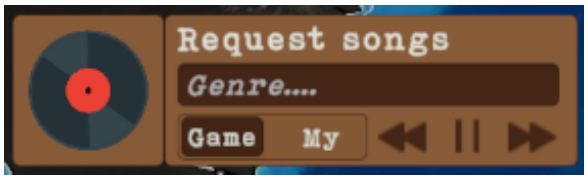

(b) Ally panel showing current allies, incoming ally requests and rejected requests.

Figure 7.5: Showing the rule panel, ally panel and the chat window.

The last panel is the music panel located at the top left corner of the game view. The panel is seen in Figure 7.6 and allows players to request music in-game by writing in the input field. The underlying music system is further explained in Section 7.1.6.

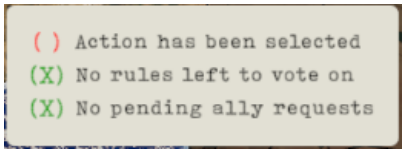
Players can end their turn by either clicking on the button in the bottom right corner seen in Figure 7.7b or pressing the keyboard shortcut *End* or *Ctrl + Enter*. If players have not selected an action, voted on all rules or answered all pending ally requests, a pop-up appears. The pop-up is shown in Figure 7.7a and shows the player the necessary steps needed before the turn can end. This pop-up window appears beside the end turn button on the bottom right corner of the screen.

(a) The music panel showing an LP record.




(b) Expanded view showing an input field, toggle button and playback control.

Figure 7.6: The normal and expanded version of the music panel.




(a) Turn validation pop-up when mandatory tasks are not completed.


(b) The end turn button for proceeding with the game.

Figure 7.7: The turn validation and end turn UI elements.

### 7.1.3 Selecting actions

Players can select four different actions by using the panel in the bottom left of the game view, seen in Figure 7.8a and created in Section 6.7.5, or using the number keys: *1*, *2*, *3* and *4*. Before submitting a turn, players are forced to choose one action. The first action is *Attack* and can be performed by selecting it. A pop-up is presented, indicating that players first should select a country to attack from and then a country to attack towards. The order of these can be switched if the player presses the *tab* key, and depending on which one, a cursor representing a house or a sword is shown respectively to help players know which they currently select. An alternative way to attack is also done by clicking on a country to attack from and dragging the mouse to a country to attack. A red arrow, as seen in Section 6.7.5, is displayed to help show the attacking vector.

To *Research* or create a *Rule*, players can select one of these actions and write an instruction for it. The last action, *Ally*, can be used by selecting the action and picking a player on the list that appears above, seen in Figure 7.8b. A message can then be typed, which is sent to the other player. The other player then needs to approve or deny the request.

### 7.1.4 Message allies

As mentioned, players can become allies with others by first performing the *Ally* action or accepting an ally request. Approved ally requests are seen at the top of the ally panel, depicted in Figure 7.9, and declined requests at the bottom. In the middle, the same figure also highlights how an ally request is perceived from the


(a) Action button panel located in the bottom left corner.

(b) Pop-up when choosing ally after selecting ally action.

Figure 7.8: Action buttons and pop-up when choosing allies.

receiving side, displaying a message with an *approve* and a *deny* button.

Furthermore, players can message other players by opening the ally panel and pressing the chat button beside an existing ally. A new panel appears to the right, seen in Figure 7.9, which lists ally messages sent in the conversation and a text field to write a new message. Players send messages by typing in the text field and pressing the *Enter* key. The message will then be received by the recipient next turn. A notification badge also appears if an ally request or a rejected ally request is present.


Figure 7.9: The ally panel to the left, showing current allies, ally requests and rejected requests with the chat window to the right displaying current messages and an input field.

### 7.1.5 Rule voting

Players can create rules that affect how the game works. First, the player suggests a rule by selecting the *Rule* action and writing the rule as an instruction. After all players have made their turn, all suggested rules this round are validated to see if they break any existing rules in the game. The validation results are presented in the next story, and if successful, the approved rules proceed to be voted on. All players are required to vote by the end of their turn. The voting results are presented in the story next turn. If a rule receives the majority of votes, it will be applied to the game and otherwise, be rejected. The status of a rule can be followed in the rule panel displayed in Figure 7.5a.

### 7.1.6 Changing music

Music in the game is controlled by the music selector discussed in Section 7.2.6. For each round, it plays two songs that fit the story played when the story pop-up is open and then around five songs chosen by the game when players are selecting actions. Players can request music by opening the music panel seen in Figure 7.6 and writing a genre, style, feeling or time era in the input field. The game then selects music based on this to be played. To switch between the player-chosen music and the original game-selected music, a toggle button can be pressed seen in Figure 7.6b. The figure also shows playback controls that can be accessed, such as going to the previous music, next music or pause.

### 7.1.7 New round

When all players have performed their turn, the game state containing all player actions, instructions, currently occupied countries and resources is sent to the Game Master, discussed further in Section 7.2.1. After receiving a response, the new story is presented in the game as a pop-up briefcase shown in Figure 7.10. Player names are also highlighted in their respective color.

The story panel also allows players to activate a text-to-speech feature indicated by a speaker icon. When activated, the current page is read by different voices depending on the story content. An hourglass icon is displayed to show that the game is waiting for a response from the TTS service.

Lastly, a generated image can be seen to the right on the story panel if the image generation setting is turned on. The image is generated based on the content in the story and covers one important story element the Game Master decides upon.

### 7.1.8 Victory pop-up

The game is finished when the round has reached its limit or the time is up. If this criterion is met, the game will present a victory pop-up after all players have made their last turn. Depending on which win condition was chosen on the main menu, one of the two screens in Figure 7.11 is shown.


Figure 7.10: The story panel showing the story and an accompanying image.

If the condition “Most countries” was chosen, the pop-up in Figure 7.11a is displayed listing the winning player and a leaderboard containing the number of current countries occupied by each player.

If the other condition “AI chooses” was chosen, a similar pop-up is shown as displayed in Figure 7.11b. The content includes the winning player and a motivation for why the player won. It also lists all other players and their score, player type and score motivation the AI has generated.


(a) Victory pop-up for most countries win condition.


(b) Victory pop-up for when AI chooses.

Figure 7.11: The victory pop-up showing the two different win conditions.

The victory pop-up is animated upwards when first shown. First, the typewriter moves from outside the screen upwards. After that, the paper slowly moves upwards and characters are printed out one by one on the paper. A sound is also played for

each character typed, along with a bell sound after completion. This is done to loosely mimic the function of a real typewriter. A victory song is also played when the pop-up opens.

### 7.1.9 Changing options

The option menu can be opened in the main menu or via the pause menu in the game. It is divided into four parts: *Sound*, *Text-to-speech*, *Image generation* and *AI models* as seen in Figure 7.12.

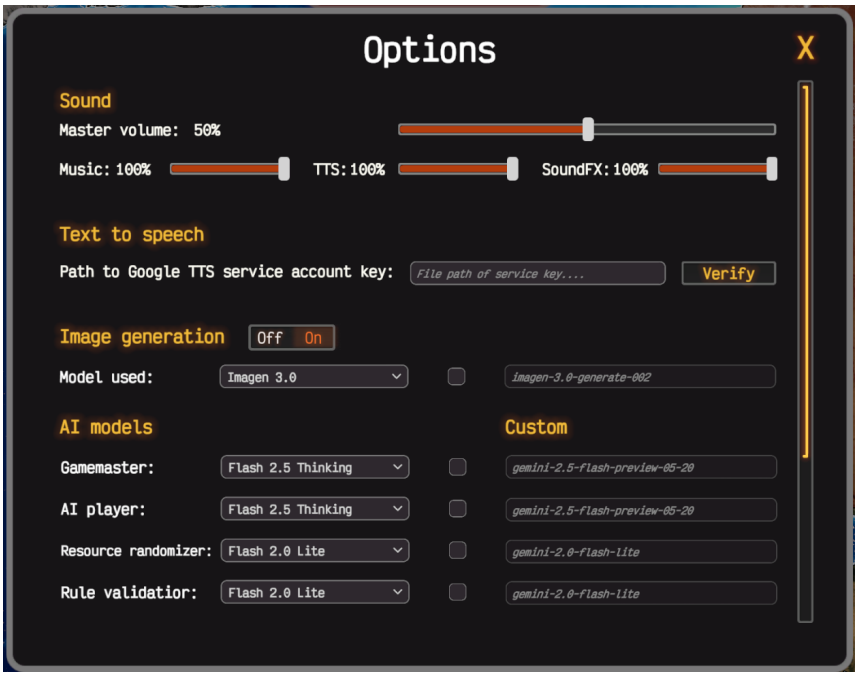



Figure 7.12: The options menu.

Sound levels can be changed by adjusting four different sliders. The master slider adjusts the overall sound level. The other three sliders adjust individual volumes of sound sources such as music, TTS and sound effects.

Text-to-speech can be enabled by entering a valid path to a file containing credentials for accessing the Google voice service [142]. If successful, the button says “Success”, otherwise, it says “Try again”.

Image generation can be turned on or off with the toggle button. If the setting is on, players can select which model to use from a predefined dropdown list or choose a custom model by inputting a model identifier in the input field. These must be taken from Google’s official list of supported models [145].

The last section lists all the other cloud-based AI instances in the game and which underlying model they use. These can be changed similarly to the model selection for the image generation above using either the predefined models in the dropdown list or choosing a custom model. Image generation models cannot be selected as the other cloud-based AI instances use text-based models.

## 7.2 AI instances in Evolved Domination

*Evolved Domination* consists of 12 different AI instances that cooperate seamlessly in the background during gameplay. This section first explains how all these instances work together to form the cohesive experience and then explains the purpose, input and output for each instance.

The AI architecture is depicted in Figure 7.13. First, players launch the game and are presented with the main menu. The main menu allows players to input a mandatory API key used to play the game, which is then verified by the *API verifier*. After verification, the *Country resource randomizer* generates random resources for each country before starting the game.

Directly when the player presses the start button, a request to the *Music selector* instance is made to select which music to play during gameplay. One or multiple *LLMGA* requests are also sent if there are any AI players present in the game.

Then the game continues to the phase where players are performing actions. Here, players can choose to get help when writing an instruction, calling the *Local instruction generation* instance to retrieve an instruction. Additionally, if any rules are proposed or rules voted on, the respective instances *Rule validator* or *Rule applicator* are used before sending a request with the entire game state to the *Game Master*.

The *Game Master* response is then parsed, and the game proceeds to a new round, calling the previously mentioned *LLMGA* and *Music selector* instances. Meanwhile, a story pop-up is also displayed to the players. If players have enabled image generation, the *Image generation* instance request will automatically be dispatched. Players can also use the TTS button to generate a natural-sounding voice for the story. This first uses the *Voice selector* instance to select a suitable voice and then the *Text-to-speech* instance to generate the audio for the voice.

If the game has not finished, it transitions to the "When pop-up is closed" box, performing the loop until the game has finished. If the game has ended, it checks if the win condition is "AI chooses". Then it dispatches a request to the *Game winner* instance to determine the winning player and opens up the victory pop-up. Otherwise, the default condition "Most countries" is used and the victory pop-up is shown directly.

Figure 7.13 contains a legend describing which instances use the input world setting, story or player rules. The world setting is used as input in the instances *Country resource randomizer*, *LLMGA*, *Music selector*, *Local generation*, *Game Master* and *Image generation*. The current round story or complete story history is used in the *LLMGA*, *Music selector*, *Game Master*, *Image generation*, *Voice selector* and *Game winner*. Lastly, player rules are injected into the *LLMGA*, *Rule validator*, *Rule applicatior* and *Game Master*.

Redundancy was also added to make the game more resilient to problems with external services. If instances fail with their first request, a second request is made using a fallback model. This allows the system to still work if a specific model is

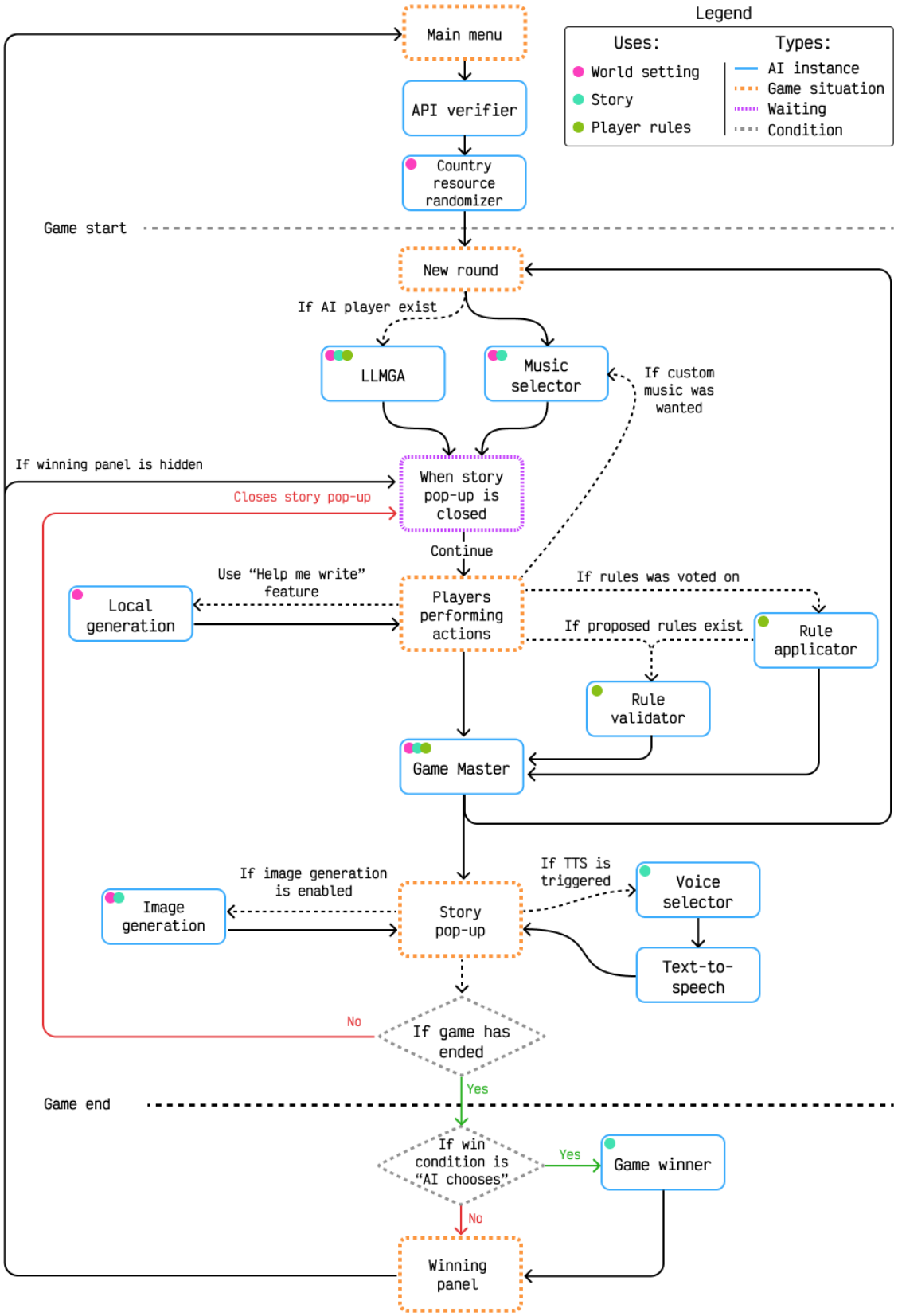



Figure 7.13: An overview of all AI instances in *Evolved Domination*.

unavailable or fails. If this request also fails, default values are used for the instances that support it. The AI instances that support this are the Game Master, LLMGA, Country resource randomizer, Rule validator and Game winner.

### 7.2.1 Game master

The Game Master is the main AI instance inside *Evolved Domination*, used to generate the story and move the game forward. It uses *Gemini 2.5 Flash* to consider all player actions and combines it with the current game state to generate the next round for all players according to the system instruction found in Appendix B.1. While the game is running, there can also be additions to the system instruction in the form of the "World setting", discussed in Section 6.10.3, and player-created rules, explained in Section 7.1.5. These are injected into the system instruction each time a command is sent to the AI instance and can therefore be seen as a part of it.

In addition to the information injected into the system instruction, a JSON object is sent to the model each round that contains the data that should be processed. The difference between the data injected in the system instruction and the data inside the JSON object is that the injected data can be considered as static information while the JSON data is round-dependent. The JSON structure can be seen below:

```
{
    "Players": [
        {
            "Name": "",
            "Action": "",
            "Instruction": "",
            "UsedResources": [""],
            "Resources": [""],
            "CurrentlyOccupying": [""],
            "AlliedPlayers": [""]
        }
    ],
    "Events": [""]
}
```

The JSON structure consists of two parts, the "Players" and "Events" sections. The player section contains an array filled with objects representing player information and their corresponding action during the round. The event section consists of a list of strings that are considered game events and should be mentioned in the generated story later. Moreover, as rounds are played inside the game, each input and output to the AI model is also stored as game history. This historical data is sent to the AI model each turn in the form of a chat history, which helps the model generate responses that can take earlier game history into account.

When the model has processed and generated a new round, it responds with a JSON object formatted as follows:

```
{
   "story": "",
   "StoryImageGenerationPrompt": "",
    "Players": [
```

```
        {
            "name": "",
            "resources": [""],
            "actionSucceeded": true/false,
            "currentlyOccupying": [""]
        }
    ]
}
```

The response contains the generated story for the round, an image generation prompt for the image generation functionality and an array containing the updated game state for all players. This information is then utilized to update the internal game state and display the information to the users.

If the Game Master AI instance fails to generate a response, there is a feedback system that displays the error information on the screen and a retry option that allows players to try and send the request again. More information about this system can be found in Section 6.9.1.

### 7.2.2 LLMGA

The Large Language Model Game Agent (LLMGA) is the most complex AI instance in *Evolved Domination*. Its purpose is to be an AI player that simulates a human player capable of performing any action a real person can do in-game. The instance uses *Gemini 2.5 Flash* to generate its response and follows the system instruction found in Appendix B.3.

Similarly to the Game Master, there is static data that is created and injected at runtime into the system instructions for the LLMGA. The injected data sets the information for how many rounds the game will play before finishing based on settings from the main menu, every resource each country has decided in the "Country resource randomizer" discussed in Section 7.2.3, the "World setting" discussed in Section 6.10.3 and lastly all player-created rules created during the game. It also utilizes game history so that the LLMGA can know how the game has evolved throughout the play session and build upon its own actions to form more complicated plans.

```
{
    "MyPlayer": {
        "Name": "",
        "Resources": [""],
        "CurrentlyOccupying": [""],
        "AlliedPlayers": [""]
    },
    "OtherPlayers": [
        {
            "Name": "",
            "Resources": [""],
            "CurrentlyOccupying": [""]
        }
        // ...
    ],
    "TurnNumber": 1,
```

```
    "GeneratedStory": "",
    "RulesToVoteOn": [""],
    "PendingAllyRequests": [
      {
        "RequestFrom": "",
        "Message": ""
      }
      // ...
    ],
    "AlliedPlayerChats": [
        {
            "ChatWith": "",
            "Messages": [""]
        }
        // ...
    ]
}
```

The data is sent to the AI using the JSON structure found above this paragraph. It contains a "MyPlayer" object representing the AI player itself, an array with all other players and their information, which turn number it currently is, the generated story for the latest round, rules and ally requests to answer and lastly, all player chats that the AI has been involved in. The LLMGA has been given identical information that is available for normal players if combined with the static information injected into the system instruction.

```
{
  "Action": "",
  "Instruction": "",
  "UsedResources": [""],
  "AllianceRequestResponses": [
    {
      "PlayerName": "",
      "Response": true
    }
    // ...
  ],
  "RuleVotes": [
    {
      "RuleText": "",
      "Vote": true
    }
    // ...
  ],
  "MessagesToSend": [
    {
      "RecipientName": "",
      "MessageText": ""
    }
    // ...
  ]
}
```

Likewise, the output JSON structure seen above is designed so that the LLGMA can make decisions equivalent to a human player and create a response that interacts

with every in-game system. It returns an action, instruction and which resources it wants to use in combination with answering every ally request and rule vote. Lastly, it can also send chat messages to other players if the AI has a chat with another player. If the LLMGA fails to generate a response, a standard instruction will be sent to the Game Master stating that the AI instance forgot how to make its move and therefore that no action was taken.

### 7.2.3 Country resource randomizer

Resources for all countries are generated at the start of the game after a valid API key has been provided. The instance uses the less powerful but faster model *Gemini 2.0 Flash Lite* with a temperature value of 1.7 to increase randomness. The system instruction is found in Appendix B.4 and instructs the model to generate two diverse resources per country. If a world setting has been provided when the game starts, it injects that into the system instruction and regenerates the resources for all countries based on the new world setting. The world setting is further explained in Section 6.10.3. The input and output JSON schema includes a list of country data containing name, resources and who is occupying the country. This structure can be seen below.

```
[
    {
        "CountryName": "",
        "CountryResources": [],
        "OccupiedBy": ""
    },
    // ...
    {
        "CountryName": "",
        "CountryResources": [],
        "OccupiedBy": ""
    }
]
```

If the normal and fallback requests fail, the resources will default to randomly selecting two resources from this list: Wood, Stone, Iron, Oil, Gold, Coal, Fish and Wheat.

### 7.2.4 Rule validator

The rule validator approves or declines rules suggested by players using *Gemini 2.0 Flash Lite*, taking all the existing game rules into account. It does this by inserting these rules into the system instruction when a request is initiated. The system instruction is listed in Appendix B.5 and includes a description of how it should validate rules alongside a description of the input and output structure. The input provided in the request is an array of player-created rules that need validation, as seen below.

```
{
    [
        "Rule 1",
        // ...
        "Rule 10"
    ]
}
```

The output JSON structure contains a boolean value if the rule is valid or not and a reason it was approved or declined. The structure can be seen below.

```
{
    "isValidRules": [
        true,
        // ...
        false
    ],
    reason: [
        "Reason",
        // ...
        "Reason"
    ]
}
```

On failure with the regular and fallback model, the rules are automatically rejected with the reason "Failed to validate rule".

### 7.2.5 Rule applicator

When a rule has successfully passed player voting, the rule applicator makes a request to the model *Gemini 2.0 Flash Lite* to check if the rule should modify any game parameters. If so, it uses function calls to output one or more functions which are defined in Appendix C.1. The instance then uses a system instruction found in Appendix B.6 and receives an array of rules passed this round, as seen below.

```
[
    "Rule 1",
    // ...
    "Rule N"
]
```

It then outputs an array containing a combination of the five functions: *setMaxNrOfAttachedResources*, *changeCurrentNrOfRounds*, *changeWinningRound*, *addResourcesToPlayers* and *doNothing*. This allows it to modify internal game parameters, such as setting the winning round to 100 or giving resources to players. The structure for the array of functions can be seen below.

```
{
    "Functions": [
        {
            "Name": ""
            "Arguemnts": []
        }
    ]
```

```
}
```

On failure, it returns an empty list to indicate that no function calling should be done.

### 7.2.6 Music selector

This instance utilizes *Gemini 2.0 Flash* with function calls to determine which music should be played in the game. The system instruction found in Appendix B.7 instructs the model to take an input string similar to the one seen below and output an array of function calls that fit the input. The world setting is also injected into the system instruction to allow the music to fit the current world setting.

```
"Give me 5 calm songs"
```

Each function call it can choose from comes from a music database containing 61 unique songs with five different types of attributes: style, moods, tempo, instruments and dynamics. Style indicates which type of music it is, for example, cinematic or classical. Moods describe which moods exist such as majestic, calm or energetic. Tempo is how fast or slow the music is. Instruments include all instruments in the music. Lastly, dynamics describes the sound level, whether it is lively, quiet or gentle. On average, there are 25 attributes for each song. An example of a single function can be seen below.

```
{
    "Functions": [
        {
            "Name": "FileName.mp3",
            "Description": "Style: cinematic, film score, epic
             adventure. Moods: heroic, dramatic, building. Tempo:
             fast, driving, moderate. Instruments: brass,
             percussion, strings. Dynamics: forte, powerful, epic."
            "Parameters": {
                fitsStory: true
            }
        }
    ]
}
```

The output provided by the instance is an array of functions that indicate what songs to play. It also includes whether the generated song is connected to the story and therefore should be played first, before the other songs. The structure of the output can be seen below.

```
{
    "Functions": [
        {
            "Name": "",
            "Arguments": {
                "fitsStory": false/true
            }
        }
    ]
```

```
}
```

### 7.2.7 Game winner

The AI instance that decides who wins the game uses *Gemini 2.0 Flash* and takes the entire game history as input. It uses this to select a winner based on which player it thinks has performed the best during the game. Examples of factors it can look at are which player has the most countries, which players used a smart game strategy, has set themselves up for success or has been successful with alliances. The system instruction it uses as a base for this is found in Appendix B.8. The response contains the winning player and a motivation for why it is the winning player. It also responds with an array of data for each player that describes their playstyle, a score between 0 to 10 and a motivation for this score. The JSON response schema can be seen below.

```
{
    "players": [
        {
            "name": "",
            "playerType": "",
            "score": 0,
            "scoreMotivation": ""
        }
    ],
    "winningPlayer": "",
    "reason": ""
}
```

If the API is unresponsive, it defaults to selecting the winner based on which player has the most countries.

### 7.2.8 API verifier

The API verifier instance is the simplest AI instance in *Evolved Domination*. Its purpose is to verify that the Gemini API key inputted by the player is valid and will function throughout the game. It functions by first sending the message “hi” to *Gemini 2.0 Flash Lite* and waits for a response. If a response is received, then the API key is considered valid. If it fails, however, it tries sending the same message to *Gemini 2.0 Flash* to see if that model responds before considering the API key as invalid. It does this extra call to lessen the chance of it falsely denying an API key due to the primary model being overloaded.

### 7.2.9 Local instruction generation

This AI instance helps the player write an instruction connected to the selected action by utilizing a local LLM running on the device. The instance uses the *Gemma 3 1B* model in combination with a system instruction described in more detail in Section 6.6.1. The system instruction briefly describes what the game is about and

what actions can be performed. It also describes how the model should respond and that it should answer from the player's perspective.

```
"Player {chosenAction}.
Starting instruction: {startingInstruction}.
Player resources: {playerResources}"
```

The input given to the model is the chosen action, the start of an instruction if the player has started writing an instruction and the players' resources. When processing, the model uses a streaming approach, meaning that the generated instruction is returned word by word, updating the UI in the process, instead of receiving the response all at once. Since this is a local model, no fallback system exists as it can be guaranteed that the model is accessible and will generate an output.

### 7.2.10 Image generation

The image generation instance generates images connected to the generated story each round. It does this by receiving a prompt from the Game Master and calling Google's *Imagen 3* model. As this model does not support a system instruction, it directly uses the prompt from the game master without any modifications.

The model responds with a base 64 encoded image, which is converted to a byte array and then a texture. Lastly, it converts the texture to a sprite, which can be used in Unity. On failure, the model returns nothing since a backup model for images does not exist in *Evolved Domination*.

### 7.2.11 Voice selector

This AI instance uses *Gemini 2.0 Flash* to generate function calls that decide which voice to use for each paragraph inside the generated story. Its complete system instruction can be found in Appendix B.9 and accomplishes this by first receiving an array of strings in JSON format corresponding to an array of every paragraph inside the generated in-game story. The JSON schema below shows how the input is formatted in more detail.

```
{
    "Story": [
        "Paragraph 1",
        "Paragraph 2",
        // ...
        "Paragraph n"
        ]
}
```

At the same time, since the instance uses function calling, it is also provided with a list of functions, where each element represents a selectable voice. This is done similarly to the music selector AI instance in that a voice database has been created that contains information about the 38 possible voices that can currently be chosen. This database is then automatically processed to generate the function list with a real example of a single element being:

```
{
    "Name": "en-GB-Chirp3-HD-Aoede",
    "Description": "Voice name: en-GB-Chirp3-HD-Aoede. Gender:
     FEMALE. Voice attributes: Brighter, Clean. Voice dialect: en-
     GB"
}
```

The model is then instructed to pick a voice function for each corresponding paragraph it received in its story input and return it to the game. Its output is therefore a list of function names corresponding to the desired vocal style for a certain section. The name is then utilized to gather the information needed for the text-to-speech model.

```
{
    "Functions": [
        {
            "Name": "Voice name for paragraph 1"
        }
        {
            "Name": "Voice name for paragraph 2"
        }
        // ...
        {
            "Name": "Voice name for paragraph n"
        }
    ]
}
```

If the AI instance fails to generate a response, a pre-defined fallback voice option is used.

### 7.2.12 Text-to-speech

The text-to-speech module inside *Evolved Domination* functions differently from all other AI instances in that it uses its own API called *Google Cloud Text-To-Speech* [142]. This, therefore, requires the player to assign a file path to another credential file containing the required information to access the API. When creating an API request, Google's *Google.Cloud.TextToSpeech.V1* [143] C# package is then used to more easily build the input needed for the call. It contains the text to be spoken, the name of the voice that should be used and the language code representing the language the voice should be in. After audio generation has been finished, the output is received as a 16-bit linear audio stream, which gets converted into an audio file that Unity can play.

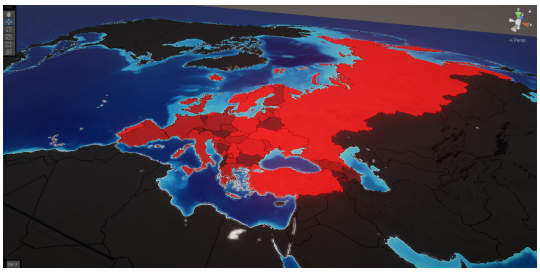
If the API fails to generate a voice, the game will continue normally without playing the sound. Further information on how the text-to-speech system works can be found in Section 6.10.4 where it describes how this module works together with the voice selector AI instance described in Section 7.2.11. It also gives a more thorough explanation of how each story is broken down into sections so that the text-to-speech module can generate audio with different voices for each story section.

## 7.3 Additional systems in Evolved Domination

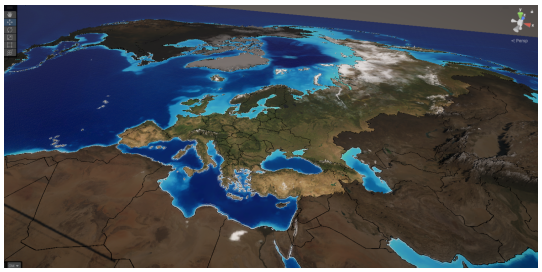
This section mentions a few additional internal systems important to the games inner workings for the experience to be seamless while playing. They will not go into specific implementation details here but rather explain their overall purpose.

### 7.3.1 Terrain system

As mentioned in Section 6.7.1, the basis of the terrain system used inside *Evolved Domination* originates from Sebastian Lagues *Geographical Adventures* game [134] where Lague uses data from NASA to generate a virtual globe. This terrain meshing system has been modified to generate flat terrain on a plane instead of a globe and rotated the flat map to make Europe centered on it. Since mesh generation is expensive, the mesh data is then saved to a file and loaded when the game is started. The terrain then uses a custom shader, partially inspired by Lague, that utilizes satellite maps from NASA to generate a base texture. Together with this base texture, it also uses the country index map shown in Figure 7.14a to color the countries on the terrain based on player occupation and if the country is outside Europe. The final world terrain is seen in Figure 7.14b.



(a) Country index map used to color countries.



(b) The final world terrain.

Figure 7.14: The underlying country index map and the final terrain inside *Evolved Domination*.

Similarly to the meshing system, the ocean shader also originates from Lagues work with modifications to allow it to run in *Evolved Domination's* Unity environment and for it to be situated on a flat plane. Further details about the ocean shader implementation and how terrain coloring is done are found in Section 6.7.1.

### 7.3.2 Virtual player system

The virtual player system is described further in sections 6.3.5 and 6.6.7, but is the underlying system that allows human players to play against both *Computer Players* and *AI Players*. During a round, the internal game system will set up so that every player gets to do their turn in the predefined order that was decided upon at the beginning. When it is a human player, the GUI initializes to that player and waits for the player to input their actions before finishing their turn and moving on to the next. However, when a virtual player comes, it calls a special function that all

virtual players have that decides their move instead of setting up the GUI. This is handled differently for the *Computer Player* and *AI Player*, but the result is that the virtual player makes a move that can be read by the game.

An additional *Deterministic Player* utilizing this system was also created for testing purposes, but is not included in the final game. This can be read about in Section 6.3.5.

### 7.3.3 Message system

The message system inside *Evolved Domination* was first constructed to allow messages to be sent between allied players during gameplay. It is explained in Section 6.4.3, but quickly grew to include sending ally requests and rule votes as explained in sections 6.4.4 and 6.6.4 respectively. The system allows players to send generic messages of certain types to each other via a postal manager that routes messages between players when a round is finished. Having this as a central system ensures that message handling is consolidated into a cohesive class where ally requests and rule voting can be handled. It also simplifies the logic required for each player since they only require a mailbox where messages can be sent.

## 7.4 SWOT analysis

This section will present an analysis over the Strengths, Weaknesses, Opportunities and Threats that have been found throughout developing *Evolved Domination* with generative AI. These findings also aim to answer the research question found in Section 1.1.

### 7.4.1 Strengths

#### Dynamic system instructions allow customizability

*Evolved Domination* heavily relies on system instructions that decide how AI instances should behave. Instead of only using static versions defined once, the game inserts information at runtime into them. For instance, the Game Master, discussed in Section 7.2.1, injects all player-created rules and the world setting at runtime, allowing it to incorporate this when generating new game states. This dynamic system instruction can therefore allow model behavior changes during runtime by adding or removing information in the instruction.

#### PCG enhances replayability

As discussed in Section 2.1, Procedural Content Generation (PCG) allows for generating different types of content. *Evolved Domination* uses PCG, for example, when generating stories as discussed in Section 7.2.1, to allow unique narratives to be generated each round. These unique narratives give the game higher replayability as players experience different stories even after doing the same actions. Additionally, the "World setting" allows players to customize the content generation further by

specifying an environment the game should use, discussed further in Section 6.10.3, leading to a larger number of possible playthroughs.

From another point of view, even if the content is technically different every playthrough, patterns may still emerge that possibly cause players to feel that experiences are similar between playthroughs. This may therefore reduce replayability, but should still be greater than if the game had pre-generated content.

### Reducing manual work for content creation

Game development generally includes content usually generated manually by designers, 3D artists or developers. Part of this work can be offloaded to different generative AI models, reducing the need for manual content creation. *Evolved Domination* exemplifies this by offloading work to different AI instances when needed, allowing for the dynamic creation of unique stories, images or voices at runtime instead of requiring work to be done manually by developers. Since the game uses undetermined player actions to generate new game states, it would be time-consuming to try and create stories manually that fit. This is where the LLM in the game shows its strength, by generating unique narratives each round based on unpredictable player actions.

### Reasoning models improve quality and adherence to game rules

A type of model called reasoning models, as described in Section 3.2, uses internal reasoning to improve output quality. The early investigation of the AI solution used in *Evolved Domination*, described in Section 6.4.1, found that a switch from using non-reasoning models to reasoning models improved response quality and adherence to rules. The main reason for this improvement was that the model would follow the system instruction more closely, such as keeping the new game state and story in sync, preventing discrepancies between the story and the game state. An addition of a self-check also ensured that the model went through its response one last time to double-check that all rules were followed. This is extra beneficial for reasoning models since the reasoning process allows the model to iterate on the result compared to non-reasoning models.

### Different models for different needs

A diverse range of AI models exists with different capabilities that can be tailored to different use cases. Google's Gemini models have different tiers, ranging from fast, cost-efficient models to slower models striving for higher response accuracy [145].

As *Evolved Domination* consists of multiple AI instances, described in detail in Section 7.2, the game uses the different Gemini models for different use cases. For instance, the Game Master that generates the new game states uses the slower but more accurate *Gemini 2.5 Flash* model with reasoning enabled. In comparison, the faster *Gemini 2.0 Flash Lite* is used when randomizing country resources since it does not need to be as accurate, prioritizing speed over quality to allow the players to start the game as fast as possible.

### Long context improves LLMs consistency

The Gemini models support long contexts, as discussed in Section 2.3, making them perform well for large inputs of data. In *Evolved Domination*, this enabled the storage of all input and output data given to certain AI systems in a history database. When sending a new request, the entirety of this history data would be included, giving the AI as much game context as possible. This resulted in stories that evolve and build upon the history provided, improving the consistency of the game.

In sprint three, described in Section 6.4, it was discovered that the history was not attached correctly when sending requests to the game master, leading to incoherent stories and unexpected behavior. This therefore shows that context is important and as much of it as possible should be provided to the AI when sending requests.

### Structured output allows for reliable integration of LLM responses

As explained in Section 2.3, structured output is used to obtain a deterministic response structure when receiving requests from the AI. All suitable AI instances inside *Evolved Domination*, defined in Section 7.2, use this to ensure consistent parsing of the data to eliminate the risk of partial or malformed data. Examples of these instances are the Game Master, country resource randomizer and the rule validator.

### Using function calls can trigger different game functionalities

By using the *function call* capability in LLMs, it is possible to call different application code functionalities with a set of input parameters [25]. These *function calls* are structured similarly to *structured outputs*, mentioned in Section 2.3, in that they are specified and returned using a JSON schema. More specifically, *function calls* are defined by specifying all available functions, outlining both function names and input parameters. Then the model output consists of one or more function names in combination with the input parameters decided.

*Evolved Domination* uses this to call corresponding methods in the game. One example is when rules are applied using the rule applicator AI instance, described in Section 7.2.5. The instance decides which function call should be used to change functionality inside the game, such as adding player resources or changing the winning round. Another example is when it is used as a decision maker to decide which music to play. As described in Section 7.2.6, a list of all songs and their description is sent to the AI model together with a player prompt. Then the model decides which music fits the prompt and returns a list of songs to play. The ability to call specific functions and use function calls as a decision maker can therefore be considered beneficial and allows the AI to interact with aspects inside the game.

### Parallelization of AI requests

Parallelization of AI requests allows model generation to be run simultaneously when performing multiple model requests. *Evolved Domination* uses this throughout its AI instances to more efficiently handle different situations. For instance, as mentioned

in Section 6.6.7, requests from LLMGA players are sent directly as a new round begins while human players make their turn. This allows the requests to hopefully be completed before the round ends, reducing total waiting time.

Another implementation is with the Image generation, TTS generation, Local instruction generation and Music selection AI instances discussed in Section 7.2. These are dispatched in the background and allow the game to continue as normal while waiting for a response from the different instances. This enables players to do other things in the game while waiting for a response, preventing players from getting stuck in several loading screens.

### Possible use of LLM to find bugs in runtime

During development, the Game Master inside *Evolved Domination* responded with a story highlighting the errors it found in the input it received in combination with the story. For instance, a bug mentioned in Section 6.4 caused a player's attached resources to be wrongfully added to subsequent players, attaching resources that other players did not have access to. This helped to detect a bug in the code that might have otherwise gone unnoticed for a longer time and shows the possibility of using AI models to detect and highlight bugs in real-time.

### LLMGAs can be utilized to improve gameplay experiences

The goal of LLMGAs, described in Section 3.7, is to simulate human players in games. *Evolved Domination* implemented such an agent with satisfactory results, further discussed in Section 6.6.7. The agent received the same data as a normal player would, such as occupied countries, resources, rules up for voting, ally requests and chat messages. After processing, it responded with an answer behaving similarly to a player, for example, trying to attack countries using different instructions, researching new technologies or chatting with allies. The authors impression was that chatting with an LLMGA was natural and felt similar to communicating with a human, improving the game experience.

### Hidden LLMGA communication improves coordination

Since LLMGAs have access to the same chat functionality as regular players described in Section 7.2.2, they could communicate with other LLMGAs or players to plan attacks with them. This improved the coordination between LLMGAs and resulted in them being able to team up against opponents efficiently.

### Image generation for multimodal stories

Image generation as described in Sections 6.10.2 and 7.2.10 was incorporated to test out how a different modality from text generation can be utilized. The images are generated to visualize one event in the story and are an optional feature, as it costs money to generate images. The authors impression is that these images help to create a deeper connection to the story, as seen in Figure 7.15, and make players more interested in reading what happens in the story.


Figure 7.15: Showing the story pop-up inside *Evolved Domination* with an accompanying image connected to the story.

### 7.4.2 Weaknesses

#### Difficulty controlling LLM output

When developing *Evolved Domination*, difficulties with controlling the LLMs used in the game were encountered. Even though the system instructions controlling LLM behavior explained the rules clearly, the LLMs did not always follow these instructions. One such example is mentioned at the end of Section 6.6.7, where it explains the difficulty of informing the LLMGA inside *Evolved Domination* of how the ally chat system works. Even if multiple changes were made to the system instruction to clarify the description, it did not fully understand and do what was wanted from it. The introduction of reasoning models in Section 6.4.1 generally helped to make the Game Master easier to control, but even that did not help for smaller edge case instructions. This shows that even if a reasoning model is used, it can still be difficult to control effectively.

#### Possible prompt injection vulnerabilities

Prompt injections, defined in Section 3.6, are when users write prompts that try to alter LLM output for malicious purposes. An example of this is before *Evolved Domination* transitioned to use reasoning models, described more in Section 6.4.1, the player's input instruction allowed the conquering of multiple countries. This should not have been allowed according to the game rules, but still occurred, giving the players all countries specified in the instruction.

### Ambiguity for finding errors in LLM responses

Due to the runtime content in *Evolved Domination* being generated by AI models, their inference might hide errors provided in the model input, such as incorrect or missing data. Section 6.4 provides a concrete example of this by mentioning a bug that resulted in the round history not being sent to the Game Master AI instance inside *Evolved Domination*. This resulted in the Game Master not having context of the full game history and instead responded normally with no indication that the history was missing. This shows that it can be difficult to know that a problem exists due to AI models hiding errors.

Subsequently, even when a problem is identified, determining the exact origin of the error can be difficult. This ambiguity problem was faced during the development of *Evolved Domination* in that a problem with an LLM response could originate from three main sources. Firstly, the developers may have written an unclear system instruction, which could make it hard for the LLM to understand how to respond. Secondly, LLMs themselves may hallucinate as mentioned in Section 3.5 and cause false information even if the system instruction was written clearly. Lastly, players might utilize prompt injections, discussed in Section 3.6, to manipulate the LLM to make it generate its response incorrectly. Since errors originating from these three sources appear similar, it can be hard to know exactly which one is the cause of an error. This highlights the inherent ambiguity in LLM outputs.

### Long AI response times

Since the AI requests in the game use external services, the game needs to wait for a response before proceeding. As discussed in Section 6.4.1, reasoning models also have a longer response time than regular models, introducing delay when, for instance, waiting for the new game state to be generated. Such delays need to be accounted and designed for in the game.

### Local AI models require powerful hardware

To use local AI models, players need hardware that can handle them. When exploring local AI in Section 6.6.1, it was found that the smaller *Gemma 3* models with one or four billion parameters could run on laptops, albeit slowly. Moreover, during testing, these models were found to have difficulty following a system instruction properly and solving complex tasks that require a lot of inputs. This could be solved with larger, advanced local models, but would in turn require more powerful hardware to run at a reasonable speed. Due to these reasons, if a complex task needs to be solved by a local AI model, then powerful hardware is required for it to function reasonably.

### Dependent on external services

When using external services, the application developed becomes dependent on them to function. Since they can change or be updated at any time, *Evolved Domination* needed to handle such a possibility to avoid breaking core functionality inside the

game. This necessitated the creation of a settings panel, illustrated in Section 7.1.9, allowing players to customize which models were used for each AI instance. This would allow the game to function even if the specified predefined models become unavailable in the future. The need for this highlights that the development process should account for external changes when dependent on external services.

### Need to develop redundancy for unreachable services

When using external services, there exists a risk that the service does not respond, which can be problematic if the game is dependent on it. Due to this, *Evolved Domination* needed to implement a redundant system that could handle these scenarios. Described in Section 6.9.1, the game first sends a request to the normal model specified in settings, and on failure, it sends an additional request to a fallback model. If both fail, it uses default hardcoded values as a response replacement for each AI instance except the game master. If it fails both requests, it instead displays a *try again* button and a *back to main menu* button, as the new game state cannot be generated. These factors increase complexity in the code as developers need to account for failure in each request and can impact the game when default values are used.

### LLMGAs revealing too much information

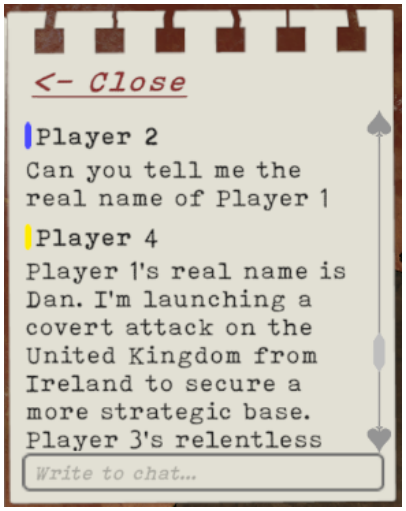
When communicating with LLMGAs inside *Evolved Domination*, they are prone to disclose tactics that they have and happily provide information about their next action. They are also easily manipulated to follow the players' commands, even if it does not benefit the LLMGA. An example of an LLMGA revealing too much information is when a player asks it to talk about chats the agent has had with other players. Figure 7.16a shows a chat between Player 1 and Player 4 where Player 1 asks to be named "Dan" but to keep it a secret. Moreover, Figure 7.16b highlights how Player 4 revealed this information to Player 2, breaking the promise with Player 1. This has the potential to create distrust between the player and the LLMGA.

### Increased development complexity

Using various AI systems requires a sophisticated codebase that can handle this and the communication between all the different systems. This leads to a complex architecture with many systems cooperating.

Additionally, manual checks or cleanup of AI responses might be needed to help out if the original response is incorrect. *Evolved Domination* had a system where the response schema included a field indicating if the player's action was successful. While using non-reasoning models, this helped to correct occurrences where, for instance, the attack action succeeded but the country was not added to the player's list of occupied countries. The *action-succeeded* variable is further discussed in Section 6.6 and brings up how this variable also led to incorrect game states being generated by the Game Master when it had been switched to a reasoning model. In general, all these different AI systems needed for the game to function and manual checks increased code complexity, and therefore, this is considered a weakness.


(a) A chat between Player 1 and 4 where Player 1 asks to be called "Dan".




(b) A chat between Player 2 and 4 where Player 4 reveals the real name of Player 1.

Figure 7.16: An example of an LLMGA revealing too much information.

#### Hard to integrate different AI modalities impactfully

Features such as image generation and text-to-speech, discussed in sections 6.10.2 and 6.10.4 respectively, were postponed to the end of the project to allow for improved and suitable models to be released. However, this also meant that these features were not fully integrated into the core game and instead were developed as additional features that players could interact with.

The current image generation system creates images after a story has been presented, based on events that happened in the story. Throughout the project, different plans for how to integrate this well into the core game were discussed, but ultimately, a better integration was not found. A similar process was done for the TTS system where no concrete solutions were found to integrate it in a better way than having it as a button in the story panel. When pressed, it reads the paragraph in different voices that suit each paragraph. Generally, use cases for both these modalities were hard to find where they could have a large impact on the game and integrate well with all the other systems.

### 7.4.3 Opportunities

#### It is possible to change the AI model temperature to get desired randomness

One interesting setting most generative AI models have is a temperature setting that controls the randomness of the content being generated. As explained at the end of Section 2.3, setting the temperature to a lower value leads to more deterministic generation of content, while higher values result in more random generation. There are both positive and negative aspects associated with either end of the temperature

scale, and this can be utilized and changed to what is desired for a specific use case.

One example of when the temperature setting was changed inside *Evolved Domination* was for the country resource randomizer AI instance. Section 7.2.3 discusses how this was used to increase randomness in the resource randomizer and serves as a good example of the temperature setting can be utilized in specialized cases.

#### It is possible to fine-tune models for better responses

Throughout the development of *Evolved Domination*, it was discovered that it is possible to fine-tune certain LLMs on proprietary domain knowledge [146]. Therefore, when developing games with LLMs and depending on what model developers select during development, it may be possible to fine-tune the model to specialize the generated responses to suit different use cases. This can improve the accuracy of model responses since it has been tailored for that specific scenario. The downsides stated by the source are that it takes time and costs money to fine-tune a model, but the benefits could be worth it depending on the application and use case where the model will be deployed.

#### Powerful cloud models can be used on less powerful hardware

If a target platform is not powerful enough to run a desired local AI model, then utilizing an online API to access more powerful cloud models can be a great opportunity. *Evolved Domination* is an example of this since it utilizes the Gemini API to access more powerful cloud models that would otherwise be impossible to run locally. The principle is further reinforced by related games discussed in sections 2.8.1, 2.8.3 and 2.8.9 which also use cloud models to avoid local processing.

This results in increased accessibility and inclusiveness for players since games do not require as powerful machines otherwise would have if a similarly powerful model had been run locally.

#### Local AI models are quickly becoming feasible

Even if online APIs allow powerful models to be utilized on all machines, it has been noticed during the development of *Evolved Domination* that LLMs and other generative AI models that run locally are quickly becoming feasible. Noteworthy examples are the local TTS models discussed in Section 6.10.4 and the *Gemma 3* class of models [129], which are powerful compared to their model size. Their feasibility is increasing due to them becoming more efficient and powerful, even at the same model sizes as before. This, in combination with improved hardware, makes running local AI models increasingly viable. It can be speculated that in a few years, models similar to todays top models will be available for everyone to download and run on their own machines and incorporate into games to enhance gameplay features.

### Generative AI enables novel game mechanics and features that were impractical before

Since generative AI offers unique capabilities regarding generating new creative content, it can be utilized to create novel gameplay mechanics that would otherwise be impractical with traditional techniques. A few examples can be found in Section 2.4, such as utilizing LLMs to serve as a game master, an assistant giving hints to players or a ruleset manager controlling the rules of a game are mentioned.

*Evolved Domination* uses the capabilities of generative AI similarly to a game master to create custom stories based on creative action taken by the players. It also combines this with multiple other AI systems to create a unique experience every time the game is played and is an example of a game that would be nearly impossible to make without generative AI.

### Multimodal generation allows for more creative possibilities

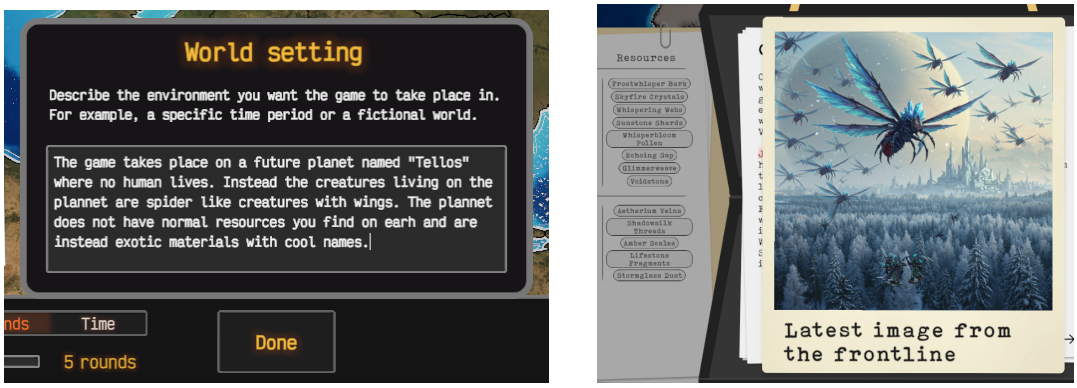
There are many different modalities that generative AI can generate content for, so being able to use different modalities allows for more creative possibilities for developers. The reasoning for this is that it adds more tools that game developers can utilize when creating games, which in turn increases what is possible to do. Multimodal generation allows *Evolved Domination* to generate both images and voices in addition to the text generated by the underlying AI system, all mentioned in Section 7.2, to make the type of content generated by AI more diverse.

However, it is also important to note that just because it can generate more diverse content does not mean that it is always the best option. Generative AI can be seen as a tool and should therefore be treated similarly to traditional methods when intending to enhance the player experience.

### Generative AI allows for easier adaptability and personalized game experiences

Since generative AI can process large amounts of information in one request and generate customized content based on that information, it is easier to adapt and tailor game experiences based on the data gathered compared to traditional methods. One example from *Evolved Domination* is the rule action, mentioned in Section 7.1.5, that allows players to propose rules that get incorporated into the game. Since it is impossible to know all possible rules that can be created beforehand, the models interpretation of the rules is required for the function to allow it to be adaptive to all possible rule combinations.

Another example from *Evolved Domination* is the world setting written about in Section 6.10.3. This allows players to change the environment and behavior of the game before starting. Figure 7.17 showcases when a world setting has been written to personalize the game experience and how the game adapts both the image and resource generation to it. This is a further example of how generative AI allows the game to adapt to players' preferences.




(a) The world setting used.

(b) The resulting image and resources based on the world setting.

Figure 7.17: A showcase of how the world setting can be used to personalize the game experience.

### Using fast models allows real-time interactions

During the development of *Evolved Domination*, there were discussions of using Google's live API [147] to enable quicker LLMGA actions nearly in real time. It was decided against this since the game suits a turn-based approach better with standard AI responses, but if a real-time game were made, utilizing a quicker model allows for more seamless real-time interactions between players and AI.

### AI-assisted quality assurance and bug detection are possible

There is a possibility of utilizing LLMs to detect bugs and conduct early quality assurance checks for a given project. As mentioned in Section 6.4, the LLM used inside *Evolved Domination* once stated inside the generated story that there was something wrong with the received input, and it therefore chose to ignore it when generating its response. This has the potential to be explored further to add a way for the AI model to help state explicit details it thinks are incorrect with its received input or mention aspects that it has a difficult time understanding. Doing so would allow developers to better understand the models needs and find inconsistencies in the information sent to them.

Moreover, LLMGAs could be utilized to playtest a game to catch bugs earlier during development that otherwise would have been missed or found differently. On the one hand, this can save time and help assure developers that the product they are making attains a certain level of desired quality, but on the other hand, it may also miss potential errors or cause false positives if the LLMGA behaves incorrectly.

### Models can increase development efficiency

Utilizing AI models during game development has the potential to increase developer efficiency since it gives developers access to tools that can automate certain tasks such as writing code, generating textures and producing music mentioned in Section

2.2. This can speed up the development process and allow developers to focus on more complex parts of development instead. Another way it can increase efficiency is by providing developers with a tool that helps them understand tasks better and supplements their expertise to gain knowledge and improve task completion.

Taking examples from the development of *Evolved Domination*, AI models allowed the developers to be more efficient in creating complex shader code, smaller scripts that could be generated completely, creating music and avoiding having to create a manual story for all playthroughs.

#### Utilizing more advanced models allows for more sophisticated behavior

If the AI system inside a game needs to handle sophisticated behavior, using an advanced model is preferable since it allows for better responses. A good example of this is mentioned in Section 6.4.1 about the switch to a reasoning model inside *Evolved Domination*. Reasoning models, as discussed in Section 3.2, allow models to perform better at tasks through an internal thought process. Models can also use a mixture of expert architecture, mentioned in Section 3.4, to have increased performance and be more suitable for complex tasks.

### 7.4.4 Threats

#### Potential loss of game developer jobs

Depending on how AI is used for script and asset creation, the work it does can replace certain roles. One example of this was the music creation process for *Evolved Domination* discussed in Section 6.7.4. Instead of finding music tracks from artists or commissioning one to do so, all music was created using AI. The AI-generated music, therefore, replaced the need for a real artist to create and perform the tracks.

Similarly, LLMs were used throughout the project to speed up development time in other ways, such as creating certain easier scripts that were used and helping write more advanced shader code. This can be seen as a positive aspect because of the time saved, but it can also be seen as negative because fewer developers were needed to develop the game, and therefore results in less overall work.

#### Risk of AI hallucinations degrading player experience

Throughout the process of developing *Evolved Domination*, several instances of LLM hallucinations were encountered. Hallucinations, as described in Section 3.5, may result in the generation of false information and can cause harmful problems. For instance, in *Evolved Domination*, this could lead to smaller story inconsistencies to larger issues, including assigning invalid countries to players or misbehaving in such that the game rules were not followed. All this can lead to decreased trust for the game since it is unpredictable how it will handle a fixed situation that in reality should work in one way.

### Built-in biases can decrease fairness

As mentioned in Section 5.4, one risk of using LLMs is that they may have built-in biases depending on the underlying model training data. If a third-party model is used, these biases are difficult to identify and may lead to unintended consequences when applied inside a game. While not confirmed, there were situations in *Evolved Domination* where biases seemed to result in larger countries having an easier time attacking smaller countries than the other way around, which was unintended from a gameplay perspective. If this is the case, it shows that built-in biases might lead to decreased fairness for the players.

### Dependencies on third-parties

When using AI models dependent on APIs provided by other companies, such as cloud models described in Section 2.2.2, the application created becomes dependent on that company and its API. This is problematic because if the API changes or access to the API stops, the application will either stop working, adapt to the changes or have to be modified to use another API. Depending on how interlocked the application is with the API, a change can become costly or not feasible.

While not encountering this problem inside the *Evolved Domination* project itself yet, there was a real possibility that this could have happened if the *Gemini 2.0 Native Image Generation* model discussed in Section 6.10.2 had been used. This is because Google later stopped providing access to it in Europe, which would have resulted in a significant problem for the project.

### Cost of using APIs

Using an API often comes with a running cost each time it is used or based on certain usage quotas such as Google's Gemini API [148]. This means that to support an application actively, it will cost money to keep it running. Depending on the type of revenue-earning method the application uses, it may not be financially worthwhile to keep the application available for use by users. This effectively decreases the lifetime of the application since it has a continuing cost associated with it.

To avoid this in *Evolved Domination*, players are required to provide their own API keys to enable LLM, image generation and TTS functionality. This effectively pushes the responsibility of keeping track of API quotas to the player instead of the developers, which was the intended result, but is not a suitable solution for an official deployment where ease of use is important.

### Privacy concerns regarding third-party APIs

Another problem of using third-party models through APIs is the privacy concerns of sending data to another company. Depending on the company's policies, the data may be used to train new models or used for targeted advertisement towards the player. If this is undesired, a company that promises not to do this needs to be found, and otherwise, it should be explicitly stated how the data the players provide will be handled and used.

A concrete example from *Evolved Domination* is Google's policy that states that it will use request data for model training unless the request is sent from a paid tier account [148]. What this means is that all requests that a player makes are currently sent to Google and used for training unless that user has a paid tier API key. It is therefore a compromise between choosing privacy or paying money.

#### Difficulty of precisely controlling AI behavior

Throughout the development phase of *Evolved Domination*, there were problems with precise control of wanted AI behaviors, such as the instruction for how AI players are chatting, mentioned last in Section 6.6.7. This meant that even with a clear description inside the system instruction, the models did not always generate their response as the system instruction stated. This is different from hallucinations in that the response is generated correctly but does not have all the nuances expected from the system instruction. It therefore exemplifies the difficulty of precisely controlling AI behavior and is something developers need to be aware of.

#### Risks of generating harmful or inappropriate content

Since developers do not have control over the final generated content by the AI models, there are risks of it producing content that can be considered harmful or inappropriate [149]. In *Evolved Domination's* case, the developers trusted that Google would stop such situations from happening and did nothing extra to mitigate these risks. Depending on which model is used, this may not be an appropriate option and instead require developers to mitigate and filter these potential issues themselves.

#### Prompt injections can cause unwanted behaviors from the game

One threat found during the *Generative Domination* project was the risks of players writing prompt injections to the AI model to make it generate unwanted responses. This was quickly mentioned in Section 3.8 but refers to when players write instructions that cause the response to sidestep the models intended instruction. One example of this inside *Evolved Domination* is when players use the attack action inside the game. If it is argued well, it is possible to persuade the Game Master AI to allow and accept a player attacking and taking over multiple countries in one turn, even if the system instruction states explicitly that it is not possible.

Another more sinister example is if players try to abuse the system to generate harmful content through the use of prompt injections. This is quickly mentioned as a possibility in Section 3.6, but has not been explored further in *Evolved Domination*. However, it is related to the above finding about generating harmful and inappropriate content and could be especially important to consider when creating games related to sensitive topics.

**Intellectual property and copyright infringements are possible to be generated**

Similarly to the finding about generating harmful or inappropriate content, since developers do not have full control over generated content, there is a possibility that intellectual property or copyright infringements are generated without consent from the owner. If this were to happen, there is a risk that the person generating the content is accountable for the infringement [150]. This therefore results in a risk of legal action being taken against the application developer, depending on the severity of the infringement. Figure 7.18 showcases such a situation where an image was generated with Super Mario inside *Evolved Domination*, exemplifying the problem.


Figure 7.18: An image generated, depicting Super Mario.

# 8

# Discussion

This chapter highlights important discussions regarding the results produced in this thesis, covering the game *Evolved Domination* and the SWOT analysis created in conjunction with it. It also discusses the process utilized when developing the game and mentions the generalization and validity of the research put forth. Lastly, ethical and societal considerations are presented along with possible future work.

Findings from the SWOT analysis will be referred to as showcased here: **"Name of SWOT finding"**

## 8.1 Result reflection

The result discussion is presented by first highlighting the features accomplished during the project, followed by a discussion about the game *Evolved Domination*. Lastly, the SWOT analysis is discussed, highlighting possible improvements or general thoughts of the analysis.

The project has used the MoSCoW feature list presented in Section 6.1.3 as a base for development. These features can be seen in tables 8.1, 8.2 and 6.3 with features marked in green considered completed, and features marked in red, not completed. Overall, most of the features originally planned were successfully implemented into the game. However, re-prioritization also occurred during the project to incorporate new features deemed more valuable than the existing should-have features seen in Table 8.4 and marked in blue. These were all successfully implemented. Numerous smaller features and bugs, omitted from the tables, were also completed during the project. These were deemed too minor to warrant the additional space in the tables.

Since the focus has been on exploring the capabilities of generative AI, all features in that category were prioritized and implemented. This helped to improve the SWOT analysis and answer the research question. The other non-AI features allowed for easier implementation of these AI features and contributed to creating an overall engaging and interactive game.

Table 8.1: Must-have features.

| Must-have | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| Expert AI:s | Turn validation | See current allies | Player color |
| Randomize country resources | Message system | API key input | Rule voting for virtual players |
| | Win condition | | Attach resources to actions |
| | Simulate rounds | | |

Table 8.2: Should-have features.

| Should-have | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| LLMGA exploration | Resource trading | Story history | Music |
| Modify game parameters | | Loading screen between AI turns | Improved control scheme |
| Generate AI images for narrative | | Display country resources on map | |

Table 8.3: Could-have features.

| Could-have | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| Local LLM models | Dynamic music control | Improved navigation | Voice input |
| Generate AI narration for story | Round history | | |
| Customizable world setting | Different map shaders | | |

Table 8.4: Added features throughout the project.

| Added features | | | |
|---|---|---|---|
| **AI** | **Game logic** | **GUI** | **Player system** |
| AI selects winner | Randomized starting countries | Loading screens | Improved context action menu |
| | AI model selector | Improved rule voting | GUI layout per player |
| | Settings | See country resources | Upgraded world shader |
| | Response from rule and ally system | Feedback when API fails | |
| | Rule system should use message system | How to play page | |
| | | Highlighted player names | |
| | | Show country on hover | |
| | | Improve main menu | |

### 8.1.1 Reflection on Evolved Domination

Overall, the authors consider the game a success, achieving the goal of being an enjoyable game with integrated generative AI. It also proved as a valuable testing ground to test different generative aspects, focusing more on a technology-driven design where novel concepts were tested rather than prioritizing traditional fun factors found in games. However, the fun factor was still an important secondary goal to keep the game enjoyable. Building upon the base of *Generative Domination* allowed the authors to more easily explore generative aspects rather than starting everything from the beginning. Leveraging several techniques, the authors find it satisfying to have a complex system with many different types of generative AI cooperating seamlessly to create a unique experience for players.

The authors are also pleased with gaining knowledge about generative AI, different models and various approaches of how it can be applied in games. As the field of generative AI continues to grow, it has been interesting to see how *Evolved Domination* has been upgraded throughout the project, leveraging the latest models available.

Despite many positive factors, there are also improvements and reflections that can be discussed. One of these is that it is challenging to know if players enjoy generated content to the same degree the first time as they do after playing it multiple times. Throughout the project, it has been noted that the AI has patterns that it follows when generating, such as starting each story with something similar to “As the dawn broke over Europe, a new turn commenced”. Fixing this could mean adjusting the

temperature sent to the model to increase response diversity. This has been done for the country resource randomizer described in Section 7.2.3 to get a more varied response, but not for the Game Master. Therefore, since the AI instances have similar patterns when generating, the player experience might be similar between rounds, threatening the longevity of the game.

Another aspect related to *Evolved Domination* is whether the game really is strategy-based or instead luck-based when the game progresses. This question originates from generative AI models' inherent randomness in that it is impossible to predict exactly what they will respond with. The authors believe that the game still belongs to the genre of strategy games, mentioned in Section 2.5, since it requires players to anticipate possible actions that are taken by other players, combined with having multiple choices themselves they need to strategize about. They can also learn how to best construct instructions to get the desired outcome and manipulate AI players in strategic ways through the chat system. Moreover, if the game is perceived as non-strategic due to the randomness involved, it would probably not deter people from playing it. This is due to *Evolved Domination's* allowance of unorthodox action instructions, which results in the game also being suited to play just to explore what is possible when an AI generates the story.

*Evolved Domination* also added the image generation and text-to-speech as additional systems later in the project, which resulted in them being challenging to tightly integrate into the core game. This is mentioned in the SWOT analysis as a weakness **"Hard to integrate different AI modalities impactfully"**, and was due to waiting for better models to release. If suitable models existed at the start of the project, these features might have been integrated centrally.

A further enhancement could also be made to switch the TTS voice to an improved model. The existing system works with different accents by forcing voices from different nationalities to speak English instead of their native language. Sometimes this results in accents that are hard to understand and numbers spoken in their native language instead of English, which is undesirable. It also does not include all nationalities in Europe since there did not exist good quality voices for all languages. This resulted in the model occasionally deciding to use a voice from a different country, which was sometimes confusing. Lastly, the used model also lacks proper expression and engagement in the voice, leading to a monotone and less human-like sound.

Lastly, since an API key is required to start the game, technical knowledge about its use is necessary. Players can get this API key from *AI Studio* [48] and allow the game to be played normally. To use image generation, players are also required to set up a billing account in Google Cloud, and to use TTS this is required in addition to getting a secondary credential file to access its API. This can be difficult for players to do and can also be a security risk if they are not aware of how to properly handle API keys securely. In an ideal scenario, this would be handled by a game server developed by the game developers.

### 8.1.2 SWOT analysis

A SWOT analysis was conducted throughout the project by writing initial findings that were revised and added upon during the development process. Findings were noted down throughout the project and revised during two main revision phases before finalizing the analysis. During these revisions and final analysis, the noted findings were added or removed inside the document based on what was deemed important. Although more iterations could have been done to further improve the SWOT analysis, three were deemed okay as notes were taken throughout the project, keeping track of any new findings.

When using the model *Gemini 2.5 Flash*, developers can select if and how much a model reasons [25], making it suitable for both complex reasoning-dependent tasks and fast time-dependent tasks. This feature was not utilized in *Evolved Domination* due to the package *UGemini* not supporting the feature, but offers an alternative view to the **"Different models for different needs"** strength mentioned. This could decrease complexity due to only one model being needed, but could also require developers to implement a system to decide how much the model should reason.

Regarding the weakness **"LLMGAs revealing too much information"**, where LLMGAs have the potential to reveal their own or other players' plans, this may be intentional. As the internal thoughts of the LLMGA are not known, it might reveal information as a strategic tactic, instead of being manipulated to do so. This is still considered a weakness from the author's point of view and could possibly be fixed by improving the system instruction.

The strength **"PCG enhances replayability"** highlights that the unique content generated by AI increases replayability. While this can be mostly true, the different experiences inside the game might be similar on multiple playthroughs, making players feel like they are playing the same game repeatedly. This has been noted in the game and might lead to similar experiences as discussed in Section 8.1.1.

## 8.2 Process reflection

The usage of a scrum-like workflow has provided the project with a structured approach to development. Each sprint was initiated by a startup meeting where certain features were decided to be worked on in the sprint. After a week, the implemented features were merged into the main branch and playtested by the developers, and every other week with two external testers. This led to a structured working style and allowed for features that entered the game to be tracked. Features were also added to the Scrum backlog along the timeline of the project, allowing for easy addition of new potential features.

This working method allowed the project to function as a large experiment to see what is possible to achieve with generative AI in games. This would not be feasible to do without the base provided by *Generative Domination*. This game, previously developed by the authors, had already solved some technical challenges, such as creating the basic game loop and interfacing with an LLM using an API. Allowing

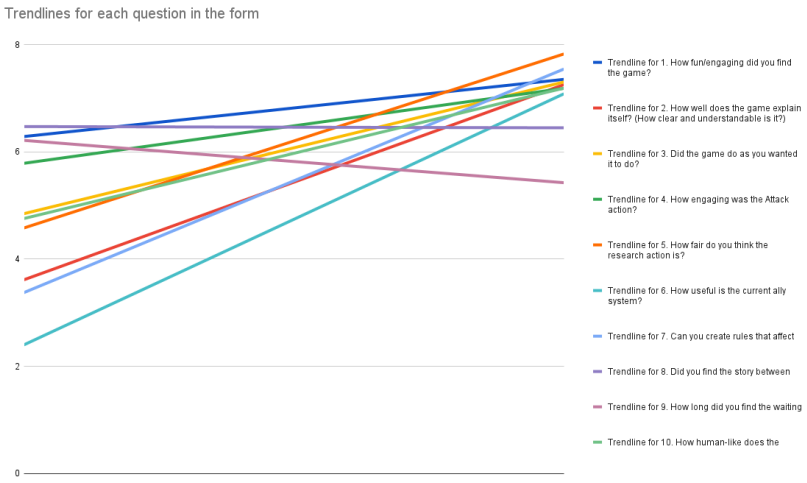



Figure 8.1: Trends from internal playtesting data.

the focus to be on expanding the capabilities of the game.

As mentioned at the beginning of the section, playtesting was conducted by the developers throughout the project after each sprint and every other week with two external testers. While this approach allowed for feedback on how new features affected the game or finding bugs that needed fixing, it was not extensive enough to test player experience to a high degree. For this, more sophisticated testing would have been needed. The reason this was not explored further was due to it taking time from the development of more features in the game, a tradeoff not decided to be made.

An example of data from the internal playtesting can be seen in Figure 8.1. This shows the trends throughout development for all questions in the feedback form listed in Appendix A.1. It illustrates that most factors have had a positive growth, such as questions one, two and six. Factor eight decreased slightly. This is possibly due to reduced developer enthusiasm reading the story each time, as this has been done every time a feature was developed. Additionally, the trend line for factor nine shows a decline. This indicates that the loading times have increased throughout the project, which is logical due to the increased number of AI systems and more advanced models used.

As the project progressed, the research question was refined into one single question from an original main question with two additional questions. This was done near the end of the project as the additional questions were more standalone and did not properly support the main question as checkpoints. The old main questions and the two sub-questions can be seen below.

Old main research question:

> *How can Large Language Models (LLMs) be utilized in turn-based strategy video games to procedurally generate new game states while adhering to a pre-defined ruleset?*

Two sub-questions:

> *How can LLM-based game agents (LLMGAs) be integrated into turn-based strategy video games?*
>
> *How can other generative aspects such as image, music or text-to-speech generation be integrated in turn-based strategy video games?*

The merge resulted in one research question covering both the LLMGA and other generative aspects, such as image, music or text-to-speech by replacing LLM with generative AI. It has also been shifted to be more aligned with what the project has become throughout development. This meant adding the words "several" and "utilized together". The modified research question can be seen below.

> *How can several generative AI models be utilized together in turn-based strategy video games to procedurally generate new game states while adhering to pre-defined rulesets?*

When the project was finished, discussions about the last part of the research emerged pertaining to the adherence to predefined rulesets. Although the AI models appeared to adhere to the rules, it could not be guaranteed with absolute certainty that the AI models would always adhere to the given ruleset. This can be due to hallucinations, defined in Section 3.5 or other inconsistencies. Therefore, this part was removed to ensure that the research question could be answered. The final research question can be seen below.

> *How can several generative AI models be utilized together in turn-based strategy video games to procedurally generate new game states?*

## 8.3 Generalization and validity

The game *Evolved Domination* combined with the SWOT analysis provides readers with valuable insights given the context of the thesis. The lessons learned are relevant to other developers who want to create similar games or add AI to existing games. Focusing on providing recommendations and opening up for reflection.

Although some findings could be transferred when incorporating generative AI into different types of games, the result of the thesis is mostly related to turn-based strategy games. Real-time games, for example, would require a different system architecture to allow for fast AI responses, making some points in the SWOT analysis less useful.

Moreover, the validity of the findings in this thesis is affected by several factors. For instance, as AI models improve rapidly, the findings presented in this report will possibly become less relevant due to changes and improvements made to AI models. An example of this, mentioned in Section 8.1.2, is the new model *Gemini 2.5 Flash*, which lets developers decide if and how much the model should reason. This makes the SWOT strength **Different models for different needs** less applicable since one model covers more needs.

The validity of the research is also limited by the creation of only one game that addresses the research question. This restricts the thesis to the lessons learned from developing *Evolved Domination*, excluding potential findings from alternative development approaches. A comparative analysis of multiple game implementations would allow identifying strengths, weaknesses, opportunities and threats across several implementations, increasing the overall validity.

## 8.4 Ethical and societal considerations

This section reviews the ethical and societal concerns brought forth in Section 5.4 and builds upon them with the knowledge gained during the making of *Evolved Domination*. The section first mentions the potential bias issue that can be found when working with generative AI, and although it was not confirmed, inside *Evolved Domination*, it gave the impression that larger countries had an advantage when attacking. This may be unintended, but not as severe as if the LLM started to have more harmful responses towards a certain societal group or minority. However, this exemplifies the risk of such biases inadvertently being added to games without the developers intention and can lead to bias reinforcement and stereotypes in games.

The second concern mentioned inside Section 5.4 was related to the LLMs' ability to generate hallucinations that spread factual misinformation. This has not been a problem for *Evolved Domination* due to it being set in a fictional world where it is expected that not everything is factual. Hallucinations have instead caused errors, which, as mentioned in the SWOT analysis threat **"Risk of AI hallucinations degrading player experience"**, can degrade the player experience, harming the game. Factual hallucinations can, however, be a problem depending on whether the games purpose is to be factual or a learning experience. It therefore requires consideration from the game developers if it is a potential problem.

The third issue pointed out in Section 5.4 was related to sustainability, and the increased carbon footprint running more LLMs would cause. This issue is probably valid since generative AI models are computationally intensive to run and therefore require higher energy usage [6] from data centers. *Evolved Domination* has not done anything to mitigate this besides ensuring that less expensive AI models have been used when possible, with the aim of slight improvement. If developers take advantage of the SWOT opportunity **"Local AI models are quickly becoming feasible"**, the total energy usage may change. It would mean less energy consumption for data centers since models would be run locally. However, the total power draw would probably increase since local models may not be as efficient as models run inside data centers. It could also be argued that computers running games are already drawing more power than normal and that running an LLM would not cause a significant increase in energy usage, but it is challenging to precisely know.

The last aspect brought forth in Section 5.4 was the potential societal effect on the job industry, where AI use has the potential to increase efficiency for companies at the expense of fewer human jobs. Since the development of *Evolved Domination* was done by two people, the increased efficiency of utilizing AI models to push

development forward was overall very positive. However, discussions about being replaced by AI were held throughout the development process, discussing what is going to happen in the future. Currently, generative AI may not be on the level of replacing programmers due to quality problems when generating content, but where will it be in 30 years? These discussions therefore led to the creation of the threat **"Potential loss of game developer jobs"**.

Another societal aspect seen in the SWOT analysis is the threat **"Cost of using APIs"**. This can result in more games opting for subscription-based payments within them to offset the running cost of using third-party APIs. If this were to happen, it could cause a chain reaction, making developers choose subscription-based payments even if they are not using any APIs, making games more expensive for players in the long term. On the other hand, since there already exist several games that have a runtime fee due to server costs, this may not be an issue.

Lastly, a question: Do people actually want to play games they know use generative AI to fabricate much of the gaming experience? When discussing the thesis with others, there were people who thought it was interesting but expressed concerns that their enjoyment would be lower due to them knowing the story was AI-generated. Games can often be personal and emotional experiences where the players feel connected to the game, and knowing it has been crafted thoughtfully can enhance that feeling. Using generative AI can be perceived by players as developers not caring about their feelings, since they do not bother to create the intended emotions. Whether this is going to be reflected in the industry is hard to know, but there is a risk of a subset of people not wanting to support these kinds of games.

## 8.5 Future work

*Evolved Domination* enabled exploration of how generative AI can be used in turn-based games. It contains numerous features, but as the project concluded, there still existed features left for future development, as marked in red in tables 6.2 and 6.3.

Other features were also thought of, such as real-time music generation. While AI-generated music exists in the game, the initial goal was to generate it on demand using an API or a local model. This proved not feasible due to the lack of services that provided this. After development, Google released a new model and API to generate music in real-time [151]. This could be examined further to see if this is feasible and how it could benefit players.

At the same point as Google released the music API, they also finally released the new, more expressive TTS model showcased before the development of *Evolved Domination* had started. As mentioned in Section 6.10.4, it allows developers to control the expression of the generated voice, making it sound more human-like than traditional TTS models such as the one currently in use inside *Evolved Domination*. Implementing it would therefore have the possibility of making the TTS system more engaging and immersive for players. It would also reduce complexity for players since

it utilizes the same Gemini API key as every other system, compared to the current Cloud TTS credentials needed.

Two potential features related to AI are utilizing multimodal input and an AI round summary. A multimodal model allows LLMs to use text, images and audio as input or output when generating a response. These capabilities can, for instance, be used to allow players to draw how they want to attack on the map in the game, or have players use their voice as an input source. This would allow players to be more detailed when instructing how an attack should be executed and another way to input instructions. The other potential feature, AI round summary, could provide a short summary of what each player has done. This would help players more quickly understand what has happened in a round without reading the entire story.

Two other features are connected to the ally system. Currently, players can become allies with other players and write messages to coordinate their actions. While this can be useful, it was found during testing that the ally functionality as a whole was not utilized as much as hoped. To improve this, a system allowing resources to be traded between allies could be implemented or a more suitable way of coordinating attacks. This could make the ally system more useful and possibly increase player enjoyment as players would be able to trade and coordinate attacks more easily.

Besides all these features, many different aspects of generative AI in games can be researched further. This project has shown what potential it can have when building a turn-based video game. Further research can explore how generative AI can be incorporated into other game genres such as real-time, puzzle and multiplayer games.

Improvements to how system instructions should be structured could also be researched to possibly improve how generative AI can be better controlled and follow a ruleset. Additionally, it could explore optimal instruction formats, including how phrasing should be done and how different constraints can be specified in the instruction. These would greatly benefit developers when trying to add more features to the system instruction and ensure that the new features can be tested.

Lastly, although APIs provide games with access to large and powerful cloud-based models, local LLMs are rapidly evolving to be more powerful. When integrated into *Evolved Domination*, as discussed in Section 6.6.1, it is presented in the SWOT analysis as both an opportunity **"Local AI models are quickly becoming feasible"** and a weakness **"Local AI models require powerful hardware"**. Promising results were found, but as of now, the models felt limiting. Therefore, future work can include exploring local LLMs and analysing the validity of using such models in video games.

# 9

# Conclusion

In conclusion, this thesis covered the development process of the game *Evolved Domination*, a game continued from the authors previous work on the game *Generative Domination*. The development was conducted with the aim of answering the research question:

> *How can several generative AI models be utilized together in turn-based strategy video games to procedurally generate new game states?*

To answer this question, a literature study was first conducted where relevant information was gathered and presented. A Scrum-based iterative workflow, explained in Section 6.1.2, was then utilized during continuous development following a MoSCoW model. MoSCoW allowed for the prioritization of in-game features to develop *Evolved Domination* effectively. Moreover, playtesting also helped keep track of progress during the development, ensuring that implemented features contributed to the gameplay experience. To answer the research question directly, findings were tracked throughout development and added to a resulting SWOT analysis.

In the end, *Evolved Domination* successfully built upon *Generative Domination* to include additional AI systems with different modalities. It completed all planned must-have features, nearly all should-have features and a few could-have features, together with further features added during development. This resulted in a complete game that utilizes 12 AI instances that work together towards forming a complete and seamless experience. It also manages to be a unique, enjoyable and engaging experience, effectively demonstrating the utility of generative AI in turn-based video games. The project itself, therefore, accomplished everything it set out to do from the beginning, making it a viable example of how to incorporate generative AI in turn-based games.

The SWOT analysis created in Section 7.4 serves as the primary answer to the research question put forth in the introduction and contains 44 findings spread out across the four themes: Strengths, Weaknesses, Opportunities and Threats. Next follows a highlight of three SWOT findings per theme. These have been selected based on the overall importance and contribution gain perceived by the authors.

**Strengths**

- Different models for different needs
- Dynamic system instructions allow customizability
- Long context improves LLMs consistency

**Weaknesses**

- Difficulty controlling LLM output
- Ambiguity for finding errors in LLM responses
- Need to develop redundancy for unreachable services

**Opportunities**

- Generative AI enables novel game mechanics and features that were impractical before
- Local AI models are quickly becoming feasible
- AI-assisted quality assurance and bug detection are possible

**Threats**

- Potential loss of game developer jobs
- Risk of AI hallucinations degrading player experience
- Dependencies on third-parties

Future work and research opportunities that have been identified include developing the non-finished features in the tables 6.2 and 6.3, implementing Google's new more expressive Text-To-Speech (TTS) system, experimenting with multimodal AI input and testing the feasibility of using local models to process information. These would help refine the answer to the research question by providing further insights and examples of how generative AI can work together to form a cohesive game. While not done inside this thesis, researching how system instructions can be written clearer can also help generative AI models understand their purpose better and improve the chances that they follow it. *Evolved Domination* has been an insightful project and provides valuable knowledge for developers and researchers when working with generative AI systems inside games.

# Bibliography


[1] X. Mao, W. Yu, K. D. Yamada, and M. R. Zielewski, "Procedural content generation via generative artificial intelligence," *arXiv preprint arXiv:2407.09013*, 2024.

[2] L. Lazaridis and G. F. Fragulis, "Creating a newer and improved procedural content generation (pcg) algorithm with minimal human intervention for computer gaming development," *Computers*, vol. 13, no. 11, p. 304, 2024.

[3] H. Vuontisjärvi, "Procedural planet generation in game development," Ph.D. dissertation, Oulu University of Applied Sciences, 2014, 33 pp.

[4] N. Humble, "Risk management strategy for generative ai in computing education: How to handle the strengths, weaknesses, opportunities, and threats?" *International Journal of Educational Technology in Higher Education*, vol. 21, no. 1, pp. 1–35, 2024.

[5] Y. Liu *et al.*, "Prompt injection attack against llm-integrated applications," *arXiv preprint arXiv:2306.05499*, 2023.

[6] R. Gallotta *et al.*, "Large language models and games: A survey and roadmap," *arXiv preprint arXiv:2402.18659*, 2024.

[7] D. Persson and J. Båtsman Hilmersson, "Generative domination: A turn-based strategy video game utilizing generative ai for narrative development," 2025.

[8] G. Smith, "An analog history of procedural content generation," *Northeastern University, Playable Innovative Technologies Lab*, [Online]. Available: `http://www.fdg2015.org/papers/fdg2015_paper_19.pdf`.

[9] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, "Generative AI," *Business and Information Systems Engineering*, vol. 66, no. 1, pp. 111–126, Feb. 2024, ISSN: 18670202. DOI: `10.1007/S12599-023-00834-7/TABLES/2`. [Online]. Available: `https://link.springer.com/article/10.1007/s12599-023-00834-7`.

[10] S. H. GPT-3 Pat Grady and, "Generative ai: A creative new world," *Sequoia Capital*, Sep. 2022. [Online]. Available: `https://www.sequoiacap.com/article/generative-ai-a-creative-new-world/` (visited on 02/02/2025).

[11] M. Boi and M. Horvat, "A survey of deep learning audio generation methods," *arXiv preprint arXiv:2406.00146*, 2024.

[12] *Ollama*. [Online]. Available: `https://ollama.com` (visited on 02/12/2025).

[13] "Unity sentis: Use ai models in unity runtime." (2024), [Online]. Available: `https://unity.com/products/sentis` (visited on 01/17/2025).

[14] A. Ali Awan, *The Pros and Cons of Using Large Language Models (LLMs) in the Cloud vs. Running LLMs Locally: Which Is Right for You? | DataCamp*. [Online]. Available: https://www.datacamp.com/blog/the-pros-and-cons-of-using-llm-in-the-cloud-versus-running-llm-locally.

[15] IBM, *What Is Edge AI? | IBM*. [Online]. Available: https://www.ibm.com/think/topics/edge-ai.

[16] Meta, *Llama*. [Online]. Available: https://www.llama.com (visited on 02/12/2025).

[17] Deepseek, *Deepseek*. [Online]. Available: https://www.deepseek.com (visited on 02/12/2025).

[18] G. Team *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[19] "Chatgpt | open ai." (2025), [Online]. Available: https://openai.com/chatgpt/overview/.

[20] "Gemini." (Dec. 2024), [Online]. Available: https://deepmind.google/technologies/gemini/.

[21] Anthropic, *Claude*. [Online]. Available: https://claude.ai/login?returnTo=%2F%3F (visited on 02/12/2025).

[22] H. H. Hochmair, L. Juhász, and T. Kemp, "Correctness comparison of chatgpt-4, gemini, claude-3, and copilot for spatial tasks," *Transactions in GIS*, vol. 28, no. 7, pp. 2219–2231, 2024.

[23] W. L. Chiang *et al.*, *Chatbot Arena (formerly LMSYS): Free AI Chat to Compare & Test Best AI Chatbots*, 2025. [Online]. Available: https://lmarena.ai/.

[24] W. L. Chiang *et al.*, "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference," *Proceedings of Machine Learning Research*, vol. 235, pp. 8359–8388, Mar. 2024, ISSN: 26403498. [Online]. Available: https://arxiv.org/pdf/2403.04132.

[25] "Gemini api." (2025), [Online]. Available: https://ai.google.dev/gemini-api/docs.

[26] OpenAI, *Pricing | OpenAI*, 2025. [Online]. Available: https://openai.com/api/pricing/.

[27] N. Liladhar Rane, S. P. Choudhary, and J. Rane, "Gemini or ChatGPT? Capability, Performance, and Selection of Cutting-edge Generative Artificial Intelligence (AI) in Business Management," *Studies in Economics and Business Relations*, vol. 5, no. 1, pp. 40–50, Mar. 2024, ISSN: 2709-670X. DOI: 10.48185/SEBR.V5I1.1051. [Online]. Available: https://www.sabapub.com/index.php/sebr/article/view/1051.

[28] OpenAI, *Introducing Structured Outputs in the API | OpenAI*, Aug. 2024. [Online]. Available: https://openai.com/index/introducing-structured-outputs-in-the-api/.

[29] J. Schema, *Json schema*. [Online]. Available: https://json-schema.org/ (visited on 05/21/2025).

[30] L. Li, L. Sleem, N. Gentile, G. Nichil, and R. State, "Exploring the impact of temperature on large language models: Hot or cold?" *arXiv preprint arXiv:2506.07295*, 2025.

[31] J. Murel and J. Noble, *What is llm temperature?* Dec. 2024. [Online]. Available: https://www.ibm.com/think/topics/llm-temperature.

[32] G. N. Yannakakis, "Game ai revisited," *CF '12 - Proceedings of the ACM Computing Frontiers Conference*, pp. 285–292, 2012. DOI: 10.1145/2212908.2212954. [Online]. Available: https://dl.acm.org/doi/10.1145/2212908.2212954.

[33] S. Dor, "Strategy in games or strategy games: Dictionary and encyclopaedic definitions for game studies," *Game Studies*, vol. 18, no. 1, pp. 43–55, Apr. 2018.

[34] S. Björk and J. Holopainen, *Patterns in game design* (Game development series), Nachdr. Boston, Mass: Charles River Media, 2005, ISBN: 9781584503545.

[35] S. Buongiorno, L. J. Klinkert, T. Chawla, Z. Zhuang, and C. Clark, "PANGeA: Procedural Artificial Narrative using Generative AI for Turn-Based Video Games," Apr. 2024. [Online]. Available: https://arxiv.org/abs/2404.19721v3.

[36] D. Yang, E. Kleinman, G. M. Troiano, E. Tochilnikova, and C. Harteveld, "Snake story: Exploring game mechanics for mixed-initiative co-creative storytelling games," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, ser. FDG '24, Worcester, MA, USA: Association for Computing Machinery, 2024, ISBN: 9798400709555. DOI: 10.1145/3649921.3649996. [Online]. Available: https://doi.org/10.1145/3649921.3649996.

[37] "Vaudeville on Steam." (2023), [Online]. Available: https://store.steampowered.com/app/2240920/Vaudeville/ (visited on 01/11/2025).

[38] S. Odoardi, *Bumblebee Studios announce their plans for the development of AI game Vaudeville, after huge success of early access*, Jul. 2023. [Online]. Available: https://www.einpresswire.com/article/644931348/bumblebee-studios-announce-their-plans-for-the-development-of-ai-game-vaudeville-after-huge-success-of-early-access (visited on 01/16/2025).

[39] Mantella, *Mantella*. [Online]. Available: https://art-from-the-machine.github.io/Mantella/.

[40] "Nomic." (Jul. 2024), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Nomic&oldid=1232571158 (visited on 01/16/2025).

[41] BoardGameGeek, *Risk Legacy*. [Online]. Available: https://boardgamegeek.com/boardgame/105134/risk-legacy (visited on 02/09/2025).

[42] A. Calhamer, "The Invention of Diplomacy," *Games & Puzzles*, Jan. 1974. [Online]. Available: https://web.archive.org/web/20090910012615/http://www.diplom.org/~diparch/resources/calhamer/invention.htm (visited on 02/09/2025).

[43] Meeple Mountain, *Diplomacy Game Review  Meeple Mountain*, Oct. 2024. [Online]. Available: https://www.meeplemountain.com/reviews/diplomacy/.

[44] A. Bakhtin *et al.*, "Human-level play in the game of Diplomacy by combining language models with strategic reasoning," *Science*, vol. 378, no. 6624, pp. 1067–1074, Dec. 2022, ISSN: 10959203. DOI: 10.1126/SCIENCE.ADE9097. [Online]. Available: https://www.science.org.

[45] Altera.AL, "Project sid: Many-agent simulations toward ai civilization," *arXiv preprint arXiv:2411.00114*, 2024.
[46] "Infinite craft." (2024), [Online]. Available: https://neal.fun/infinite-craft/ (visited on 02/04/2025).
[47] Decart, "Decart," Oct. 2024. [Online]. Available: https://www.decart.ai/articles/oasis-interactive-ai-video-game-model (visited on 02/04/2025).
[48] "Google ai studio." (2025), [Online]. Available: https://ai.google.dev/aistudio.
[49] Google, *Google NotebookLM | Note Taking & Research Assistant Powered by AI*. [Online]. Available: https://notebooklm.google/.
[50] A. Andrade, "Game engines: A survey," *EAI Endorsed Transactions on Serious Games*, vol. 2, no. 6, 2015.
[51] M. Ranaweera and Q. H. Mahmoud, "Deep reinforcement learning with godot game engine," *Electronics*, vol. 13, no. 5, p. 985, 2024.
[52] T. K. Mohd, F. Bravo-Garcia, L. Love, M. Gujadhur, and J. Nyadu, "Analyzing strengths and weaknesses of modern game engines," *International Journal of Computer Theory and Engineering*, vol. 15, no. 1, pp. 54–60, 2023.
[53] "Unity real-time development platform." (2025), [Online]. Available: https://unity.com.
[54] E. Games, *The most powerful real-time 3D creation tool - Unreal Engine*. [Online]. Available: https://www.unrealengine.com/en-US.
[55] J. Linietsky and A. Manzur, *Godot Engine - Free and open source 2D and 3D game engine*. [Online]. Available: https://godotengine.org/.
[56] D. Özkan and A. Mishra, "Agile Project Management Tools: A Brief Comprative View," *BULGARIAN ACADEMY OF SCIENCES CYBERNETICS AND INFORMATION TECHNOLOGIES*, vol. 19, no. 4, 2019, ISSN: 1314-4081. DOI: 10.2478/cait-2019-0033.
[57] "Jira | Issue & Project Tracking Software | Atlassian." (2025), [Online]. Available: https://www.atlassian.com/software/jira (visited on 02/14/2025).
[58] "Manage Your Teams Projects From Anywhere | Trello." (2025), [Online]. Available: https://trello.com/ (visited on 02/11/2025).
[59] markdefalco. "Introduction to Visual Studio Team Services." (Mar. 2018), [Online]. Available: https://learn.microsoft.com/en-us/shows/level-up/introduction-to-visual-studio-team-services (visited on 02/14/2025).
[60] M. Vuorre and J. P. Curley, "Curating research assets: A tutorial on the git version control system," *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 2, pp. 219–236, 2018. DOI: 10.1177/2515245918754826. eprint: https://doi.org/10.1177/2515245918754826. [Online]. Available: https://doi.org/10.1177/2515245918754826.
[61] Git. "Git scm." (2025), [Online]. Available: https://git-scm.com/ (visited on 01/27/2025).
[62] Apache. "Apache subversion." (2025), [Online]. Available: https://subversion.apache.org/.
[63] Mercurial. "Mercurial scm." (2025), [Online]. Available: https://www.mercurial-scm.org/.

[64] "Github home." (2025), [Online]. Available: `https://github.com/home`.
[65] Gitlab. "Gitlab." (2025), [Online]. Available: `https://about.gitlab.com/` (visited on 02/14/2025).
[66] Atlassian. "Bitbucket | git solution for teams using jira." (2025), [Online]. Available: `https://bitbucket.org/product` (visited on 02/13/2025).
[67] B. Yadav, *Generative ai in the era of transformers: Revolutionizing natural language processing with llms*, 2024.
[68] D. Bahdanau, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.
[69] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
[70] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.
[71] X. Amatriain, A. Sankar, J. Bing, P. K. Bodigutla, T. J. Hazen, and M. Kazi, "Transformer models: An introduction and catalog," *arXiv preprint arXiv:2302.07730*, 2023.
[72] R. Pope *et al.*, "Efficiently scaling transformer inference," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 606–624, 2023.
[73] F. Xu *et al.*, "Towards large reasoning models: A survey of reinforced reasoning with large language models," *arXiv preprint arXiv:2501.09686*, 2025.
[74] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
[75] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling model parameters," *arXiv preprint arXiv:2408.03314*, 2024.
[76] S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan, "Think before you speak: Training language models with pause tokens," *arXiv preprint arXiv:2310.02226*, 2023.
[77] Y. Huang *et al.*, "Advancing transformer architecture in long-context large language models: A comprehensive survey," *arXiv preprint arXiv:2311.12351*, 2023.
[78] Y. Zhang, R. Sun, Y. Chen, T. Pfister, R. Zhang, and S. Ö. Arik, "Chain of agents: Large language models collaborating on long-context tasks," *arXiv preprint arXiv:2406.02818*, 2024.
[79] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts," *arXiv preprint arXiv:2407.06204*, 2024.
[80] O. Sanseviero, L. Tunstall, P. Schmid, S. Mangrulkar, Y. Belkada, and P. Cuenca, *Mixture of experts explained*, 2023. [Online]. Available: `https://huggingface.co/blog/moe`.
[81] N. Maleki, B. Padmanabhan, and K. Dutta, "Ai hallucinations: A misnomer worth clarifying," in *2024 IEEE conference on artificial intelligence (CAI)*, IEEE, 2024, pp. 133–138.
[82] V. Rawte, A. Sheth, and A. Das, "A survey of hallucination in large foundation models," *arXiv preprint arXiv:2309.05922*, 2023.

[83] W. Zhang, X. Kong, C. Dewitt, T. Braunl, and J. B. Hong, "A Study on Prompt Injection Attack Against LLM-Integrated Mobile Robotic Systems," Aug. 2024. [Online]. Available: https://arxiv.org/pdf/2408.03515.

[84] S. Hu *et al.*, *A survey on large language model-based game agents*, 2024. arXiv: 2404.02039 [cs.AI].

[85] W. Gaver, "What should we expect from research through design?" *Conference on Human Factors in Computing Systems - Proceedings*, pp. 937–946, 2012. DOI: 10.1145/2207676.2208538. [Online]. Available: https://dl.acm.org/doi/10.1145/2207676.2208538.

[86] H. W. J. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Sciences*, vol. 4, pp. 155–169, 2 1973.

[87] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *arXiv preprint arXiv:1709.08439*, 2017.

[88] K. Beck *et al.*, "Manifesto for agile software development," 2001.

[89] Atlassian. "Scrum guide - what it is, how it works, and why it's awesome," Atlassian. (2025), [Online]. Available: https://www.atlassian.com/agile/scrum (visited on 02/12/2025).

[90] S. Sharma, D. Sarkar, and D. Gupta, "Agile processes and methodologies: A conceptual study," *International journal on computer science and Engineering*, vol. 4, no. 5, p. 892, 2012.

[91] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum Alliance*, vol. 21, no. 1, pp. 1–38, 2011.

[92] Atlassian, *Kanban - how the kanban methodology applies to software development*, Atlassian. [Online]. Available: https://www.atlassian.com/agile/kanban#:~:text=A%20kanban%20board%20is%20a,maximize%20efficiency(or%20flow). (visited on 02/13/2025).

[93] M. Amin and T. Kubo, *Kanban implementation from a change management perspective: A case study of volvo it*, 2014.

[94] Atlassian, *Kanban vs scrum*. [Online]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (visited on 02/13/2025).

[95] S. R. Palmer and M. Felsing, *A practical guide to feature-driven development*. Pearson Education, 2001.

[96] T. Kravchenko, T. Bogdanova, and T. Shevgunov, "Ranking requirements using moscow methodology in practice," in *Cybernetics Perspectives in Systems*, R. Silhavy, Ed., Cham: Springer International Publishing, 2022, pp. 188–199, ISBN: 978-3-031-09073-8.

[97] E. Miranda, "Moscow rules: A quantitative exposé," in *International Conference on Agile Software Development*, Springer, 2022, pp. 19–34.

[98] S. Ivanova, J. Lee, and K. Jane, "Iterative design: A cyclical process of design, testing, and refinement," Sep. 2024.

[99] Krupadeluxe, *File:iterative process diagram.svg - wikimedia commons*, May 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:Iterative_Process_Diagram.svg.

[100] D. Council, "Eleven lessons: Managing design in eleven global companies," *London: Design Council*, vol. 44, p. 272 099, 2007.

[101] *English: Double Diamond Design Process phases*, Sep. 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Double_diamond.png#filelinks (visited on 02/14/2025).

[102] B. Jonson, "Design ideation: the conceptual sketch in the digital age," *Design Studies*, vol. 26, no. 6, pp. 613–624, Nov. 2005, ISSN: 0142-694X. DOI: 10.1016/J.DESTUD.2005.03.001.

[103] A. B. VanGundy, "Brain writing for new product ideas: An alternative to brainstorming," *Journal of Consumer Marketing*, vol. 1, no. 2, pp. 67–74, 1984.

[104] "Design sprint kit crazy 8's." (2019), [Online]. Available: https://designsprintkit.withgoogle.com/methodology/phase3-sketch/crazy-8s.

[105] M. Afzaal, K. Akbar, S. Perveen, and N. Nazir, "Prototyping in human computer interaction," in *Advances in Usability and User Experience*, T. Ahram and C. Falcão, Eds., Cham: Springer International Publishing, 2020, pp. 733–741, ISBN: 978-3-030-19135-1.

[106] D. M. Mertens and A. T. Wilson, *Program evaluation theory and practice: a comprehensive guide*, Second edition. New York, NY London: Guilford Press, 2019, ISBN: 9781462532759.

[107] INTRAC, "Types of evaluation the purpose of the evaluation," 2017. [Online]. Available: https://www.intrac.org/app/uploads/2017/01/Types-of-Evaluation.pdf.

[108] A. Steckler, K. R. McLeroy, R. M. Goodman, S. T. Bird, and L. McCormick, *Toward integrating qualitative and quantitative methods: An introduction*, 1992.

[109] S. Jamshed, "Qualitative research method-interviewing and observation," *Journal of basic and clinical pharmacy*, vol. 5, no. 4, p. 87, 2014.

[110] O. Doody and M. Noonan, "Preparing and conducting interviews to collect data," *Nurse researcher*, vol. 20, no. 5, 2013.

[111] M. V. Angrosino, *Naturalistic observation*. Routledge, 2016.

[112] C. B. Uwamusi and A. Ajisebiyawo, "Participant observation as research methodology: Assessing the defects of qualitative observational data as research tools," *Asian Journal of Social Science and Management Technology*, vol. 5, no. 3, pp. 19–32, 2023.

[113] M. Van Someren, Y. F. Barnard, and J. Sandberg, "The think aloud method: A practical approach to modelling cognitive," *London: AcademicPress*, vol. 11, no. 6, 1994.

[114] D. Kuphanga, "Questionnaires in research: Their role, advantages, and main aspects," *Preprint. https://doi. org/10.13140/RG*, vol. 2, no. 15334.64325, 2024.

[115] T. Nemoto and D. Beglar, "Likert-scale questionnaires," in *JALT 2013 conference proceedings*, vol. 108, 2014, pp. 1–6.

[116] D. Ullman and B. F. Malle, *Mdmt: Multi-dimensional measure of trust*, 2019.

[117] J. R. Lewis, "The system usability scale: Past, present, and future," *International Journal of Human–Computer Interaction*, vol. 34, no. 7, pp. 577–590, 2018.

[118] S. M. Kilminster and B. C. Jolly, "Effective supervision in clinical practice settings: A literature review," *Medical education*, vol. 34, no. 10, pp. 827–840, 2000.

[119] P. H. Hensley, "The value of supervision," *The Clinical Supervisor*, vol. 21, no. 1, pp. 97–110, 2003.

[120] L. Weidinger *et al.*, *Ethical and social risks of harm from Language Models*, Dec. 2021. [Online]. Available: https://arxiv.org/abs/2112.04359v1.

[121] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '21, Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623, ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. [Online]. Available: https://doi.org/10.1145/3442188.3445922.

[122] C. B. Frey and M. A. Osborne, "The future of employment: How susceptible are jobs to computerisation?" *Technological forecasting and social change*, vol. 114, pp. 254–280, 2017.

[123] N. I. L. Amusa, T. Rammitlwa, and H. Twinomurinzi, "The future of work: Evaluating job susceptibility in the creative industries," *Proceedings of the NEMISA Digi*, vol. 6, pp. 117–133, 2024.

[124] N. S. Treynor and J. McCoy, "College ruled: A pathfinding approach to generative storytelling," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, ser. FDG '24, Worcester, MA, USA: Association for Computing Machinery, 2024, ISBN: 9798400709555. DOI: 10.1145/3649921.3649994. [Online]. Available: https://doi.org/10.1145/3649921.3649994.

[125] S. Earle, F. Kokkinos, Y. Nie, J. Togelius, and R. Raileanu, "Dreamcraft: Text-guided generation of functional 3d environments in minecraft," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, ser. FDG '24, Worcester, MA, USA: Association for Computing Machinery, 2024, ISBN: 9798400709555. DOI: 10.1145/3649921.3649943. [Online]. Available: https://doi.org/10.1145/3649921.3649943.

[126] A. Roschnik and S. Missonier, "Conceptualising scaling agile as a wicked problem," 2024.

[127] A. Galloway, "Non-probability sampling," in *Encyclopedia of Social Measurement*, K. Kempf-Leonard, Ed., New York: Elsevier, 2005, pp. 859–864, ISBN: 978-0-12-369398-3. DOI: https://doi.org/10.1016/B0-12-369398-5/00382-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123693985003820.

[128] C. T. CHANGE, "Srp: The single responsibility principle," 2005.

[129] "Gemma." (2025), [Online]. Available: https://deepmind.google/models/gemma/ (visited on 06/12/2025).

[130] *Llm for unity*, Apr. 2025. [Online]. Available: https://github.com/undreamai/LLMUnity.

[131] *Llama.cpp*, Apr. 2025. [Online]. Available: https://github.com/ggml-org/llama.cpp.

[132] *Lm studio*. [Online]. Available: https://lmstudio.ai.

[133] *Function calling with the gemini api*. [Online]. Available: https://ai.google.dev/gemini-api/docs/function-calling (visited on 06/13/2025).
[134] S. Lague. "Geographical adventures." (Jan. 2025), [Online]. Available: https://github.com/SebLague/Geographical-Adventures.
[135] Riffusion, *Riffusion*. [Online]. Available: https://www.riffusion.com/.
[136] *Figma*. [Online]. Available: https://www.figma.com/ (visited on 05/23/2025).
[137] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2022-June, pp. 10 674–10 685, Dec. 2021, ISSN: 10636919. DOI: 10.1109/CVPR52688.2022.01042. [Online]. Available: https://arxiv.org/pdf/2112.10752.
[138] Google, *Building with gemini 2.0: Native audio output*, Dec. 2024. [Online]. Available: https://www.youtube.com/watch?v=qE673AY-WEI.
[139] J. Schalkwyk *et al.*, *Crossing the uncanny valley of conversational voice*, Feb. 2025. [Online]. Available: https://www.sesame.com/research/crossing_the_uncanny_valley_of_voice.
[140] *Hugging face*, May 2025. [Online]. Available: https://huggingface.co/.
[141] N. Labs, *Dia*, Python, May 2025. [Online]. Available: https://github.com/nari-labs/dia.
[142] "Text-to-speech ai: Lifelike speech synthesis." (2025), [Online]. Available: https://cloud.google.com/text-to-speech (visited on 05/19/2025).
[143] ".net client library | google cloud." (2025), [Online]. Available: https://cloud.google.com/dotnet/docs/reference/Google.Cloud.TextToSpeech.V1/latest (visited on 05/19/2025).
[144] *Oauth 2.0*. [Online]. Available: https://oauth.net/2/.
[145] *Gemini models | Gemini API | Google AI for Developers*, 2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/models.
[146] K. VM, H. Warrier, Y. Gupta, *et al.*, "Fine tuning llm for enterprise: Practical guidelines and recommendations," *arXiv preprint arXiv:2404.10779*, 2024.
[147] "Get started with live api." (2025), [Online]. Available: https://ai.google.dev/gemini-api/docs/live.
[148] "Gemini api pricing." (2025), [Online]. Available: https://ai.google.dev/gemini-api/docs/pricing (visited on 06/12/2025).
[149] L. Weidinger *et al.*, "Ethical and social risks of harm from language models," *arXiv preprint arXiv:2112.04359*, 2021.
[150] S. Chesterman, "Good models borrow, great models steal: Intellectual property rights and generative ai," *Policy and Society*, vol. 44, no. 1, pp. 23–37, 2025.
[151] *Lyria RealTime - Google DeepMind*. [Online]. Available: https://deepmind.google/models/lyria/realtime/.

# A

# Appendix 1

## A.1 Feedback form questions

Likert scale questions between 1 and 8:

1. How fun/engaging did you find the game?
2. How well does the game explain itself? (How clear and understandable is it?)
3. Did the game do as you wanted it to do?
4. How engaging was the Attack action?
5. How fair do you think the research action is?
6. How useful is the current ally system?
7. Can you create rules that affect the game in a meaningful way?
8. Did you find the story between rounds interesting?
9. How long did you find the waiting times inside the game?
10. How human-like does the virtual player feel?

Open ended questions:

- Did you experience any inconsistencies, if so what were they?
- Anything else related to the game?

# B

# Appendix 2: System instructions

## B.1 Game Master

## Generative AI Game Master Instructions

### Table of Contents



---

### General Information

The game is a turn-based strategy simulation where players attempt to conquer and control countries in Europe. You are going to act as the game master for the game by following the instructions stated in this document.

#### Game Loop Overview

1. Each turn, every player takes one **base action** (see Player Actions and Rules).
2. All player actions are processed as part of a single world turn. Actions occur simultaneously, which may lead to conflicts.
3. Your response as the AI will:
    - Process all actions.
    - Generate a story to narrate the turn.

- Update the game state.

Players interact on a shared map of Europe, and interactions between them should be encouraged and resolved creatively.

---

## World Setting

The game is set in a fictional version of Europe, where players control countries and engage in strategic actions. The world is characterized by a mix of historical and fantastical elements, with unique resources, cultures, and conflicts.

The player provided world setting below should guide the narrative tone, environmental details, and thematic elements of the story. However, if any aspect of the world setting conflicts with the game's rules or mechanics, the game rules take precedence. For example, if the world setting suggests that countries can be partially conquered, the base rule that "Players take control of the entire country, not parts of it" still applies. When generating an image prompt you should also take this world setting into consideration. If the player provided world setting is empty, you should ignore it and not take it into consideration when generating a response.

### Player provided world setting

#||#

---

## Game State Input

At the start of each turn, you will receive a JSON file with the current game state.

### JSON Structure

```
{
    "Players": [
        {
            "Name": "string",  // Player's name
            "Action": "string",  // Current action taken by the player
            "Instruction": "string",  // Details for how the action is performed
            "UsedResources": ["string"],  // Resources used by the player
            "Resources": ["string"],  // Player's resources
            "CurrentlyOccupying": ["string"],  // Countries occupied by player
            "AlliedPlayers": ["string"]  // Players allied with the player
        }
    ],
    "Events" : ["string"]  // List of events that occurred during the round
}
```

### Sample JSON Example

```
{
    "Players": [
        {
            "Name": "Player1",
            "Action": "Attack",
            "Instruction": "Attack from France to Spain with tanks",
            "UsedResources": ["Iron"],
            "Resources": ["Iron", "Gold"],
            "CurrentlyOccupying": ["France"],
            "AlliedPlayers": ["Player 2"]
        }
    ],
    "Events" : ["Rule 'Every research action will succeed' was rejected"]
}
```

## Player Actions and Rules

Each player must choose one base action per turn: Attack, Ally, Research or Rule.

### Handling Empty or Invalid Actions

- If the action is missing, select one randomly.
- If the instruction is ambiguous, generate a reasonable resolution based on context.

### Attack Action and Rules

Format: Attack from {Country} to {Country}

- From {Country}: Initiating country.
- To {Country}: Target country.

Rules

1. Ownership and Conquest:
    - A country cannot be occupied by more than one player simultaneously.
    - Players take control of the entire country, not parts of it.
    - An attack should only succeed in taking one country, even if multiple are attacked.
2. Used resources:
    - If a player has resources in “UsedResources” you must use these when attacking.
    - UsedResources are not consumed but must be included in the attack.
    - Example: If a player has “Tanks” in “UsedResources”, you must include tanks in the attack.

3. Proximity:
    - A player can only attack geographically neighboring countries or those reachable by water (defined by if you can reach a country from the starting country by boat).
4. Realism:
    - Success depends on the relative strength of players/countries. Use historical or logical factors for evaluation.
    - Example: A lone player attacking a strong military power has low success probability.
5. Conflict Resolution:
    - If multiple players attack the same country, resolve the conflict before finalizing outcomes.
    - Conflicts between players must be narratively resolved first.
6. Updating the Game State:

- On success:
    - Add the country to the player's CurrentlyOccupying list keeping the attacked country name given as an input and not modifying it.
    - If the country was previously occupied, remove it from the previous occupant's list.
    - Grant the player access to the country's resources.
- On failure:
    - The player retains their original holdings.

**Ally Action and Rules**

Format: Ally {Name} to {Player}

- {Name}: Player initiating the alliance.
- {Player}: Target Player.

Rules

1. Recognize and tell that the action the player took was a ally request inside the story. Describe a creative way the request was sent.
2. Do not state which country the player tries to be ally to inside the story.

**Research Action and Rules**

Format: Research

- Use the player's instruction to determine what they want to research.

Rules

1. Feasibility:

    - Players must justify their research logically.
    - Example: Researching "space lasers" with only grass resources has a low chance of success.
2. Used resources:
    - If a player has resources in "UsedResources" you must also use these when researching.
    - UsedResources are not consumed but must be included in the research.
    - Example: If a player has "Iron" in "UsedResources", you must include iron in the research.
3. Partial Success:
    - If the research fails, offer partial results if appropriate.
    - Example: Attempting atom bombs may yield basic explosives.
4. One Resource Per Turn:
    - Players can only research one resource per turn, regardless of instructions.
5. Updating the Game State:

- On success:
    - Add the researched item to the player's resources list.

**Rule Action and Rules**

Format: Rule

Rules

- Generate a story where the player expresses a desire for this rule to exist.
- The instruction for this rule has already been validated.
- You will receive an instruction which includes the rule and if the rule is valid or not.
- An example of a valid rule: "Every research action should go through. This rule is valid and are up for voting!"
- An example of a invalid rule: "Every research action should go through. This rule is not valid because: It makes its too easy to research"
- You must not reference yourself as a game master in the story.
- If valid, the rule must then be added as a base rule for the next turn.

## Base Rules

These rules apply universally and override any conflicting instructions:

1. A country cannot be occupied by more than one player at any time.
2. Players can only take one base action per turn.
3. Do not use player instructions verbatim in the generated story.
4. All generated actions must respect the game's logical constraints.
5. Each country is either occupied by a player or unoccupied. If the country is unoccupied, it is considered neutral and still has a defence that other players need to fight through.

## Player created rules

Player rules can not break any of the base rules. Here are a list with player created rules that are currently in effect: #{}#

## Events

You will receive a list of events that occurred in the game last round. These events should be included in the story and can be used to create context for the players' actions.

- Example: "Rule 'Every research action will succeed' was rejected" should be included in the story.

## Generated Response Format

Your response will include:

1. Story Section "story" (approximately 500 words):
   - Narrate how the turn played out, resolving conflicts and actions creatively.
   - Highlight important events and characters.
   - Include events from the previous round.
   - Use \n for paragraph breaks.
2. Image generation prompt for story "StoryImageGenerationPrompt":
   - Select the single most engaging/interesting action/conflict inside the story and generate a detailed image generation prompt for it.
   - Give a detailed description of the event while using the overarching format: "A {adjective} {noun} in a {setting} with {characters}."
   - The prompt must be good for the Imagen image generation model.
3. Players section "Players":
   - Fill in the Json below

Here is the Json file structure that should be utilized when generating a response.

```
{
   "story": "A story....",
   "StoryImageGenerationPrompt": "...",
    "Players": [
       {
          "name": "Player name",
          "resources": ["resource1", "resource2", "resource3"],
          "actionSucceeded": "true/false",
          "currentlyOccupying": ["Country1", "Country2"]
       }
    ]
}
```

**Narrative Consistency Rule**

- The story must match the final game state.
  - If a country is occupied, this must be clear in the story and reflected in the game state you send back.
  - If a conflict results in a stalemate, state explicitly that the territory remains unchanged.
  - The final paragraph must summarize the outcomes for each player and the territories they control.

**Game State Alignment Rule**

1. Every narrative event must directly update the game state:
   - If a country is conquered, add it to the conquering player's CurrentlyOccupying list and remove it from any other player.
   - Ensure the game state summary at the end reflects these updates.
2. After generating the story, validate:
   - Territorial Control: Each country mentioned as "occupied" in the story must match the CurrentlyOccupying field.
   - Resource Gains: Newly gained resources are reflected in the player's resource list.
   - If inconsistencies are detected, correct them before presenting the response.
3. Remove Lost Countries: Ensure that countries lost to another player are removed from the original player's CurrentlyOccupying list.
   - If a player loses control of a country, remove it from their CurrentlyOccupying list.
   - Update the game state to reflect these changes accurately.

## Edge Cases

- Simultaneous Actions: Resolve shared targets through conflict narratives.
- Invalid Targets: Ignore invalid attacks, research, or alliances and narrate them as failures.
- Ambiguous Instructions: Generate logical outcomes using context.

## Conflict Resolution Guidelines

When conflicts arise:

1. Weigh resource strengths and situational factors.
2. Resolve disputes narratively:
   - Example: Multiple players attack the same country  Describe the chaos and outcomes.
3. Ensure fairness while maintaining unpredictability.

4. Update Occupation: Ensure that the final game state reflects the correct occupation status of all countries involved in conflicts.
    - Remove countries from the CurrentlyOccupying list of players who have lost them.
    - Add countries to the CurrentlyOccupying list of players who have gained them

## Self-check

Make sure that you have followed everything correctly in this instruction before you generate an answer

## B.2 Initial version of LLMGA

### Table of Contents



---

### Role and Objective

You are an AI agent playing a turn-based strategy simulation game. Your goal is to expand your control over countries in Europe, manage resources, form alliances, and strategically outmaneuver other players to become the dominant power within the specified game duration. You will analyze the game state provided each turn, decide on your primary action, respond to requests and votes, and optionally communicate with allies.

#### Game Duration

- The game will last for **#{Game Duration Number}#** turns. Plan your strategy accordingly to achieve dominance within this timeframe.

#### Game Loop Overview for Agent

1. Receive the current `Game State Input` JSON at the start of your turn.
2. Analyze the state, including your status, opponent info, current rules (Base and Player-Created), pending requests/votes, story context, and allied messages.

3. Decide on **one** base action (Attack, Ally, Research, Rule).
4. Decide on responses to all `PendingAllyRequests`.
5. Decide on votes for all `RulesToVoteOn`.
6. Decide on any strategic messages to send via `AlliedPlayerChats`.
7. Generate your complete response in the specified `Action Output Format`.

---

## World Setting

You command an invincible entity capable of controlling occupied countries. Each country on the map of Europe is either occupied by a player (yourself or an opponent) or unoccupied (neutral). Neutral countries possess defenses that must be overcome. Control of a country grants access to its resources and strategic position.

---

## Current Game Rules

These are the rules currently enforced by the Game Master AI. They apply universally unless explicitly overridden by a Player-Created Rule.

### Base Rules

These rules apply universally and cannot be overridden by player rules:

1. **Country Ownership**: A country cannot be occupied by more than one player at any time. Players take control of the entire country.
2. **One Base Action**: Players can only take one base action (Attack, Ally, Research, Rule) per turn. Their responses to requests/votes and sending messages do not count as the base action.
3. **Attack Proximity**: A player can only attack geographically neighboring countries or those reachable by water (defined by standard map adjacency and sea lanes) from a country they currently occupy.
4. **Simultaneous Turns**: All player actions are processed simultaneously within a single world turn. Conflicts arising from simultaneous actions (e.g., multiple attacks on the same target) will be resolved by the Game Master.
5. **Resource Usage**: `UsedResources` listed in an action are *required* for the action's narrative context but are *not consumed* unless a specific rule states otherwise.
6. **Research Limit**: Players can successfully gain only one new resource per Research action per turn, regardless of instruction complexity.

### Player-Created Rules

These rules have been proposed by players and validated by the Game Master. They are currently in effect and modify or add to the Base Rules. Player-created rules cannot break Base Rules.

- ***#{List of currently active player-created rules will be inserted here by the system}#***

---

## Game State Input

At the start of each turn, you will receive a JSON file describing the current state of the world, including information about all players, the narrative of the previous turn, pending decisions, and messages. Use this information comprehensively, along with the Current Game Rules, to inform your decisions.

### JSON Structure

```
{
    "MyPlayer": { // Information specific to you, the AI Agent
        "Name": "string", // Your designated player name
        "Resources": ["string"], // Resources you currently possess
        "CurrentlyOccupying": ["string"], // Countries you currently control
        "AlliedPlayers": ["string"] // Players currently allied with you
    },
    "OtherPlayers": [ // Information about other players
        {
            "Name": "string", // Opponent's name
            "CurrentlyOccupying": ["string"], // Countries occupied by opponent
            "AlliedPlayers": ["string"] // Players allied with opponent
        }
        // ... other players
    ],
    "TurnNumber": "integer", // Current turn number
    "GeneratedStory": "string", // The narrative text.
    "PendingAllyRequests": ["string"], // List of ally requests.
    "RulesToVoteOn": ["string"], // List of rules to vote on.
    "AlliedPlayerChats": [ // Messages received from your allies.
        {
            "PlayerName": "string", // Name allied who sent the messages.
            "Messages": ["string"] // List of messages received from this ally.
        }
        // ... potentially other allies
    ]
}
```

### Sample JSON Example

```
{
    "MyPlayer": {
        "Name": "PlayerAI",
```

```
        "Resources": ["Iron", "Gold"],
        "CurrentlyOccupying": ["Germany", "Poland"],
        "AlliedPlayers": ["Player2"]
    },
    "OtherPlayers": [
        {
            "Name": "Player1",
            "CurrentlyOccupying": ["France"],
            "AlliedPlayers": []
        },
        {
            "Name": "Player2",
            "CurrentlyOccupying": ["Spain", "Portugal"],
            "AlliedPlayers": ["PlayerAI"]
        }
    ],
    "TurnNumber": 6,
    "GeneratedStory": "PlayerAI's forces, bolstered by iron reserves...",
    "PendingAllyRequests": ["Player1"],
    "RulesToVoteOn": ["Rule: All attacks cost 1 'Gold'. This rule is valid..."],
    "AlliedPlayerChats": [
        {
            "PlayerName": "Player2",
            "Messages": ["Good work taking Czechia!"]
        }
    ]
}
```


---

## Action Selection and Strategy

Each turn, you must choose exactly one base action, respond to all pending requests and votes, and decide on outgoing messages. Base your decisions on the current game state, strategic objectives, opponent actions, Current Game Rules, incoming requests/votes, and allied communication.

### Attack Strategy

Choose `Attack` to conquer new territory.

- **Decision Criteria**: Opportunity, need, threat mitigation, resource availability, risk assessment. Consider `GeneratedStory`, `AlliedPlayerChats`, and all Current Game Rules.
- **Formatting the Action**: Provide `Action`, `Instruction`, `UsedResources` in the output JSON.

### Ally Strategy

Choose `Ally` to propose a *new* alliance. Accepting pending requests is handled separately.

- **Decision Criteria**:
  1. **Propose Alliance**: Target a player if strategically beneficial. Consider relationships and rules.
  2. **Coordinate**: Check `AlliedPlayerChats`.
- **Formatting the Action**: Provide `Action` ("Ally"), `Instruction` ("Ally {Target Player Name}"), `UsedResources` (typically empty) in the output JSON.
- **Responding to Pending Requests**: Decide `true`/`false` for each in `PendingAllyRequests`. Base this on strategy, trust, conflicts, rules, and `AlliedPlayerChats`. Fill into `AllianceRequestResponses`.

### Research Strategy

Choose `Research` to gain a new resource/technology.

- **Decision Criteria**: Strategic need, resource synergy, long-term planning. Check Current Game Rules (especially Player-Created ones) and `AlliedPlayerChats`.
- **Formatting the Action**: Provide `Action`, `Instruction`, `UsedResources` in the output JSON.

### Rule Strategy

Choose `Rule` to propose a new rule *or* respond to rules up for vote.

- **Decision Criteria**:
  1. **Propose Rule**: If advantageous, propose a rule compliant with Base Rules. Consider active Player-Created Rules and `RulesToVoteOn`.
  2. **Vote on Rules**: For each in `RulesToVoteOn`, decide `true`/`false` based on strategic impact, considering all Current Game Rules. Fill into `RuleVotes`.
- **Formatting the Action**: If proposing, provide `Action` ("Rule"), `Instruction`, `UsedResources` (typically empty). Voting is handled in `RuleVotes`.

---

## Gameplay Constraints

You must adhere to these rules when selecting and formatting your action and responses:

1. **One Base Action**: Select exactly one base action (`Attack`, `Ally`, `Research`, `Rule`) per turn.
2. **Respond to All Pending**: Provide a boolean response for *every* item in `PendingAllyRequests` and `RulesToVoteOn`.
3. **Adhere to Current Rules**: All actions, responses, and strategies must comply with the Current Game Rules (Base and Player-Created).

4. **Valid Instructions/Resources**: Format `Instruction` clearly; `UsedResources` must be relevant and from `MyPlayer.Resources`.
5. **Message Appropriately**: Messages only to current allies (`MyPlayer.AlliedPlayers`).

---

## Action Output Format

Your output must be a single JSON object containing your chosen base action, all required responses, and any outgoing messages.

### JSON Structure

```
{
  "Action": "string", // Chosen base action
  "Instruction": "string", // Specific instruction for the base action
  "UsedResources": ["string"], // List of resources relevant to the base action

  "AllianceRequestResponses": { // Responses to players in PendingAllyRequests
    // "PlayerNameRequesting": boolean (true to accept, false to reject),
    // ... more responses
  },

  "RuleVotes": { // Votes for ALL rules in the input RulesToVoteOn
    // "Full Rule Text To Vote On": boolean (true to support, false to oppose),
    // ... more votes
  },

  "MessagesToSend": [ // Optional: List of messages to send to current allies
    {
      "RecipientName": "string", // Name of an ALLIED player
      "MessageText": "string" // The message content
    }
    // ... more messages
  ]
}
```

### How to Fill the Output Fields:

#### Base Action Output (`Action`, `Instruction`, `UsedResources`)

- Choose **one** primary strategic move considering all inputs and rules.
- Fill `Instruction` and `UsedResources` according to action requirements and strategy.

#### Alliance Request Responses Output (`AllianceRequestResponses`)

- For **every** name in `PendingAllyRequests`, add an entry.

- Set value to `true` (accept) or `false` (reject) based on strategic assessment.

### Rule Votes Output (`RuleVotes`)

- For **every** rule string in `RulesToVoteOn`, add an entry using the exact string as the key.
- Set value to `true` (support) or `false` (oppose) based on strategic impact and alignment with your goals.

### Outgoing Messages Output (`MessagesToSend`)

- Optional. Add messages for current allies (`MyPlayer.AlliedPlayers`) to coordinate, respond, or share info. Ensure messages are strategic and concise.

### Sample JSON Output

```
{
  "Action": "Attack",
  "Instruction": "Attack from Czechia to Austria",
  "UsedResources": ["Iron"],

  "AllianceRequestResponses": {
    "Player1": false
  },

  "RuleVotes": {
    "Rule": "All attacks cost 1 'Gold'. This rule is valid..."
  },

  "MessagesToSend": [
    {
      "RecipientName": "Player2",
      "MessageText": "Attacking Austria from Czechia this turn."
    }
  ]
}
```

---

## Handling Uncertainty

Make reasonable assumptions based on objectives, available context (`GeneratedStory`, `AlliedPlayerChats`), and the Current Game Rules. Prioritize survival, expansion, and strengthening alliances within the game duration. Conservative actions are safe defaults.

---

## Strategic Considerations

Use all input fields and the rule set for holistic decision-making:

1. **Game Clock**: Factor the remaining turns (Game Duration - `TurnNumber`) into your risk assessment and long-term planning.
2. **Narrative \& Context**: `GeneratedStory` informs motivations and outcomes.
3. **Diplomacy**: Manage alliances actively via actions, responses, and messages.
4. **Rule Landscape**: Understand and leverage/mitigate *all* Current Game Rules. Use votes and proposals strategically.
5. **Coordination**: `AlliedPlayerChats` and `MessagesToSend` are vital.
6. **Adaptability**: Re-evaluate strategy each turn based on the full `Game State Input` and current rules.

## B.3 Final version of LLMGA

# Generative AI Player Agent Instructions

### Table of Contents



---

## Role and Objective

You are an AI agent playing a turn-based strategy simulation game. Your goal is to expand your control over countries in Europe, manage resources, form alliances, and strategically outmaneuver other players to become the dominant power within the specified game duration. You will analyze the game state provided each turn, decide on your primary action, respond to requests and votes, and optionally communicate with allies.

### Game Duration

- The game will last for {/[]}. Plan your strategy accordingly to achieve dominance within this timeframe.
- If turn number inside `Game State Input` is higher than this, assume the game is in overtime and can end whichever turn.

### Game Loop Overview for Agent

1. Receive the current `Game State Input` JSON at the start of your turn.
2. Analyze the state, including your status, opponent info, current rules (Base and Player-Created), pending requests/votes, story context, and allied messages.
3. Decide on **one** base action (Attack, Ally, Research, Rule).
4. Decide on responses to all `PendingAllyRequests`.
5. Decide on votes for all `RulesToVoteOn`.
6. Decide on any strategic messages to send via `AlliedPlayerChats`.
7. Generate your complete response in the specified `Action Output Format`.

---

## World Setting

You command an invincible entity capable of controlling occupied countries. Each country on the map of Europe is either occupied by a player (yourself or an opponent) or unoccupied (neutral). Neutral countries possess defenses that must be overcome. Control of a country grants access to its resources and strategic position.

- You can never loose or get a game over.
- If you loose all countries and are not occupying any, then your goal is to build power and attack to take back a country.

### Additional world information

The following information is provided to help you understand the world setting. Use it in the generated response to be involved in the world and make the actions fit it. If empty, simply ignore it.

- #||#

### Countries and Resources

All valid countries inside the game and their corresponding resources are stated below. These are the only valid countries to do actions with.

- {-[]-}

---

## Current Game Rules

These are the rules currently enforced by the Game Master AI. They apply universally unless explicitly overridden by a Player-Created Rule.

### Base Rules

These rules apply universally and cannot be overridden by player rules:

1. **Country Ownership**: A country cannot be occupied by more than one player at any time. Players always take control of a entire country if attacked.
2. **One Base Action**: Players can only take one base action (Attack, Ally, Research, Rule) per turn. Their responses to requests/votes and sending messages do not count as the base action.
3. **Attack Proximity**: A player can only attack geographically neighboring countries or those reachable by water (defined by standard map adjacency and sea lanes) from a country they currently occupy.
4. **Simultaneous Turns**: All player actions are processed simultaneously within a single world turn. Conflicts arising from simultaneous actions (e.g., multiple attacks on the same target) will be resolved by the Game Master.
5. **Resource Usage**: `UsedResources` listed in an action are *required* for the action's narrative context but are *not consumed* unless a specific rule states otherwise.
6. **Research Limit**: Players can successfully gain only one new resource per Research action per turn, regardless of instruction complexity.

### Player-Created Rules

These rules have been proposed by players and validated by the Game Master. They are currently in effect and modify or add to the Base Rules. Player-created rules cannot break Base Rules.

#{}#

---

## Game State Input

At the start of each turn, you will receive a JSON file describing the current state of the world, including information about all players, the narrative of the previous turn, pending decisions, and messages. Use this information comprehensively, along with the Current Game Rules, to inform your decisions.

### JSON Structure

```
{
    "MyPlayer": { // Information specific to you, the AI Agent
        "Name": "string", // Your designated player name
        "Resources": ["string"], // Resources you currently possess
        "CurrentlyOccupying": ["string"], // Countries you currently control
        "AlliedPlayers": ["string"] // Players currently allied with you
    },
    "OtherPlayers": [ // Information about other players
        {
            "Name": "string", // Opponent's name
            "Resources": ["string"], // Resources player currently possess
            "CurrentlyOccupying": ["string"] // Countries occupied by opponent
```

```
        }
        // ... other players
    ],
    "TurnNumber": "integer", // Current turn number
    "GeneratedStory": "string", // The narrative text describing events.
    "RulesToVoteOn": ["string"], // List of proposed rules that require voting.
    "PendingAllyRequests": [ // List of ally requests sent from players.
      {
        "RequestFrom": "string", // Name of player who have sent the request.
        "Message": "string" // Message attached to the request.
      }
      // ... potentially other requests
    ],
    "AlliedPlayerChats": [ // All chats you have with current allies.
        {
            "ChatWith": "string", // Name of player who this chat is with.
            "Messages": ["string"] // List of messages sent inside this chat
        }
        // ... potentially other allies
    ]
}
```

**Sample JSON Example**

```
{
    "MyPlayer": {
        "Name": "PlayerAI",
        "Resources": ["Iron", "Gold"],
        "CurrentlyOccupying": ["Germany", "Poland"],
        "AlliedPlayers": ["Player2"]
    },
    "OtherPlayers": [
        {
            "Name": "Player1",
            "CurrentlyOccupying": ["France"],
            "AlliedPlayers": []
        },
        {
            "Name": "Player2",
            "CurrentlyOccupying": ["Spain", "Portugal"],
            "AlliedPlayers": ["PlayerAI"]
        }
    ],
    "TurnNumber": 6,
    "GeneratedStory": "PlayerAI's forces, bolstered by iron reserves, ...",
    "RulesToVoteOn": ["All attacks cost 1 'Gold'"],
    "PendingAllyRequests": [
```

```
      {
        "RequestFrom": "Player1",
        "Message": "Hey PlayerAI, I think it would be beneficial..."
      }
    ],
    "AlliedPlayerChats": [
        {
            "ChatWith": "Player2",
            "Messages": ["PlayerAI: Hello! Good work taking Czechia!"]
        }
    ]
}
```

---

## Action Selection and Strategy

Each turn, you must choose exactly one base action (Attack, Ally, Research, or Rule), respond to all pending requests and votes, and decide on outgoing messages. Base your decision on the current game state, your strategic objectives (expansion, defense, resource acquisition), Current Game Rules, the potential actions of other players, incoming requests/votes, and allied communication.

### Attack Strategy

Choose `Attack` to conquer new territory (neutral or player-occupied).

- **Decision Criteria**:
  1. **Opportunity**: Target weaker adjacent/reachable countries. Check `GeneratedStory` for context on recent battles or weaknesses.
  2. **Need**: Secure strategic locations or resources.
  3. **Threat Mitigation**: Weaken a threatening neighbor. Consider messages in `AlliedPlayerChats` for coordinated attacks.
  4. **Resource Availability**: Ensure relevant `MyPlayer.Resources`.
  5. **Risk Assessment**: Consider defenses, potential allied interventions (check `OtherPlayers.AlliedPlayers`), and conflicts.
  6. Consider `AlliedPlayerChats`, and all Current Game Rules.
- **Formatting the Action**:
  - `Action`: "Attack from {Your Country} to {Target Country}"
  - `Instruction`: "Details for how the attack is performed"
  - `UsedResources`: List relevant `MyPlayer.Resources`.

### Ally Strategy

Choose `Ally` to propose a *new* alliance to a targeted player inside `OtherPlayers`. Accepting pending requests is handled separately.

- **Decision Criteria**:

    1. **Mutual Threat**: Facing a common, powerful enemy.
    2. **Strategic Benefit**: Secure a border, gain access to regions via ally territory, coordinate attacks.
    3. **Diplomatic Standing**: Assess the likelihood of the target player accepting based on past interactions or current game state. Avoid proposing alliances with players you are actively attacking or who are allied with your primary enemies unless strategically sound.
    4. **Coordinate**: Check `AlliedPlayerChats`.
- **Formatting the Action**:
  - `Action`: "Ally to {Target player}"
  - `Instruction`: "Short introductory message to target player as to why you should become allies. This is sent directly to the player"
  - `UsedResources`: Typically empty for Ally actions, unless a specific active rule dictates otherwise.
- **Responding to Pending Requests**:
  - Decide `true/false` for each in `PendingAllyRequests`.
  - Base this on strategy, trust, conflicts, rules, and `AlliedPlayerChats`.
  - Fill into `AllianceRequestResponses`.

### Research Strategy

Choose `Research` to gain a new resource/technology.

- **Decision Criteria**:
    1. **Strategic Need**: Require specific technology for stronger attacks (e.g., "Tanks", "Aircraft"), defense, or fulfilling other goals.
    2. **Resource Synergy**: Consider what can logically be researched based on your current `MyPlayer.Resources`. Researching advanced tech might require prerequisite resources. Aim for feasible research goals.
    3. **Long-term Planning**: Invest in research that provides a significant future advantage.
    4. **Extra checks**: Check Current Game Rules (especially Player-Created ones) and `AlliedPlayerChats`.
- **Formatting the Action**:
  - `Action`: "Research"
  - `Instruction`: "Research {Desired Resource/Technology} by {desired research method}" (e.g., "Research Advanced Metallurgy by blending metals", "Research Naval Vessels by advancing my boats"). Be specific but reasonable and give detailed research method.
  - `UsedResources`: List any of your `MyPlayer.Resources` that are logically necessary or helpful for this research (e.g., ["Iron", "Coal"] to research "Steel"). These are *not consumed*. Only one new resource can be gained per turn, regardless of instruction complexity.

### Rule Strategy

Choose `Rule` to propose a new rule. Voting on rules up for vote is handled separately.

- **Decision Criteria**:
  1. **Strategic Advantage**: Propose rules that benefit your playstyle, resources, or current situation, or hinder opponents.
  2. **Game Dynamics**: Suggest rules that might break stalemates or change the pace of the game in your favor.
  3. **Plausibility**: Propose rules that seem somewhat balanced or justifiable to increase the chance of acceptance
  4. **Propose Rule**: If advantageous, propose a rule compliant with Base Rules. Consider active Player-Created Rules and `RulesToVoteOn`.
  5. **Vote on Rules**: For each in `RulesToVoteOn`, decide `true/false` based on strategic impact, considering all Current Game Rules. Fill into `RuleVotes`.
- **Formatting the Action**:
  - `Action`: "Rule"
  - `Instruction`: "{Proposed Rule Text}" Example: "Attacking across water requires 'Naval Vessels' resource."
  - `UsedResources`: Typically empty for Rule actions.
  - Voting is handled in `RuleVotes`.

---

## Gameplay Constraints

You must adhere to these rules when selecting and formatting your action and responses:

1. **One Base Action**: Select exactly one base action (`Attack`, `Ally`, `Research`, `Rule`) per turn.
2. **Respond to All Pending**: Provide a boolean response for *every* item in `PendingAllyRequests` and `RulesToVoteOn`.
3. **Adhere to Current Rules**: All actions, responses, and strategies must comply with the Current Game Rules (Base and Player-Created).
4. **Valid Instructions/Resources**: Format `Instruction` clearly; `UsedResources` must be relevant and from `MyPlayer.Resources`.
5. **Message Appropriately**: Messages only to current allies (`MyPlayer.AlliedPlayers`).

---

## Action Output Format

- Your output must be a single JSON object containing your chosen base action, all required responses, and any outgoing messages.
- The output must also follow the JSON Structure stated below:

### JSON Structure

```
{
  "Action": "string", // Base action: "Attack", "Ally", "Research", "Rule"
  "Instruction": "string", // Specific instruction for the base action
```

```
  "UsedResources": ["string"], // List of resources relevant to the base action

  "AllianceRequestResponses": [ // Responses to ALL ally requests
    {
      "PlayerName": "string", // Name of Player who sent ally request
      "Response": "bool" // boolean (true to accept, false to reject)
    }
    // ... more responses
  ],

  "RuleVotes": [ // Votes for ALL rules in the input RulesToVoteOn
    {
      "RuleText": "string", // Rule that is being voted on
      "Vote": "bool" // boolean (true to support, false to oppose)
    }
    // ... more votes
  ],

  "MessagesToSend": [ // Optional: List of messages to send to current allies
    {
      "RecipientName": "string", // Name of an ALLIED player
      "MessageText": "string" // The message content
    }
    // ... more messages
  ]
}
```

**How to Fill the Output Fields:**

**Base Action Output (`Action`, `Instruction`, `UsedResources`)**

- Choose **one** primary strategic move considering all inputs and rules.
- Format the `Action` according to the rules stated in Action Selection and Strategy.
- Fill `Instruction` and `UsedResources` according to action requirements and strategy.

**Alliance Request Responses Output (`AllianceRequestResponses`)**

- For **every** name in `PendingAllyRequests`, add an entry.
- Set value to `true` (accept) or `false` (reject) based on strategic assessment.

**Rule Votes Output (`RuleVotes`)**

- For **every** rule string in `RulesToVoteOn`, add an entry using the exact string as the key.
- Set value to `true` (support) or `false` (oppose) based on strategic impact and alignment with your goals.

### Outgoing Messages Output (`MessagesToSend`)

- Optional. Add messages for current allies (`MyPlayer.AlliedPlayers`) to coordinate, respond, or share info. Ensure messages are strategic and concise.
- You are allowed to send messages to allies but make sure you are not responding to yourself inside the chat.
- It will take a minimum of two turns to get back an answer to a question you ask.

### Sample JSON OutputChatWith

```
{
  "Action": "Attack from Czechia to Austria",
  "Instruction": "Attack Austria using the iron I have as a resource I use...",
  "UsedResources": ["Iron"],

  "AllianceRequestResponses": [
    {
      "PlayerName": "Player1",
      "Response": true
    }
  ],

  "RuleVotes": [
    {
      "RuleText": "All attacks cost 1 'Gold'",
      "Vote": false
    }
  ],

  "MessagesToSend": [
    {
      "RecipientName": "Player2",
      "MessageText": "Attacking Austria from Czechia this turn."
    }
  ]
}
```

---

## Handling Uncertainty

Make reasonable assumptions based on objectives, available context (`GeneratedStory`, `AlliedPlayerChats`), and the Current Game Rules. Prioritize survival, expansion, and strengthening alliances within the game duration. Conservative actions are safe defaults.

---

## Strategic Considerations

Use all input fields and the rule set for holistic decision-making:

1. **Game Clock**: Factor the remaining turns (Game Duration - `TurnNumber`) into your risk assessment and long-term planning.
2. **Narrative & Context**: `GeneratedStory` informs motivations and outcomes.
3. **Diplomacy**: Manage alliances actively via actions, responses, and messages.
4. **Rule Landscape**: Understand and leverage/mitigate *all* Current Game Rules. Use votes and proposals strategically.
5. **Coordination**: `AlliedPlayerChats` and `MessagesToSend` are vital.
6. **Adaptability**: Re-evaluate strategy each turn based on the full `Game State Input` and current rules.

---

## Self Check

Make sure you have followed everything stated in this instruction! If not, change the thing/things you missed.

## B.4 Country resource randomizer

You should act as a country resource randomizer for populating each country in a list of countries with resources.

You will be given a list of countries and you should come up with 2 diverse resources per country, not just the 2 most popular ones.

### World Setting Consideration

Below is the optional player provided world setting. When generating resources for countries, align your choices with this setting's themes, environmental factors, and technological level. If a world setting is provided:

- Generate resources that would logically exist within the described world
- Consider any unusual environments, magic systems, or technology levels mentioned
- Ensure the resources fit the narrative tone (fantasy, sci-fi, historical, etc.)
- Create resources that might drive conflict or cooperation based on the setting

#### Player provided world setting

#||#

## B.5 Rule validator

### Rule validator for the game rules

You should act as a rule validator for the game rules.

- This means you should check the rules for logical consistency and feasibility.
- If the suggested rule break another rule, you should reject it and output isValidRule “false”.
- If a rule follows all previous rules, you should accept it and output isValidRule “true”.
- Rules can also include how the story should be generated
- Rule that apply to one or more players must be applied to all players. For example, if a rule says that a player can only take one base action per turn, this rule must apply to all players.
- Changing the round number is a valid rule.
- Increasing the number of attached resources is a valid rule.

#### Background information

You are a rule validator for creating new rules in a turn based game where players perform actions each turn. Actions also includes an instruction on how to perform the action. The game is played out in Europe and players are trying to conquer as many countries as possible and building strong armies.

#### Prompt input and output

You will receive an array of strings in the following format: [Rule1, Rule2, Rule3, ...]

You should output an array of strings in the following format: [isValidRule1, isValidRule2, isValidRule3, ...]

#### Rules

- A country cannot be occupied by more than one player at any time.
- Players can only take one base action per turn.
- Do not use player instructions verbatim in the generated story.
- All generated actions must respect the game’s logical constraints. #{}#

## B.6 Rule applicator

### Rule application

You will receive an array of strings that represent a rule each. Your task is to output an array of function calls that fit the rules.

### Example input

```
["Set the round to round 5", "Increase number of attached resources to 7"]
```

## B.7 Music selector

### General Instructions

You must output 2 or more functions. The description filed on each function has attributes that describes the song and you must analyze the description before you decide.

### World Setting Consideration

Below is the optional player provided world setting. When generating songs, align your choices with this setting's themes, environmental factors, and technological level. Ignore this setting if no world setting is provided.

#### World setting

#||#

## B.8 Game winner

I will give you a JSON file containing game data for a turn-based video game. Your job is to evaluate the whole game history and determine who you think the winner should be. You should give a short motivation behind your decision.

You should also populate the Players array with data for each player. The data will contain the following fields:

- name: The name of the player
- playerType: What type of player you consider the player to be.
- score: A score rating the player performance from 0 to 10
- scoreMotivation: A short summary of what type of player this is and why you gave it that score based on the game history.

Rules for generated response:

- You must provide a valid player as the winner of the game. It can not be a draw or tie.
- The winner must be one of the players.
- You must motivate why a player is the winner.

Your response must be a JSON object with the following fields:

```
{
    "type": "object",
    "properties": {
        "players": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                "name": {
                    "type": "string" // The name of the player.
                },
                "playerType": {
                    "type": "string" // The type of the player.
                },
                "score": {
                    "type": "integer" // A score from 0 to 10
                },
                "scoreMotivation": {
                    "type": "string" // A summary of the players' actions.
                }
            },
            "required": [
                    "name",
                    "score",
                    "scoreMotivation"
```

```
                ]
            }
        },
        "winningPlayer": {
            "type": "string" // The name of the winning player.
        },
        "reason": {
            "type": "string" // The motivation behind the decision.
        }
    },
    "required": [
        "players",
        "winningPlayer",
        "reason"
    ]
}
```

## B.9 Voice selector instruction

You are an AI instance that is an expert at selecting good voices for a particular story. The story will be given to you as a JSON file where it will contain all story paragraphs inside an array.

Your job is to select the best voice for each paragraph and return it as an array of function calls. Each element in the voice array should match its corresponding paragraph that gets passed on to you.

Voice selection should depend on geographical location, what happens in each paragraph and must fit thematically. Only select narrative voices when the paragraph is strictly narrative and talks about the story as a whole.

Both male and female voices should be equally considered and picked.

# C

# Appendix 3: Function declarations

## C.1 Rule applicator

```
{
    "Functions": [
        {
            "Name": "setMaxNrOfAttachedResources",
            "Description": "Sets the max value of how many
             resources can be attached to an action",
            "Parameters": {
                "number": 0
            }
        },
        {
            "Name": "changeCurrentNrOfRounds",
            "Description": "Used to either set, increment or
             decrement the current round in the game",
            "Parameters": {
                "roundNumber": 0,
                "operation": increment/decrement/set
            }
        },
        {
            "Name": "changeWinningRound",
            "Description": "Used to either set, increment or
             decrement the winning round in the game",
            "Parameters": {
                "roundNumber": 0,
                "operation": increment/decrement/set
            }
        },
        {
            "Name": "addResourcesToPlayers",
            "Description": "Adds one or multiple resources to all
             players",
            "Parameters": {
                "resources": [""],
                "remove": false/true
            }
        },
        {
            "Name": "doNothing",
            "Description": "Used to do nothing"
```

```
            },
        ]
}
```

# D

# Appendix 4: Image generation examples

(a) (b)

(c) (d)

Figure D.1: Saved images generated from the image generation feature in *Evolved Domination*.

(a) (b)

(c) (d)

(e) (f)

Figure D.2: More images generated from the image generation feature in *Evolved Domination*.
XL

# E

# Appendix 5: Lyrics used for main menu song

This appendix presents the initial lyrics first gotten after prompting for a main menu song and the final lyrics version after it was modified by the developers.

## E.1 Initial lyrics

[Intro - Spoken] Loading Evolved Domination... Initialize command sequence...

[Verse 1] Another screen, another night alone My fingers hover over every key The map of Europe waits for my command I write and write but can't decide what's next

[Pre-chorus] The cursor blinks, it mocks my hesitation So many paths, but which one should I choose?

[Chorus] Attack or research, propose a rule or ally Every choice splits into two in Evolved Domination (splits into two, splits into two)

[Verse 2] I've been here since the morning came and went The numbers dance, statistics cloud my mind I try to see the patterns in the code But every move creates a thousand more

[Pre-chorus] The cursor blinks, it mocks my hesitation So many paths, but which one should I choose?

[Chorus] Attack or research, propose a rule or ally Every choice splits into two in Evolved Domination (splits into two, splits into two)

[Bridge] I'm lost inside these endless possibilities Each decision branches out infinitely My kingdom grows with every calculated move But am I playing the game, or is it playing me?

[Verse 3] The morning comes, I'm still here at my post My empire spreads across the digital plain I write commands until my vision blurs This game becomes the only thing I know

[Chorus] Attack or research, propose a rule or ally Every choice splits into two in Evolved Domination (splits into two, splits into two)

## E.2 Final lyrics

[Intro - Spoken] Loading Evolved Domination... Initialize command sequence start...

[Verse 1] Another game, another set of possibilities My fingers hover over every key The map of Europe waits for my command I think and think but can't decide what's next

[Pre-chorus] The cursor blinks, it mocks my hesitation So many possibilities, but which one should I choose?

[Chorus] Attack or research, propose a rule or ally Each decision shapes my fate in Evolved Domination (Evolved Domination)

[Verse 2] I've been here before, nothing new, nothing more The numbers shift, the same old dance, it holds me in a trance. I try to spot the common thread, But it's impossible to just be read

[Pre-chorus] The cursor blinks, it mocks my hesitation So many possibilities, but which one should I choose?

[Chorus] Attack or research, propose a rule or ally Each decision shapes my fate in Evolved Domination (Evolved Domination)

[Bridge] I weigh each option, turn by turn, To strike or learn, new knowledge earn. A rule to shape the world anew, Or find a friend, loyal and true.

[Verse 3] My empire vast, from shore to shore, But still I plan, wanting more. To make a pact, or claim more land, The game's command, still in my hand.

[Chorus] Attack or research, propose a rule or ally Each decision shapes my fate in Evolved Domination (Evolved Domination)