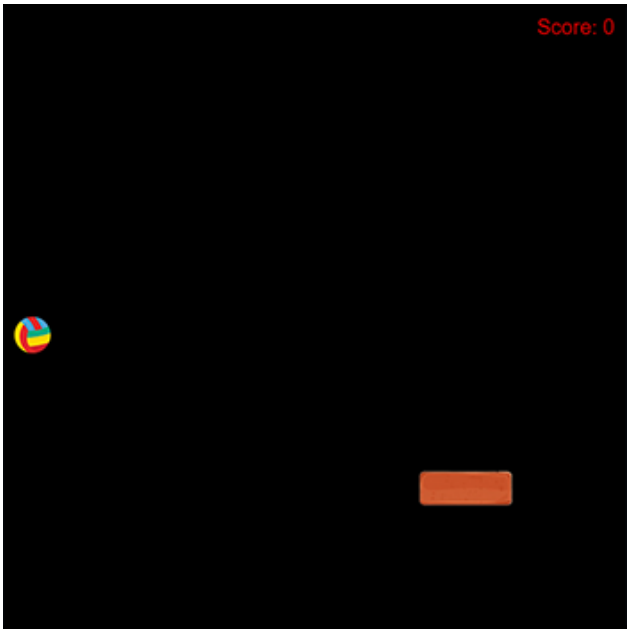
CHALMERS
UNIVERSITY OF TECHNOLOGY



Argama: Generating games from plain English

Developing a user friendly and pedagogical web application that uses a controlled natural language to interpret a game description and translate it into code

Bachelor thesis

MARTIN HAGMAR, SARA KITZING, STINA MARKAN, OLIVER OTTERLIND, JACOB ROHDIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Bachelor's thesis 2019

Argama: Generating games from plain English

Developing a user friendly and pedagogical web application that uses a controlled natural language to interpret a game description and translate it into code

Martin Hagmar, Sara Kitzing, Stina Markan, Oliver Otterlind, Jacob Rohdin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Argama: Generating games from plain English
Developing a user friendly and pedagogical web application that uses a controlled natural language to interpret a game description and translate it into code
Martin Hagmar, Sara Kitzing, Stina Markan, Oliver Otterlind, Jacob Rohdin

Supervisor: Krasimir Angelov, Department of Computer Science and Engineering
Examiner: Magnus Myreen, Department of Computer Science and Engineering

Bachelor's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Pong game created on the Argama website.

Typeset in LaTeX, template made by David Frisk
Gothenburg, Sweden 2019

Argama: Generating games from plain English
Developing a user friendly and pedagogical web application that uses a controlled natural language to interpret a game description and translate it into code
Martin Hagmar, Sara Kitzing, Stina Markan, Oliver Otterlind, Jacob Rohdin
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The purpose of the thesis was to develop a web application for students where they can create a game by writing a game plan in plain English. The goals were to use modern coding practices and principles in the development, and the application should contain similar functionality as GameChangineer while being more user-friendly.

GameChangineer was evaluated in order to identify possible improvements in regard to user experience. A controlled natural language was defined in order to interpret game plans. A server was implemented to handle the generation of Processing code that was used to display the game on the web application.

The outcome of the project was a web application using mostly modern practices and principles, with a functionality close to the GameChangineer website, but less complex. The final user test showed that the application's user experience improved compared to GameChangineer.

Keywords: Controlled natural language, Grammatical Framework, Processing, GameChangineer, game generation, user experience.

Sammandrag

Rapportens syfte var att utveckla en webbapplikation för studenter där ett spel kan skapas utfrån en spelbeskrivning skriven på engelska. Målet var att applikationen skall använda modern kodpraxis och programmeringsprinciper, innehålla liknande funktionalitet som GameChangineer samt vara mer användarvänlig.

GameChangineer analyserades för att identifiera sätt att förbättra användarvänligheten. Ett kontrollerat språk definierades för att kunna tolka användarens spelbeskrivning. En server implementerades för att hantera genereringen av Processingkod och för att generera spelet i webbapplikationen.

Projektets resultat var en webbapplikation med mestadels moderna kodpraxis och principer med funktionalitet som nära efterliknar GameChangineers, dock inte lika komplex. Enligt de sista användartesterna visade sig användarvänligheten vara förbättrad jämfört med GameChangineer.

Nyckelord: kontrollerat språk, Grammatical Framework, Processing, GameChangineer, spelgenerering, användarvänlighet.

Acknowledgements

We would like to thank our supervisor Krasimir Angelov for his assistance and insights during the project. We would also like to thank the testers for giving valuable insights when developing Argama.

Finally, we would like to thank the team behind GameChangineer for contributing with the original application which gave inspiration and valuable insights during the project.

Martin Hagmar, Sara Kitzing, Stina Markan, Oliver Otterlind, Jacob Rohdin, 2019

Contents

List of Figures

Glossary

Argama Arcade Game Maker, the name of the application.

CNL Controlled natural language.

CSS Cascading Style Sheets.

GF Grammatical Framework.

HTML Hypertext Markup Language.

HTTPS Hypertext Transfer Protocol Secure.

HTTP Hypertext Transfer Protocol.

IP Internet Protocol.

RGL Resource Grammar Library.

TCP Transmission Control Protocol.

1
Introduction

Computer games are loved and consumed all over the world. They provide endless hours of fun challenges, and they can create new friendships between people living on different continents. They can teach morals and stir up emotions, provide a new understanding for, and an improved ability to solve problems. A love for computer games is something that can be shared no matter your age, race, or gender, your political views or religion.

The path leading from loving computer games to being able to create one is normally not a straight-forward one. While programming is a powerful tool, there might be a high learning curve due to the many new concepts and rules that define it. Before even starting, the choice of which programming language to learn needs to be made, and there are several hundred to choose from. Then, beginning with learning the syntax of the language, take the many small steps toward understanding how to go from nothing to a fully functioning game.

During recent years the question of how to encourage students to learn programming has been debated, and last year the Swedish government decided that programming will be added to the syllabus as a mandatory part of the education even in the early grades [2]. The world is becoming increasingly digitalized and if every student learns to write code it might mean that they will be equipped with more tools to tackle the problems of the future than what earlier generations had.

The aim of this project is to create a web application that can be used as an aid when teaching students programming concepts. The web application will provide them with the opportunity to code their own game in plain English by using proper grammar and adhering to the given limitations.

In this introductory chapter the background of the project, which is a platform called GameChangineer, will be reviewed. A more detailed purpose, as well as the goals, and limitations of the project will also be described further.

1.1 GameChangineer

Michael S. Hsiao, a professor at Virginia Tech, has developed a learning platform called *GameChangineer*. The platform is meant to teach children programming concepts along with proper sentence structure, and has been used in several different

American schools since its launch in 2016, and has received very positive feedback [3].

On the GameChangineer website, the content is split into four frames, one frame at the top, two in the middle, and one at the bottom as depicted in Figure 1.1. The user is presented with an input form in the right frame. The form asks the user for a game title, the users name, and a game plan. The game plan is written in a text box with auto-complete functionality. If the user begins typing and it matches any of the available game characters a text box will be shown and the user can click directly on the character name to insert it into the game plan.

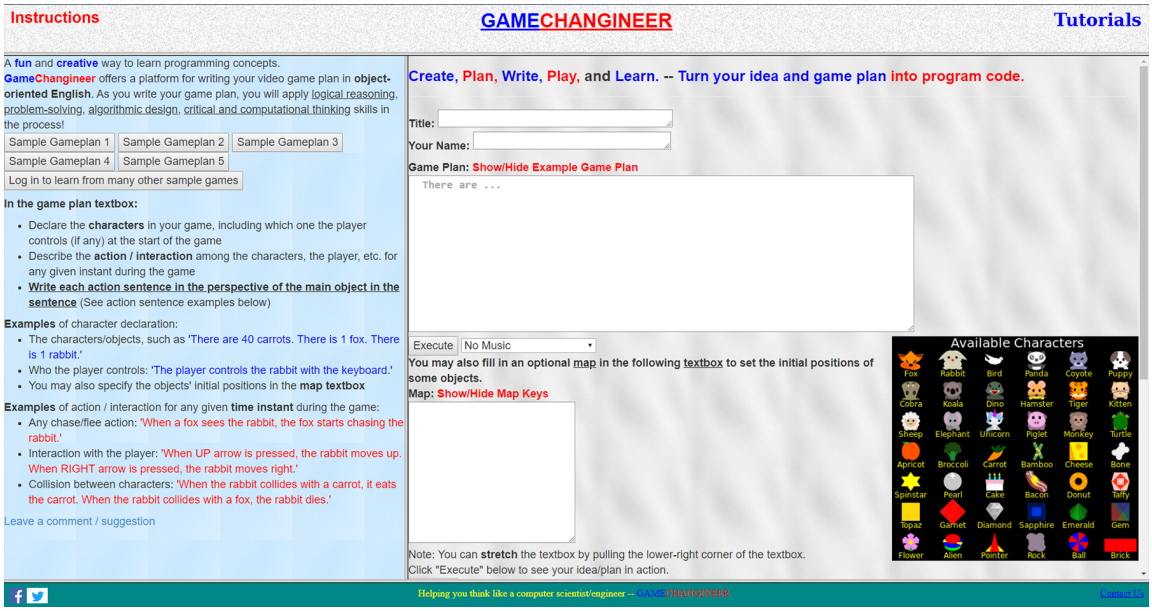


Figure 1.1: The GameChangineer platform. Print screen from [1]. Created with permission.

When the user has finished writing the game plan, the user presses the execute button to generate their game. The game, along with information about any errors the user made in the game plan, will then be presented in the left frame of the website. If there are severe errors made in the game plan, the user will only be shown the errors and no game will be shown at all.

There are two major observations about GameChangineer that are relevant for this project. The first observation is the visual design of the platform. The use of modern design patterns is limited, and the user is presented with an extensive amount of information right from the start, but also in particular when the user has made errors in the game plan. The second observation is the complexity of the game engine. For example, the user can write their game plan in many different ways since the use of synonyms is supported, and it is possible to create animations for different physic phenomenons, such as gravity or Newton's Laws of Motion.

1.2 Purpose

The purpose of this project is to develop a web application for students where they can create a game by writing a game plan in plain English. The application should use modern coding practices and principles, and contain similar functionality as GameChangineer while being more user-friendly.

1.3 Goals

The main goal of this project is to develop a working prototype of the web application described in the previous section, that a user will be able to visit and use to create a game, while having fun and learning along the way. The web application should have a pleasant user experience, and minimize the possibility of feeling stuck or frustrated due to bad error messages, and either insufficient or excessive information.

The goals that the project aspires to accomplish are the following:

- Developing the application by using modern practices and principles.
- Designing for user experience.
- Making the content pedagogical.
- Generating easily understandable error messages.
- Successfully interpret game descriptions, including synonyms and errors.

This project has no affiliation with the GameChangineer platform and there is no intention from the project group to infringe on any intellectual property rights related to it. However, the project still has a secondary goal which is to provide ideas and inspiration to its creator Michael S. Hsiao and his team — hopefully, the work described in this report can benefit the future development of GameChangineer.

1.4 Limitations

In order to clarify the scope and focus area of the project, certain limitations are required. The web application will only be implemented for English. The language was chosen since it is a well-known language in most parts of the world, and known by all the project group members. However, the expectation is that well-written, modular code will lead to that the future addition of new languages will be relatively simple to implement.

In addition, the web application will be developed to be used with web browsers on computers, and not on mobile phones. Developing applications with compatibility for several platforms adds complexity, and in this case also difficulties in regards to implementing movement in the games. For example, using the arrow keys on the keyboard to move in a game has no given substitute on a mobile phone. Therefore, the attention will be focused on computer users although a mobile adaption might

be worthwhile to continue exploring in the future.

This project will not be deployed on a live server, thus no work will be done to make sure the back end of the web application can handle requests from a large number of users.

1.5 Outline

Following this introductory chapter, the thesis will continue on with a presentation of the theory behind the technologies and techniques that are relevant for this project. This includes defining a controlled natural language, developing a client-server application, designing for user experience, as well as the educational aspect.

Next, Chapter 3 Method will present how the web application was created, and how it got the name Argama. A detailed explanation of the resulting controlled natural language, server, and website that makes up the application will be presented in Chapter 4 Results. A discussion of the obtained results and user tests, and possible improvements of both the application and the process will be reviewed in Chapter 5 Discussion. Finally, the thesis will end with Chapter 6 Conclusion, which connects the purpose defined in Section 1.2 with the results of the project.

2
Theory

In this chapter, the techniques and technologies used to carry out the project will be described further. The chapter is split into five sections to reflect on the different parts of the project. The first three sections inform about the technologies used to create the web application, which includes a controlled natural language, a server, and a website. Next follows a section about how to design a website with good user experience in mind. Lastly, a collection of educational guidelines are presented.

2.1 Defining a controlled natural language

A grammar is a set of rules for how certain types of words in a language follow each other. An example for the English language is *he is* which is correct, but *he are* or *he am* are incorrect, which is decided by the English grammar. In a similar way a controlled natural language (CNL) can be defined, which also decides how certain words follow each other [4].

A CNL is a middle ground between a formal programming language and a natural language [5]. The CNL can be designed to use the same grammatical structure as a natural language while avoiding ambiguity since a CNL is derived from a natural language. This enables more people to be able to learn programming principles considering that the users are using a familiar natural language rather than a formal programming language [4].

2.1.1 Grammatical Framework and RGL

Grammatical Framework (GF) is a functional programming language that can be used to create a CNL [6]. In GF the CNL is divided into an abstract and a concrete syntax [7, p. 11] which will be explained in further detail in Section 2.1.2.

The Resource Grammar Library (RGL) is the standard library for GF. The RGL covers the basic syntax of many languages, among the languages covered are English, French, German, Spanish, and Swedish [8]. The tool aids in creating a concrete implementation with the supported languages based on the abstract grammar definitions.

By using the RGL, the implementation of the concrete syntax of a new CNL will be sped up, since the RGL defines the grammatically correct combinations of the

words. A developer can use the functions from the RGL without worrying about inflections and word agreement which automatically leads to correct word forms and sentences [9, Lesson 4], [10].

2.1.2 Creating a CNL using Grammatical Framework

A CNL written in GF is divided into two or more GF-files that contains the grammatical rules. There is one abstract file and the rest are concrete syntax files, one for each language [9, Lesson 1]. To clarify the difference between these files, a detailed explanation and a code example will be presented below. The example has two files where the language of the concrete file is English. RGL will also be explained in more detail since it is a significant part of the examples of concrete syntax.

The abstract syntax contains `startcat`, `cat` and `fun`. The `startcat` defines the default starting category when generating and parsing with the CNL [9, Lesson 1]. In the example below the default category is *Creation*. The keyword `cat` introduces the category declarations, in other words, the types that can be worked with to create text segments. `fun` contains the function declarations that can be used to convert between the different categories [9, Lesson 1].

```
abstract ExampleAbs = Numeral ** {
    flags startcat = Creation;

    cat
        Creation; Object;

    fun
        ThereIs:  Object ->           Creation;
        ThereAre: Digits -> Object -> Creation;

        Ball, Taco : Object;
}
```

In the code example above, the function *ThereAre* pairs an *Object* and multiple *Digits* to create the category Creation. The Digits are part of a library that is imported with the `Numeral **` statement [7]. The Object is generated from one of the declared functions that create the category Object. In the example, there are two functions creating objects; *Ball* and *Taco*.

The next part of the CNL is the concrete syntax, which starts with a `lincat`, as seen in the example below. The `lincat` introduces the linearization type definitions that states what type of word or text segment the category belongs to. The categories are the same categories as in the abstract syntax [9, Lesson 1].

The code example shown below uses two different types; *NP*, and *CN*. NP is a noun phrase, which means a subject, or an object. CN is a common noun, which means that it does not have a determiner. All of these types are a part of RGL and

have multiple different constructors that can create them [8].

The keyword `lin` is the linearization of definitions, here the functions defined in the abstract syntax are converted with the help of constructors in RGL into actual words, sentences, and text segments [9, Lesson 1]. The function name is written first and needs to be the same as in the abstract syntax. After the function name follows the input parameters if they exist [9, Lesson 1], which are used by the constructors. These constructors, *mkN* and *mkCN*, create a noun and a common noun [8].

```
concrete ExampleCon of ExampleAbs =
 NumeralEng ** open SyntaxEng, ParadigmsEng in {
 lincat
    Creation = NP;
    Object = CN;

 lin
   ThereIs  object        = mkUtt(mkCl(mkNP a_Det object));
   ThereAre number object = mkUtt(mkCl(mkNP (mkDet (mkCard number)) object);

   Ball = mkCN (mkN "ball");
   Taco = mkCN (mkN "taco" "tacos");
 }
```

When looking at the linearization of Ball in the example above, it can be seen that the word ball is written as a string. The constructor mkN accepts a string as an input and turns it into a noun. The noun is then accepted as a parameter by the mkCN, which makes it a common noun. Depending on the input parameters that are used and how they are used, different word forms, sentences, and text segments can be created [8]. As mentioned in Section 2.1.1, RGL aids in keeping the correct inflections, which is something that makes the concrete syntax easier to write and handle.

The linearizations of the two functions ThereIs and ThereAre from the example above contains a few more RGL functions. Those RGL functions are the following; *mkUtt*, *mkCl*, *a_Det*, *mkDet*, and *mkCard*. The first, mkUtt, represent sentences, followed by mkCl that are declarative clauses, which simplified means statements. Next are a_Det and mkDet which both are determiners, these decide if a word is in definite form or not. Last is the constructor mkCard which works with cardinal numbers [8].

The CNL generated from the abstract and concrete syntax examples shown above can understand sentences such as *There is a ball*, and *There are 99 tacos*. In the sentences, ball can be alternated with taco, tacos can be alternated with balls, and the number 99 can be replaced with any number between 2 and 99.

2.2 Developing a website application

An application is divided into a front end, and a back end. While the server is the back end, a website can be used as the front end. It is also commonly known as the client-side since it gives the user a possibility to connect to the server and interact with the functionality within. The logic and calculations are handled on the server, and the website determines the look, feel, and visual design of the application.

There are three main programming languages used on the web today; Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript [11]. They each have their own responsibilities, but what connects them is that they work in the browser, presenting information to the user. HTML makes up the structure of the website, such as text, tables and images. By using CSS, all the elements that are created in HTML can be styled, for example, all the text can be set to a certain font, size, and color. JavaScript is responsible for all dynamic content, it can alter how the website looks in real time with no need to reload pages. JavaScript has become a true staple in web development [12], and there exists many JavaScript frameworks and libraries that make life easier for developers.

Details about the additional libraries and frameworks that can be used to create a website and that are relevant for this project will be described further in the following sections.

2.2.1 The Bootstrap framework

Bootstrap is a framework for applications that use HTML, CSS and JavaScript [13]. It consists of a collection of tools and code fragments that may be used in order to build a responsive website [13], [14], [15], meaning that the site will look and work as the developer has intended on different screen sizes and devices [16]. Bootstrap is a time saver for developers by offering reusable code that already has a basic, responsive design instead of each developer having to write their own unique code.

Bootstrap comes with wide documentation, making it easier for developers to know how and when to use its features [13], [14]. Some examples of tools that Bootstrap offers are; a responsive grid layout system, components such as buttons, drop-downs and forms, as well as utilities such as shadows, colors, position and text transformation [15], [17].

2.2.2 Processing and the Processing.js library

Processing is an open source programming language intended for visual representations, such as software prototyping and data visualization. It was designed to be used by beginners but has over time also become a tool used by professionals in their work [18]. The syntax of Processing can be described as a simplified version of Java. This can be seen in the code snippet below, which represents an example of how to create a ball in Processing.

```
Sprite ball;

void setup() {
    ball = new Sprite(loadImage("ball.png"));
}
void draw() {
    ball.update();
}
```

Here, *Sprite* is a custom class created to visualize objects. The *setup* method is executed once at the start of the Processing program, and here it initiates the ball object and loads its image. The *draw* method is run continuously as long as the program runs. In this case, it runs the balls update method, which is a custom method that, for example, can update the object's coordinates if the object is meant to move.

Furthermore, Processing is also used for programming simple games, and there exists several websites that list collections of arcade type games created with Processing [19], [20]. The GameChangineer platform's source code shows that it uses Processing to render the user-created game, which provides a proof-of-concept that it works for these kinds of applications.

Since HTML in itself is a simple language meant for static text and images, the development of a game that the user can control demands that a library is used. Processing.js is a JavaScript library that makes it possible to run Processing code within Javascript tags on a website. It needs to be imported in every HTML file where it will be used, and it has the support for both inserting the Processing code directly in the HTML, and importing it from a Processing file.

2.3 Server: the back end of an application

A server, also known as the back end, is used to host a service or an application on the internet, making it possible for users to access it. The server should supply the clients with the necessary files, and it should also be able to receive data from the user and handle it correctly.

2.3.1 Hypertext Transfer Protocol

To receive requests from, and to send data to the client, a server can use the Hypertext Transfer Protocol (HTTP), which runs on top of the Transmission Control Protocol/Internet Protocol (TCP/IP). HTTP works by sending and receiving *requests* and *responses* between two different programs. If a client wants to visit a website, the client's web browser sends a request to the server. A request message has several different types of request methods that are used to communicate to the server what it wants, two of these methods are GET and POST. GET and POST are both used for requesting objects, however, POST is also used for sending data to

the server, such as form answers. The server can send a response back to the client, which includes one of several status codes depending on if the server can send back the requested file or not. Common status codes are *200 OK* and *404 Not found* [21].

There are several different versions of HTTP, and two of the main versions are 1.1, and 2.0. One of the differences between 1.1 and 2.0 is that 2.0 has the ability to initiate push procedures from the server [22]. For example, in HTTP 1.1, the client would request the index.html file, and the server would respond back with the requested file. When the client receives and compiles the index.html file it detects that it needs additional files that are imported in index.html. The client will then proceed to request the imported files that are specified in the index.html file from the server. However, in HTTP 2.0, the client can request index.html and the server can then proceed to push the imported files to the client.

2.3.2 Java Sun HTTP Server API

Instead of writing the whole code for implementing HTTP communication on a server from scratch, the possibility to use an Application Programming Interface (API) exists, for example, the Java Sun HTTP Server [23]. The API allows a developer to set up the server with a specified IP address, port, executor, and handlers. By using a library, the process will probably be easier and save them time in comparison to the developer doing everything by themself.

2.4 Designing for user experience

User experience can be seen as a comprehensive view of the product [24]. It is the experience people get when using a product, interacting with the interface rather than the code which the functionality is based on [25]. When designing for user experience one of the most important components to consider is the user's expectations, emotions, and needs [24]. There are different tools and techniques to be used when designing for user experience, two of them are the use of design patterns, and user testing, which are described further in this chapter.

2.4.1 Design patterns and user flow

Design patterns are guides and solutions to common design problems. They bring out the familiar parts in order to make the user recognize the design and how to interact with the interface, while still being general enough for the designer to be able to be creative [26, pp. xi-xv].

A major goal of any product design is to create a good *flow*, which means that the design focuses on what the user needs to do and delivers a solution for it with as much ease as possible [26, pp. 14-15]. For example, when writing a text document a user knows that by pressing the CTRL and S keys together, or by clicking on the floppy disk, the document will be saved. It is something they have learned from other programs that utilize this design pattern. By not using the common

key-binding, or by forcing the user to search through the menus to find the option to save the document, the user's flow can be disrupted. Saving the document is no longer a straight-forward operation, but instead something the user has to figure out how to do, which might lead to lesser user experience.

2.4.2 User testing

Bringing in users to test an existing application can provide valuable information about its user-friendliness, as well as what could be improved. It also gives feedback on whether the product and its functionality are relevant for the intended users or not. For example, if designing an application for the elderly, testing the application on teenagers that have better vision and that usually are more experienced with computers and other devices, it might not give any relevant insight about if the application works well for its target market.

When planning for user tests, two of the things that should be defined is the age group that will be tested, as well as the task that are to be carried out [27]. The task is abstractly explaining what the user is doing in the system during the test, for example, if the user is testing a banking application one task could be to transfer money between two accounts. The test should not specify the exact steps that the user is expected to take to complete the task, since it might affect how the user interprets, and proceeds using the application.

2.5 Guidelines for educational games and texts

The demand for including programming in education increases. The UK government wants children at the age of 11 to 14 to be able to use two or more programming languages, and the Japanese government is making programming a compulsory subject in primary school [28]. A popular approach for including programming in the curriculum is through games [29], which increases the importance of knowing how to combine programming and games with education.

2.5.1 Teaching children to program by using games

There are five success factors to consider when games are used to teach how to program: Motivation, Integration and Involvement, Interaction and Feedback, Adaption to the Audience, and Integration of Education Content into Gameplay [29]. All of which are described in more detail in the following section.

Motivation is important when learning, not only because students enjoy education more, but also because it lowers the dropout rate [29], [30]. Students generally get motivated just by using games in education [30], [31], [32] which results in a higher learning effect [29]. Motivation can be divided into two parts; intrinsic and extrinsic motivation, where intrinsic motivation, which means that the student gets motivated by themselves and the tendency of wanting to explore and learn [33], is the relevant one for education while using games [29].

There are four factors to consider in order to achieve intrinsic motivation, all of which are based on individual perception [29]. The game should challenge the experience as well as the fantasy and imagination of the student. Furthermore, it should arouse curiosity for the unknown but still give the student control over the situation [29], [34].

The second success factor, Integration and Involvement, covers the key factor for combining games and education — students being involved in the curriculum [29]. Including games in the education can make students become more actively involved in their education since the player of a game has to be an active participant when playing [29], [35]. Games also have a tendency to make us wanting to participate to a higher extent [36]. With games in the curriculum, all students get to participate in their own game in front of them instead of receiving information from the same teacher [29].

Giving feedback, and enhancing the interaction with students and the educational content is a fundamental aspect for teaching in general [29], hence the importance of Interaction and Feedback when teaching programming through games as well. There needs to be communication between the student and the game in order for the game to be successful. This can be achieved through the possibility for the player to change the code once feedback is given. The feedback encourages to learn by mistakes [29], [37], which together with the possibility to interact with the game as well as the trial and error method as the way of deducing the logic and rules of the game are not only factors for success when playing a game, but also the critical factors during the learning process [29].

Adaption to the Audience is the fourth success factor. Games are in general adapted to the audience [29], however, educational games need to be both pedagogical and enjoyable from a game design perspective, which usually is not combined when an educational game is developed. This makes the adaption of educational games tricky [38], [39]. The adaptiveness games provide can reduce frustration from the higher end of the class. Since the game adapts to every student's own learning ability and needs, it reduces the need to listen to "stupid questions". Instead, every student can work at their own pace, and because of it be more focused than they otherwise would have been [29].

In order to make a game educational, there needs to be educational content introduced in the game, thus the fifth and last success factor. The game needs to have a balance between the gameplay and education, where they harmoniously blend together [39]. This means both including teachers during the design implementation, and for the functionality of the game to be integrated into the curriculum [29], making sure the gameplay and learning aspects are not completely separated. This helps the student feel the progress of both the game and the subject when playing, as well as give them more challenges when increasing the difficulty of the game [29], [40].

2.5.2 Writing educational texts

It is important for the texts to be pedagogical when writing for educational purposes. Otherwise, the student will not understand the content, which defeats the goal of an educational text. There are guidelines to follow for increasing the readability of texts, as well as special guidelines for text on the web [41].

Some of the common guidelines are:

- The most important should be written first, for example as a summary [41], [42], [43].
- A new paragraph should be marked with a blank line [41].
- Use short sentences rather than long [41], [44].
- The contrast between the background and the text should be high. This can be checked through a Contrast Ratio Analyzer [41].
- The font face Verdana should be used, it is created especially for reading on screens [41], [43].
- Unnecessary and decorative graphics should be removed [41].
- Important words and sentences could be underlined as long as it is done meaningfully [41].
- The layout should be airy [41].

Students with disabilities, such as reading disabilities, need to be taken into consideration as well. Using alt-tags for all images is one way to help [41]. An alt-tag is a description of what the image shows. It is created for people with visual impairments who have either turned of image display or use speech synthesis [41]. Offering a version of the text in a structured way, for example as a numbered list, may also help [41]. Making the text readable when the user increases the text size is another way to help people with reading disabilities [41], [43].

3
Method

The development of the application was split into three parts; a controlled natural language, a server that can use the CNL to translate a user's game plan into Processing code, and a website for users to interact with. The methods used for creating these parts will be described in the following sections.

3.1 Creation of the controlled natural language

To define the grammar, and thereby the CNL used in the application, the functionality of GameChangineer was analyzed. The first step was therefore to set limitations on what functionality to implement, with focus on the time frame for the project. Then, a decision on what needed to be included in the CNL was made, dependent on the functionality the user needed in order to create games. For example, the user should be able to add objects, and the objects should be able to interact with each other as well as being able to move. It should also be possible to win or lose a game.

After deciding what to include in the CNL, GF was used in order to define it. The sentences that needed to be implemented were going to be dependent on each other, for example, by an object needing to exist before an action could be connected to it, which gave a natural order in when the sentences needed to be created.

First, the sentences for creating objects were implemented. Next, the focus was on implementing sentences that allowed objects to perform actions, followed by adding the possibility for objects to interact with each other. After that, actions as a result of object interaction, such as two objects bouncing off each other when they collide, were implemented.

Finally, player control, as well as win and lose conditions were implemented. The player controls made it possible for the user to assign key presses in order for an object to perform the desired action on user command. For the winning and losing conditions, instantaneous win or loss, as well as incrementing, or decrementing the player's score was added.

3.2 Developing the server

At the start of the development, the main purpose of the server was to supply the client with the necessary files to render the website, and a simple implementation

of HTTP was created. A test handler was implemented in order to see how the handler interface worked. The test handler could serve the client that connected to the server with one file; index.html. After the test handler was implemented, the first proper handler, GET handler, was implemented. It served the client with HTML, JavaScript, and CSS files.

After this stage, the server needed to be able to process input from the client. Thus followed the implementation of the handler that could handle POST requests; POST handler. When the game plan was properly received on the server, the next step was to integrate the CNL. With GF's runtime system, a Grammar handler was implemented. The handler would take the data input that was sent from the client, split it into separate sentences, and parse it into the different categories and functions defined in the CNL.

To generate and display a game on the website, the server had to be able to generate Processing code. A compiler was developed, which generated code based on the parsed game plan. Since the code was meant to run on the website and not directly on the server, the server itself did not need a Processing library. However, the website did, and the JavaScript library Processing.js was therefore added among the website's files.

3.3 Website design process

The process of designing the website started with creating and performing a user test on the GameChangineer platform. Based on the information from this user test, design sketches were made and used as a guide when the actual design for the website was implemented. Later on in the project, a user test was also carried out on the new application. The details of each step are presented in this section.

3.3.1 User test on GameChangineer

The user test was conducted with one user at the time. Before the test, the user taking the test was given a clear task that he or she had to accomplish. In this case, it was to use the GameChangineer platform and create a simple Pong game. The observer, i.e. the person that oversees the test, was to clearly describe the task to the tester and to provide the tester with hints if they ended up completely stuck in the process and not being able to move forward on their own. However, the point was to understand how the tester used the application without any assistance, so the observer was to interfere as little as possible during the test.

3.3.2 Creation of design sketches

Prior to implementing the design on the website, several mockups were made. The mockups were created both by hand and digitally in Adobe Photoshop, which made it possible to try the design and different ideas without the need for full website functionality. The digital mockups made it easy to see how the chosen color scheme

worked together and if the colors were soft enough to not be distracting while still bringing some playfulness to the design.

A selection of which design patterns to use were taken into consideration during the creation of the mockups. The design patterns helped to solve design problems and to make sure that the interface became clear and simple for the user.

3.3.3 Implementation of the design

The first step when implementing the design was to add all the components to the views. The design mockup worked as a guide of which elements should be added to each view, however, the position as well as the look of each element, was not relevant at this stage. The implementation was done with HTML, along with some CSS to improve the positioning of the elements and reduce cluttering.

The next step was to add functionality to the elements. Buttons were connected to the right actions, the vertical menu was added, and a basic tutorial was implemented. The technology used in this step was mainly JavaScript, with some addition of HTML and CSS when needed. The Processing.js library was used to visualize the black game square.

Finally, the design of the website was tweaked to look like the mockups. For this step, CSS and Bootstrap were used in order to change the look and exact position of the elements.

3.3.4 User test on the new application

Following the full implementation of the design and functionality, a new user test was created and performed. To get a good comparison, the user test had the same task for the user to perform. Besides being tailored for the new application, the course of action was the same; the observer was to have a limited or no influence on the tester. The result of both of the user tests was used in order to evaluate the design of the application and develop the design further.

3.4 Naming the application

When naming the application, options were brainstormed among the group members, where the focus was to have some kind of association to the functionality of the application. Words that could describe the product, or part of the functionality of it where found, as well as their synonyms. Examples of such words were: *Develop, Make, Create, Design, Generate, Arcade game, Game, Programming, Build* and *Play*. The words were used when acronyms were created and as shortenings which were added together in order to come up with a reasonable name.

4
Results

In this section, the final version of the application is presented. The result of the created CNL, and server are described, as well as the functionality and design of the website. Design patterns used, and the name of the application is also presented below.

4.1 The scope of the controlled natural language

The final CNL that was constructed has the functionality to be able to understand the necessary sentences to create games. This includes being able to create objects, defining different actions that these objects can perform, defining interactions between the objects, player control, and lastly, end conditions. In the sections below these parts are described in more detail.

4.1.1 Creating objects

To be able to create objects is the first necessity for the user to have when creating a game. The CNL is designed so that sentences that are used to create objects always starts with *There*. The next word is either *is* or *are*, depending on if the user desires to create a single object or multiple of the same object. The CNL can handle sentences that create up to 99 objects of the same type. Lastly follows the name of the objects that the user wants to create. An example of two sentences and how they are parsed is presented below.

```
There is a ball.

CreateObject (ThereIs Ball)

There are 17 hegdehogs.

CreateObject (ThereAre (IIDig D_1 (IDig D_7)) Hedgehog)
```

4.1.2 Assigning actions to objects

The next thing the CNL can handle is actions related to the objects, for example, *The hedgehog wanders*. The sentences for actions always start with *The*, followed by the object performing the action.

There are several kinds of movement actions that an object can perform including, but not limited to, wander, move, and chase. There are some differences between these when it comes to sentence structure. The action "Wander" does not require any additional information, which means that *The hedgehog wanders* is a sufficient sentence. However, "Chase" requires another object to chase, for example, *The taco chases the monster*. Finally, "Move" should be followed by an adverb to describe which direction the object will move, such as *The hedgehog moves right*. The end of the sentence may also be switched from right to any direction, or if the objects moves fast or slow. An example of how this will be parsed is shown in the example below.

```
The hedgehog moves right.

GiveAction (ActionToStandardAction (Move (MakeAgentSg Hedgehog) Right))
```

The actions can further be separated into three different categories; normal actions, event actions, and resulting actions. Normal actions can be used like the examples above, independently in a single statement. Event actions and resulting actions are a bit more complicated and are used together to create conditional actions which will be explained in further detail in the following sections.

4.1.3 Conditional, event, and resulting actions

Conditional actions make use of actions to act as triggers on specific events. These conditional actions are split into two sections, event actions, and resulting actions. In the example below a conditional action, and how it is parsed, is presented.

```
When the ball collides with the hedgehog, the ball disappears.

ConnectActions (MakeCondActionWhen (Collide (MakeAgentSg Ball)
(MakeObstacle (MakeAgentSg Hedgehog))) (Disappear (MakeAgentSg Ball)))
```

In the shown example, *When the ball collides with the hedgehog* is the event action and is acting as a trigger for the resulting action *the ball disappears.* Conditional action sentences always start with either *when*, or *if*, which is then followed by one event action and is finished with a resulting action.

4.1.4 Key events for user control

Having a CNL that is able to interpret conditional actions improves the possibilities of making a more complex game. Conditional actions also provides possibilities for making user control possible in the program, by using sentences such as the one presented in the example below. The keys that can be used for user control are the arrow keys and the w, a, s, and d keys. These keys were chosen since these are the most commonly used in video games, providing a sense of familiarity to the users.

```
When the up arrow key is pressed, the monster moves up.

ConnectActions (MakeCondActionWhen UpArrowKeyPressed
(ActionToResultAction (Move (MakeAgentSg Monster) Up)))
```

4.1.5 End conditions and score

The final part of the CNL handles sentences for winning and losing a game. There is one action for incrementing the score, another that decrements the score, one action for winning, and one for losing the game. All of these are of the type resulting actions. However, there exists one more action that is called score reaches, which is an event action. It can be used together with the action for winning the game, an example and how it is parsed is presented below.

```
When the score reaches 9, the game is won.

ConnectActions (MakeCondActionWhen (ScoreReaches (IDig D_9)) Win)
```

4.2 The design and functionality of the server

The final server is successful in integrating most parts of the CNL. It also supplies the client with all the necessary files, and is able to receive the game plan from the client. It can use the parsed game plan from the CNL to generate code that then is sent to the browser to render the game. In the sections below, the server, as well as the procedure of generating Processing code, will be explained further.

4.2.1 Server architecture and flow of information

The server implements HTTP 1.1 with the Java Sun HTTP Server API, which means that handlers are used to respond to client requests. In HTTP 1.1 when a client requests an HTML file, it also needs to request all the necessary associated files, such as images or JavaScript files, to compile the HTML file. For the server to supply these files, two handlers are used; a GET handler that supplies text files, and a Picture handler that supplies image file formats.

The server may also receive a game plan from the client, and for this the POST handler is used. The server receives an HTTP request with an embedded URL, the URL contains the requested path. The path is then matched to a handler, for example, if the path is to an image the Picture handler is used. The full details on the flowchart of the server can be seen in Figure 4.1.

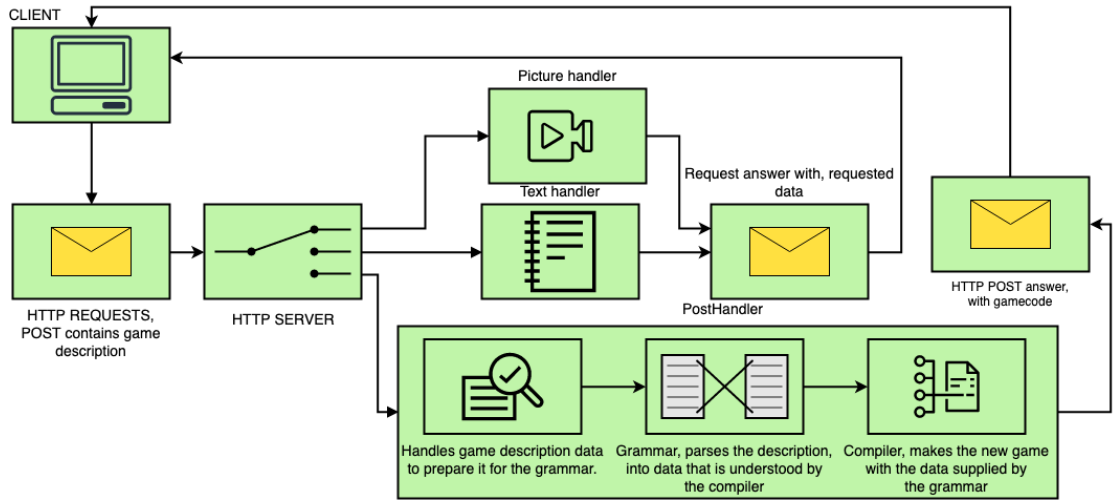


Figure 4.1: A flowchart that represents the way the server receives a request from the client, processes it, and sends back a response.

4.2.2 Generation of Processing code

When the server starts, the POST handler reads the game-view.html file contents, and finds the line numbers of three parts of the Processing script. The line numbers, as well as the game-view.html file, are saved into the local memory for inserting generated code in the right places later in the process. The first line number represents the beginning of the Processing script, and this is where global variables are defined. The second is in the setup method, which is where the global variables are initialized and edited. The final is the draw method, where global variables are updated. Below is a code snippet where three lines are marked with comments, this is were all code will be inserted after a clients game plan has been compiled into code.

```
<script type="text/processing" data-processing-target="mycanvas">

  //<!-- Start of scripttag. Do not remove -->
  boolean gameIsRunning;
  boolean redrawing;
  ...

  void setup() {
          font = loadFont("FFScala.ttf");
          frames = 30;
          frameRate(frames);
          size(600, 600);
          background(#212121);
          gameIsRunning = false;
          textBottomCenter("Click to start the game!");
          title("Argama");

  //<!-- Start of setuptag. Do not remove -->
        redraw();
```

```
 }

 void draw() {
        if (gameIsRunning) {
                background(#212121);

//<!-- Start of drawtag. Do not remove -->
        }
}
...
</script>
```

When a user presses the "Compile" button, the browser sends a POST request to the server, the server interprets that it is a request for the POST handler and forwards it accordingly. The POST handler takes the game plan and prepares it for the CNL by splitting it into separate string sentences that are converted into lower case.

When the game plan is ready for parsing, the CNL is started in the GF runtime system, which is running on the server. The game plan is then forwarded line by line through the CNL. The expressions generated for each line of data is then traversed until the actions and objects are isolated.

The Grammar handler class contains several different methods, that use the visitor pattern [45] to recursively identifying the types of expressions that the sentence consists of. For example, if the user compiles the sentence *There is a ball*, the method *on_CreateObject* will be called, which then calls *on_ThereIs*. Next, the object is extracted from the expression, and the Grammar handler will call the compilers method *createObject*, which in turn generates the concrete code from predefined strings of Processing code.

The generated code is put into one of three different lists. When all lines of the description have been parsed, and the code has been generated, the POST handler starts sending game-view.html to the client line by line. However, when the line number of global variables, setup, or draw is in line to be transmitted, one of the three lists is inserted into the transmission. This way the browser receives a game-view.html file that is modified by the addition of code that generates a game on the game canvas.

In order for the server to generate as little code as possible, classes and methods for the game characters were created in Processing beforehand and are a part of the original Processing script in game-view.html. This leads to that the compiler can create objects of predefined classes, and call the objects' methods in order to generate less code.

A detailed example of the full process, from the user's input, to finished generated code can be seen below.

```
User input:

    There is a ball.
    There are 2 monsters.

Handled by POST handler:

    there is a ball
    there are 2 monsters

Parsed by the CNL:

    CreateObject (ThereIs Ball)
    CreateObject (ThereAre (IDig D_2) Monster)

Generated code by the compiler:

    Global variables:

        Sprite ball;
        XSprite monster;

    Setup:

        ball = new Sprite(loadImage("res/ball.png");
        monster = new XSprite(loadImage("res/monster.png"), 2);

    Draw:

        ball.update();
        monster.update();

Added code in game-view.html:

    <script>

        //global variables

        Sprite ball;
        XSprite monster;

        setup(){
            ...
            ball = new Sprite(loadImage("res/ball.png");
            monster = new XSprite(loadImage("res/monster.png"), 2);
            ...
        }

        draw(){
```

```
            ...
            ball.update();
            monster.update();
            ...
        }

        ...
</script>
```

4.3 The contents of the website application

To provide an easier understanding of the final website for Argama, the results will be presented in regards to functionality, and visual design, as well as the design patterns that were chosen for the final design. The full result can be found in Appendix C.

4.3.1 Using the application

The website is written in HTML and utilizes JavaScript for extra functionality. There are two main pages for content; the index, and the game view. The game view, which is shown in Figure 4.2, is the core page of the application and this is where the user can create games. The page is designed to not feel cluttered, and to only show information when it is needed or prompted by the user.

Figure 4.2: The game view in Argama.

As default, the game view shows the game creation section, which includes an input for the game title as well as the game plan, and a button to compile the code. Right next to that section, the main navigation is found, and it is designed as an

overlaying section which is minimized in the beginning. The navigation contains much more information that the user can access by clicking on any of the colored round buttons, which will then open up an overlaying tab, as shown in Figure 4.3. The tab contains an iframe that displays an HTML page with information that is paired with the clicked button. All of the menu transitions are coded in JavaScript.

Figure 4.3: The game view with the help tab opened.

To the right of the navigation, the game container is found, and it is coded in Processing. To be able to use Processing on the website, the code is wrapped in a JavaScript tag and the Processing.js library is used. The script is loaded into a *canvas*, an HTML5 component that is used to draw graphics on a web page. When the user first enters the page, the canvas is displayed as a black square with a contrasting text displaying the title and a starting prompt. When the user writes and compiles their game plan, the server is adding the new lines of code that display the new game content to the Processing code on the HTML page.

The user has the ability to go back to the index page and its tutorials by clicking on the button in the top right corner. This button is always used as navigation between the index page and the game view page, and vice versa, making it easy for the user to go back and forth if needed.

If the user starts a tutorial on the index page, the game view will open, as well as the info tab in the menu where the text consists of the first step of the chosen tutorial, as shown in Figure 4.4. The user can navigate through the tutorial by using the “Next” and “Back” buttons, and see the progress by how many filled circles there are. The “Next” button changes to be called “Finish” in the last step of the tutorial. By pressing this button, the user will be sent to the game view as it looks and functions when a tutorial is not started.

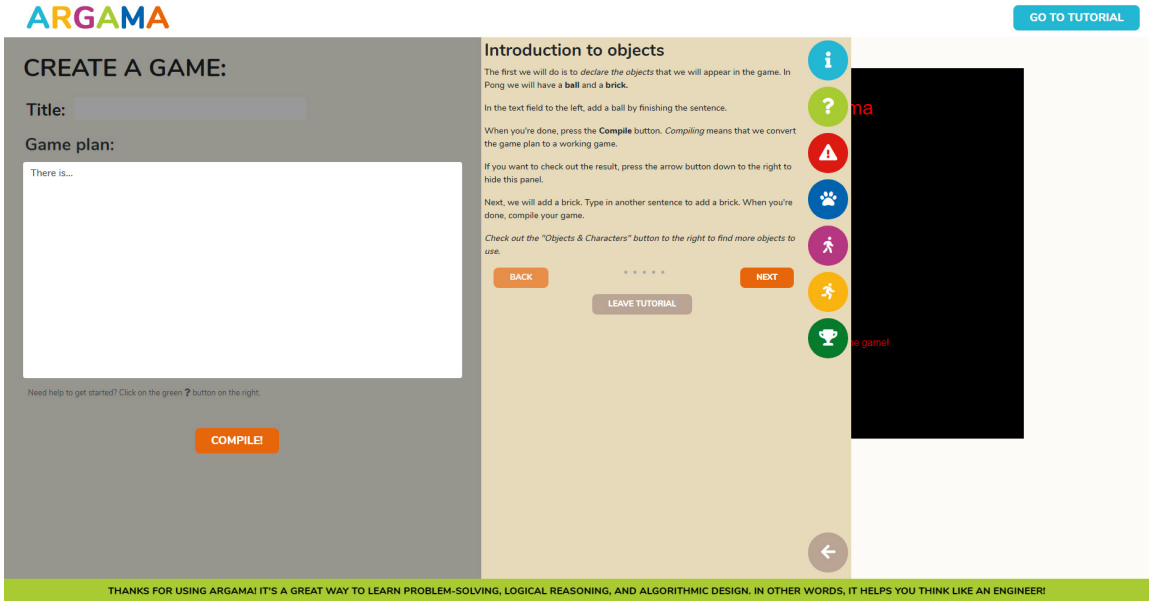


Figure 4.4: The game view with a tutorial active.

There is no restriction or check to make sure that the user writes what the tutorial expects. The user is also able to leave the tutorial whenever they want by pressing the "Leave tutorial" button. If the button is not clicked, the tutorial will stay on the same step when the user changes or closes the tab, or if the user starts to write a different game. The dark overlay, however, disappears when the tutorial is not showing. The only way to make the tutorial disappear is by pressing the "Leave tutorial" button, finish the tutorial, or navigate to the index page and then back. For each of these actions, the game view will look and function as it does when a tutorial is not started.

4.3.2 The application's visual design

The index page, as seen in Figure 4.5, serves as a clear entry point for the user, and it has two choices they can make. Either, they choose one of the tutorials that are found in the carousel that is placed in the center of the screen, or they can opt out and go straight to the game creating by pressing the "Skip tutorial" button on the top right.

The point of the index page is to catch new users and provide them with an introduction to Argama by guiding them through a tutorial of how to create a game. This gives them the opportunity to learn about all the different objects, and actions that they can use to create their game and where to find the information that they need. A user that has previous experience with the application will recognize the layout and easily find, and click on the "Skip tutorial" button to get straight to the game view.

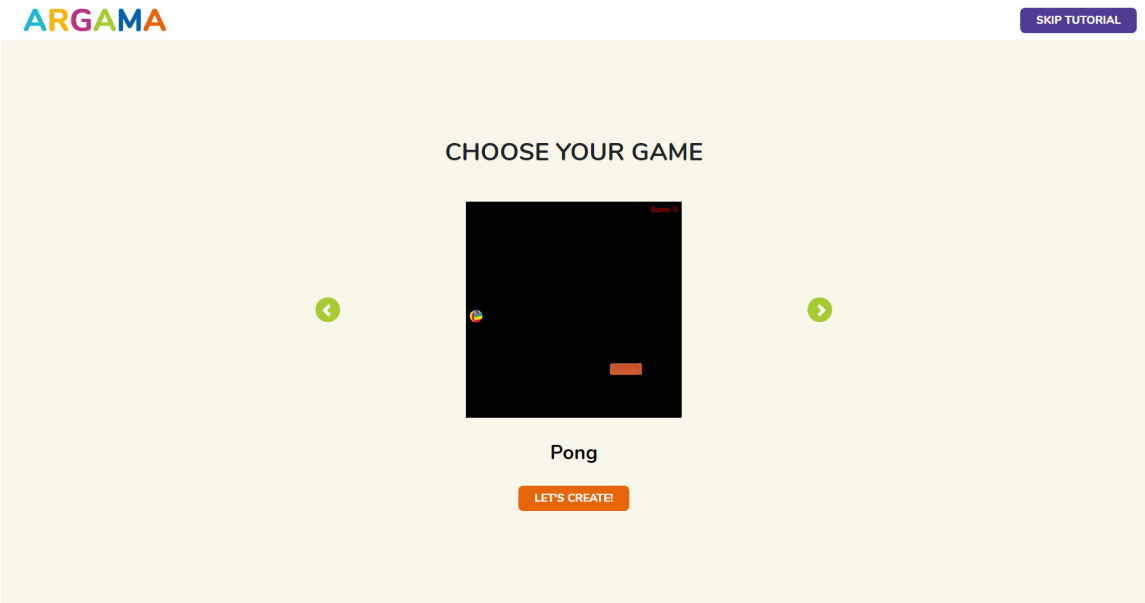


Figure 4.5: The index view in Argama.

It was important to create a design that felt welcoming and friendly to the user, hence the base colors are different hues of warm yellow-toned grey instead of the more traditional neutral grey palette. The colors of the buttons are sourced from a picture of Lego® bricks, which is fitting since Lego® could be associated with both creativity and to build, or create new things. The colors were picked specifically to be vibrant but not too bright so that they walk the fine line between drawing attention to themselves while at the same time not being distracting for the user.

The Bootstrap framework was used as a design base on the website since this meant that many components came with basic styling. To refine the design and make it look exactly like the mockup, custom CSS was created. This can, for example, be seen on the round buttons, which is not possible to have using standard HTML and Bootstrap only.

Every component on the website is using the same font, which is called Nunito. It is imported directly in the CSS from the Google Fonts directory. This way, the user does not need to have the font installed on their own computer to see it on the website. Nunito is a youthful, slightly rounded sans serif font created by Vernon Adams, and it is used in several different font weights, ranging from regular to extra-bold. Even though Section 2.5.2 states that Verdana is the best font to use, it was not chosen since the design of it is dated and much more harsh, and it would not harmonize with the overall design.

The icons on the round buttons are vector icons from Font Awesome. It is a toolkit used by designers, which allows the use of all their icons by importing their CSS file directly in the files where they are needed. By using vector icons, it is very easy to change them if needed, and their size will change automatically just like an ordinary font would.

Furthermore, the icons were chosen to represent the content in their respective tab as well as possible, for example, a simple *i* was chosen for the information tab. This was done to make the navigation intuitive without needing to use words, the user should understand what information they will be presented with when clicking on one of the buttons. This probably works best for the three first buttons (information, help, and error) since these symbols are well known. However, the buttons each have their own tool tip, which is a small floating text label, that is displayed next to the button when the user hovers the mouse cursor over it which informs them of what information they can find by clicking the button.

4.3.3 Usage of design patterns

As seen in Figure 4.3, the menu in the game view contains the two-panel selector pattern, the first panel being the buttons and the second being the information contained in each tab.

The tutorial uses the wizard pattern, guiding the user through the steps of making the desired game. This wizard does not force the user to do every step or to finish the whole tutorial in order to do something else. It is made to be a helpful guide rather than rules to follow. To see how the tutorial implements the wizard pattern, see Figure 4.4. Another pattern that is used in the tutorial is the escape hatch pattern. It can be seen by the "Leave tutorial" button, see Figure 4.4, which makes it possible for the users to leave the tutorial whenever they want.

The multi-level help pattern is implemented through giving the user the possibility to get small tips by looking through the tabs in the menu, as well as letting the user get a full walk-through tutorial on how to create some given games.

The index page uses clear entry points. The user is presented with two possible entries for how to use the application; either start a tutorial or create a game, see Figure 4.5. The entry points are task-orientated by using the labels "Skip tutorial" and "Choose your game" paired with the "Let's create" button.

The buttons for the categories in the menu are one example of where the website uses the button group pattern. They all have the same shape and size, as well as similar icons. The rest of the buttons lead to actions on the website. They also have similar looks by using the same shape and font, see Figure 4.2.

The "Compile" button shown in Appendix C.3, is a prominent done button. It is placed at the end of a visual flow, finishing the action of writing code. The button is big and labeled in order for the user to understand the action.

The list contained inside the "Objects & Characters" and the "Actions" tabs use the row striping pattern. It alternates between two different shades of beige for every row in the list, see Appendix C.6 and C.7.

The illustrated choices pattern is used for the tab buttons in the menu. Each button uses a picture in order to distinguish the buttons from one another, instead of text that describes each category, see Figure 4.2.

4.4 The name

The application's name is *Argama*, the first two letters of each of the words in the sentence "Arcade Game Maker".

5
Discussion

In this chapter, the results from the user tests, as well as the CNL, server, and website will be discussed. A discussion about possible improvements, including, but not limited to, error messages, design, and educational guidelines, will follow.

5.1 Results from the user tests

User testing could be done early in the design process since there already existed a platform which the new application was to be based on. Furthermore, this meant that future design choices could be based on the conclusions drawn from the results of the user test. The importance of choosing relevant persons as testers is mentioned in Section 2.4.2, and despite the small portion of project time dedicated to testing, an effort was made to make sure that the testers chosen could provide relevant test data.

The first user test was done on the GameChangineer platform. In order to produce concrete results that could be of use, the task was kept simple so that the same task could be done on Argama. The user was to find the page where they could create a game, and write a Pong game, which included a ball that bounced, and a brick.

After compiling the answers from the test, the main complaints that emerged were that GameChangineer was perceived to be messy and that it was hard to find information. That lead to users having difficulty with beginning to write their game plan. The full details of the first user test can be found in Appendix A.

One of the goals of this project was to create a user-friendly design, so when creating the design for Argama it was prioritized to improve the elements that were complained about in the GameChangineer test. For starters, there are several things that could contribute to the platform feeling messy. For example, the contrasting color scheme, the amount of information that is presented to the user, no use of a clear entry point, or even that it uses a non traditional flow of information, since the game plan is written on the right part of the screen and the game is presented on the left part instead of the more traditional left-to-right.

To create a design that felt less cluttered, most of the information was hidden inside the vertical menu, so that the user can easily find it if they need it. The use of white space, which means areas of the design that are empty, increased to make

the user have a more pleasant experience since the eye would have an easier time focusing on the important parts of the application. The chosen color scheme has less contrast and therefore might not distract the user. The design was also made to fit the left-to-right flow of information.

Besides hiding unnecessary information, a tutorial was also incorporated into the design, to help the user get started with writing their game plan. Other helpful elements were also added, such as a green help button, and a short text below the game plan text area that prompts the user to click on the help button if they have trouble getting started. All of the texts in the application were written using the educational guidelines mentioned in Section 2.5.2.

The second user test was done on Argama, to evaluate if the design did have improvements in comparison with GameChangineer. The full details of the user test can be found in Appendix D. Several of the testers mentions that the design "looks good and is clear" when asked about their first impression. However, there are some things mentioned that could be improved upon to further make the design more intuitive, including a clear description on the index page that tells the user what the website is about, changing out the title "Choose your game" to be more informative, and adding information about the latest additions in the CNL to the information tabs.

In order to have a reasonable comparison to GameChangineer the testers were not allowed to use the tutorial to create their game, still, they seem to have had an easier time getting started. The vertical menu was being used, mostly by the tester thoroughly reading the help tab, but also the other tabs to a lesser extent. The testers did not get stuck, as in not knowing what to do next, in the process of writing their game. However, they did spend time reading the information provided by clicking the tabs.

A notable difference is how long the tests took to complete. The mean time elapsed for the GameChangineer tests was 37 minutes, whereas the same test for Argama had a mean time of 25 minutes. During the GameChangineer test, one user gave up and their time is therefore not included in the result. All of the Argama testers completed the test.

To conclude, based on the user test it seems that Argama has completed the goal regarding user-friendly design. It could be argued that the way it is designed is helping the user too much since at least for a Pong game the user can find the full solution in the different tabs. However, Pong is a simple game, and by giving the user extra help in the beginning, they might have more fun and therefore increase their motivation, which might lead to them trying to create more games that are more challenging than Pong.

5.2 Evaluation of the CNL

A challenge of starting to learn programming is that new programmers are not used to writing with the constraints of a syntax. That can make it hard to learn programming principles, since the programmer might want to try different expressions in order to comply with the syntax, and this could be difficult when the language is not familiar to them. However, if a person is trying to program with a natural language there could be a problem of ambiguity, and that the computer will not be able to interpret the written text correctly. Thus, a CNL can be used to enable the user to use a familiar syntax that is constrained, which means that it will not have the problem of ambiguity, to learn programming principles.

The final version of the CNL only has support for using numbers when creating several of the same object. In the English language it is, however, grammatically correct to use letters to write out the numbers one to twelve. Thus, to have a CNL that functions the same way as the English language, this should be implemented. The CNL is also limited to only be able to create a maximum of 99 of the same object, since there is no reason to go above this limit because the screen is already overflowed by 99 objects.

As mentioned in Section 1.3, Argama should "successfully interpret game descriptions, including synonyms and errors", however, in the current version there is no support for writing synonyms. There exists only one instance of being able to write a type of sentence in two ways, and that is conditional actions, where it is possible to write "If" instead of "When". Due to the time constraint of the project, developing a CNL that could interpret some defined sentences, so that the user could write a complete game plan, was prioritized over a CNL with only a few sentences that had synonyms.

The CNL has some limitations when it comes to the actions. In order to make sense, certain actions can not be written in certain ways, while others can be written but will not do anything relevant. Two examples of this can be seen below.

```
When the ball moves right, the ball moves left.

The ball collides with the border.
```

The first action is accepted by the CNL, although it does not provide any good functionality. Worth noting is that this action can provide good functionality when the combined sentences are reasonable. The second sentence has been limited in the form of an event action. This means that the CNL does not accept it as an normal action as it is written currently, and it does not work as a resulting action either, since it would not provide any functionality as these types of actions.

5.3 Limitations in the server's functionality

The final version of the server support all the necessary requests from the client. The server was successful in integrating the website and the CNL. Even though the server was successful in its role in the project, some of the technologies are not state of the art, for example, HTTP 1.1 is used, which is still sufficient for the project but not as modern as HTTP 2.0.

The functionality connected to generating error messages was unfortunately not designed, nor implemented on the server. However, it was the next step of development, and the server already handles some of the errors internally without the browser receiving any information about it.

Furthermore, some sentences were implemented in the CNL but the server can not generate the Processing code associated with them. For example, incrementing and decrementing the score, as well as making objects disappear or explode.

5.4 Analysis of the website components

The finished website turned out as intended for some parts of the website, while other parts turned out better, and some parts worse. Below follows a deeper analysis of how the functionality, design, and educational aspects turned out in comparison with what was intended.

5.4.1 Functionality

The tutorial works as expected, the same tutorial as the active picture on the index page will open and the possibility to jump back and forth between the steps of the tutorial works as intended. The possibility to keep the progress when closing and changing the tabs also works well. However, a drawback is that there is no straightforward way to add new tutorials. Tutorials needs to be added directly in the code, which makes it hard for others than a developer to do so.

The dark overlay that is visible over irrelevant parts during the tutorial did not really work out as expected. It was difficult to make the overlay work as intended, see Appendix B.4, leading it to only work for the left part of the view where the user enters inputs for the title and game plan, see Appendix C.4. Because of this, the functionality of the overlay loses part of its purpose since it is supposed to fade out the whole page except for the relevant elements at the beginning of the tutorial, and as the tutorial continues change the overlay depending on what the text in the tutorial references. As a result of the limitation to where the overlay can be added, the effect of the overlay will not contribute to the user experience as much as expected.

5.4.2 Design

The design turned out very similar to the mockup, but with a few additions since new functionality was added along the way. One difference is the amount of buttons in the game view menu. There where originally four buttons in the mock-up (see Appendix B.2), which later on was changed to seven (Appendix B.4), excluding the back button. The reason for the addition was the need for more space since more information needed to be added. It seems more reasonable to add extra buttons instead of adding more text in the existing tabs, especially to make sure the user still can find the information easily instead of having to search through long texts in the original tabs. However, one drawback is that the interface gets a bit more cluttered, making it harder to get an overview of the page.

The difficulties with the functionality of the dark overlay lead to the fact that the position of the elements that might have an overlay had to be absolute. Because of this, they will not be responsive when the size of the browser window changes. Which means that the elements will be stacked onto one another when the window is too small, making it close to impossible to use the page. Since responsive websites are common today, this might lead to lesser user experience.

5.4.3 Pedagogy

The success factor Motivation is mostly fulfilled by the game. It challenges the fantasy, since only the imagination decides how the game is supposed to work. It also arouses curiosity for the unknown by giving the possibility to learn a bit of programming, giving the student the motivation to learn more and to create new, more advanced games later on. The student is in control of the games outcome as well, as long as the syntax is correct.

The challenge of the experience part of the Motivational success factor, is a bit ambiguous since it does challenge the student but only to some extent. When the student already has created a game, the next experiences might not be as challenging as when doing the first one.

The student may change what is written in their game plan whenever they like. However, the only type of feedback provided is by compiling their game plan and seeing the result, leading to partly fulfilling the Interaction and Feedback factor.

The game only adapts to the students success factor to some extent. The student has the possibility to make use of tutorials when new to the concept, and afterward make the game more advanced, which is a kind of adaption to the success factor. However, the student has to make the call by itself, which might lead to the game not being challenging enough. There is no push from the application for the student to go further, leading to them being able to only create games on the same level of difficulty to make it easier for themselves. There is also nothing that gives any information about how hard an implementation will be, which might lead to the student trying to create a game that is too difficult for their level of knowl-

edge, resulting in them not improving but instead to losing hope or simply giving up.

It is hard to say if the educational aspect and gameplay are in balance throughout the game since it is not created or evaluated by either game developers or teachers. Because of this, the Integration of the Educational Content into Gameplay factor might not be fully realized. It will most probably be at a decent stage though, since the game is developed by using the functionality of an existing product, which at least will fulfill the educational purpose.

The texts in the application follow several of the guidelines of educational texts that were mentioned in the theory chapter. First of all, every new paragraph is marked by a blank line, and unnecessary graphics are removed in order to make it easier to follow. The texts also begin with a short summary and marks meaningful words. When checking the contrast ratio, the contrast between the foreground and background color fulfills the requirements. However, the length of the sentences are not worked with enough, which leads to some of the sentences being longer than they optimally should be. The reason for this is the lack of time.

5.5 Possible improvements for Argama

As in most projects there is room for improvements, which could be realized if time or other restrictions had not been an issue. Possible improvements for each part of Argama, as well as the method used, are discussed in the following sections.

5.5.1 Planning and work distribution

If the project were to be done again, the main difference when it comes to the process would be the assessment of how much work each part actually is. Generating the visual game on the website was taken into consideration from the beginning. However, the Processing part of it and especially the work it needed, was not properly assessed. The workload of implementing the server was also assessed to be less than it actually ended up being. This resulted in an uneven distribution of manpower, leaving one member of the group to take care of the whole implementation of both the server and Processing.

If more focus would have been put into the Processing part, for example by having one member being responsible for it, and started working with it from the beginning, the Processing code would have been ready for implementation earlier in the process. It might have lead to more functionality being added to the game, since there would have been more group members working in parallel on different tasks, rather than multiple group members working on similar things.

By dedicating one group member to work single-handedly with the Processing implementation, the work distribution would have been more even. It would be an advantage for the group by keeping the work progressing evenly between all tasks,

rather than having one member work when the rest are putting on the finishing touches on their respective areas.

5.5.2 Expanding the controlled natural language

The CNL that was constructed has the minimum required functionality to create games. An improvement would be to have a larger and more complex CNL that can handle synonyms for both words and different sentences. As of now, the user is limited to use the exact same sentence structure as described in the result section. If more synonyms could be used, there would be more possibilities to create the same meaning and thus the user could use their own vocabulary.

The number of actions, and the tools available for use when creating games is still lacking compared to GameChangineer. One such tool that Argama could need is being able to decide where on the game canvas objects are created. This would enable the user to create games with a more specific layout.

Another improvement that can be done is the implementation of different languages, enabling students that do not speak English fluently to learn basic programming principles in their native language. GF has the ability to create a CNL with support for several different languages, however, only English was implemented due to the limitations stated in Section 1.4. Since the implementation of the game elements is based on the abstract syntax's types, a new concrete syntax can be created with relative ease for each new language that is to be added.

In the current version of the CNL there is only support for a user to create a game that only has a single player. The player can win or lose against the computer, but there is no real way for the player to compete against a different player. Thus, an improvement could be to implement a way to have two separate scores to allow for a player one, and a player two to compete.

5.5.3 Modernizing the server

The server could be improved by using more modern technologies and improving the functionality support. For example, HyperText Transfer Protocol Secure (HTTPS), which is a commonly used protocol today, since security has become an increasingly important part on the web, but the server was implemented without support for it. This was done simply because the application does not send or receive any sensitive data. However, if there had been more time available, the server would be upgraded to support HTTPS, especially since it has several benefits, and is required for some browser and mobile device functionalities.

5.5.4 Generating and presenting error messages

After a user compiles their code on GameChangineer, they are presented with status messages for every line of code they have written. For example, if the user has written "There is a ball near the top", they will be presented with two status messages,

There is one ball near the top – Understood with 94% certainty, and *NOTE: the ball is placed randomly in the top half initially.* Naturally, when the game plan becomes a bit longer, the status messages quickly add up. If the user makes an error, the error message is presented together with the status messages, which means that it might easily be missed by the user if they quickly scroll by the mass of text that these messages create.

The error message in itself is a bit complicated, especially when considering that the platform is designed for 9-13 year old children. For example, the error message shown in Figure 5.1 is – *ERROR: Possibly an unclear antecedent in the conditional expression, since simply moving in some direction or stopping is not a valid conditional antecedent. Instead, use 'see', 'collide', 'reach', 'is [attribute]', etc.* In this case, the word antecedent might not be a word that every student knows, making it harder for them to understand and fix the error.

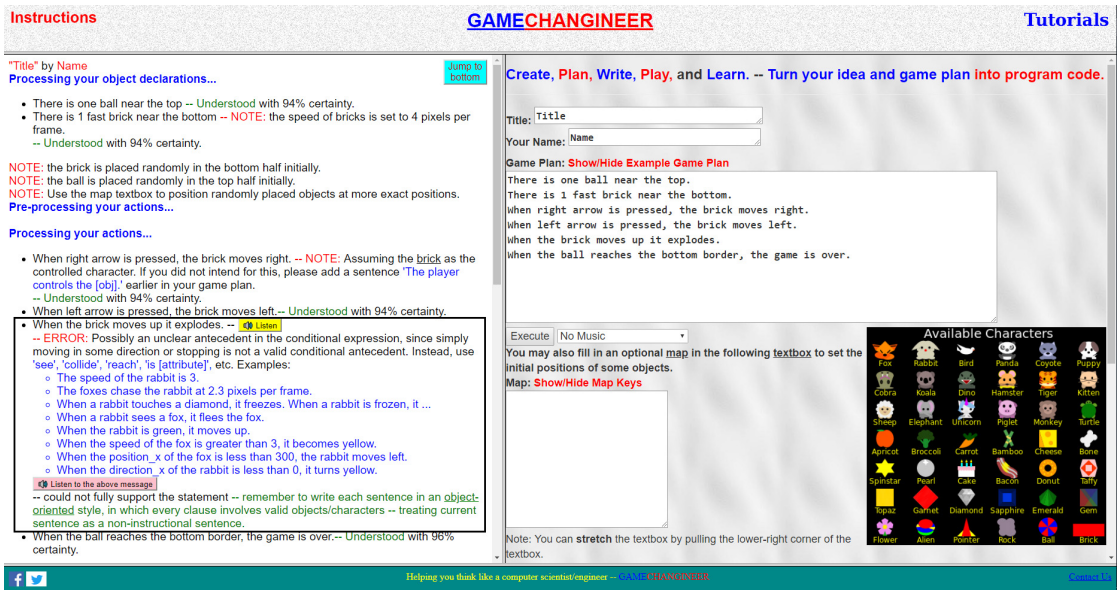


Figure 5.1: How the GameChangineer platform looks after compiling the written game plan, with the error message marked by the black square. Print screen from [1]. Created with permission.

When developing Argama, a goal was to provide the user with clear, pedagogical error messages that would be easy to understand, and that could teach the student a lesson in programming. This would be done by marking the errors directly in the game plan, and using the server to generate informative error messages that are presented in the error tab. The red button that opens up the tab should have a badge that displays the number of errors that are currently in game plan. An example of how an error message could look can be seen in Appendix B.3.

The error message should contain the wrong sentence, *why* it is wrong, examples of correct sentences that are relevant, and provide a link to the tab with the relevant information so that the user can find a guide if they want to. It should also connect the reason why the sentence is wrong to traditional programming. A sentence might,

for example, be a correct English sentence, but if it is not supported by the CNL it is incorrect syntax for the scope.

Unfortunately, the idea could not be implemented in code. This was mainly due to the time constraint, but also the fact that there were some limitations in how GF handles incorrect input. If the user writes a sentence that is not supported by the CNL, GF will respond with *The parser failed at token X*, where X is the position of the incorrect word. If the index is 1, which is the starting index, it can be expected that the user has written an incorrect sentence, for example, by trying to write something that is not part of the CNL. If the index is not 1 then the user probably started to write a correct sentence but did a spelling mistake or perhaps tried to use objects that does not exist. However, if the user wrote a correct sentence but did a spelling mistake on the first word, it would produce the same error as writing an incorrect sentence.

In order to generate relevant error messages the server would need extra features that could parse the sentences, and understand why they are wrong, for example by using pattern matching. Perhaps even another grammar could be created that would only be used for the sentences with errors.

What has been done, however, is to remove status messages completely. This gives Argama a less cluttered feel and thereby an improved user experience, which is one of the goals of the project. It also removes any uncertainty that the student might feel by reading the status messages, for example, if a sentence is perceived as 94% correct by GameChangineer, the student might wonder how to improve it and get stuck if there is no apparent way.

5.5.5 Refinements of the website design

A possible improvement of the design is to make it possible for the dark overlay in the tutorial to be added all over the view. In this case, the interface will be more straightforward during the tutorials, which makes it easier for the user to understand what is happening.

If the overlay could be added all over the view, the buttons in the menu could have an overlay or be lit up during the tutorial, making it easier for the user by guiding their eye to know where they can find more information about what is mentioned in the current part of the tutorial. It would also make it possible to add a more in-depth tutorial for the whole game view, for example to make a complete beginners tour of the menus, elements, and buttons in the game view.

For the design of the index, some kind of description of what the application does could be added. If a student uses Argama in class, there probably will not be a problem with the users understanding the purpose of the website. However, as soon as a user does not have the assistance someone that already knows what the website does, it gets a bit trickier to understand the purpose just by a quick look on

the index. To keep a playful touch on the index page, some kind of character could be added with a speech bubble containing a brief description of the websites purpose.

For the future, more design patterns could be implemented in order to make the user experience even better. One example of a design pattern that could be added is the auto completion pattern. The pattern could be added by giving the user suggestions on objects or actions that start with the same letters the users are writing in order to make it easier for the user to know which objects and actions that could be used. It also minimize the risk for spelling error on the words crucial for the CNL's understanding.

Another design pattern to add is the animated transition for when tabs are opened and closed, making the transition smoother. One concrete example could be the vertical menu where the transition when changing the categories could use an animation in order to make the change of content in the tabs less jumpy.

5.5.6 Additional functionality on the website

Another improvement could be the elements being better at adapting to the window size, so that they are responsive. It does not only make it easier for users with smaller screens, since the elements will not be stacked, but also helps people with visual impairments who are in the need of zooming in order to read the content. Making the elements adapt better will also help with the overall visual impression of the application. The space between each element will vary depending on screen size leading to the interface only looking as intended for the screen size it was designed on.

A possible addition to the tutorial could be the user being able to choose the difficulty, for example by deciding whether the overlay should be used or not. If the user already have completed several tutorials, their problem might not be to navigate the application, but rather to get help with which objects or actions to use, or simply to get inspiration for what game to create next. For Argama to stay interesting and challenging for recurring users, more tutorials for different levels of knowledge would have to be added. During this project only one tutorial was added due to the time constraint.

An example could be to make it possible to add new tutorials without having to be able to program. This could be done by making it possible for teachers to have accounts, where an easy interface with text boxes lets the teacher add the text, and the possibility to press on the elements to highlight or overlay for each step of the tutorial. The added tutorials could either be added permanently to the site, making it possible for whoever to try them out, or by giving the teacher a special link for the site where the added tutorials exists to make it possible for the students to find them.

Being able to save the code to use it later might also be a worthwhile feature to add. As it is currently, the user can always copy the written code and save it as text until later, for example on their own computer. Although, it might be frustrating not to

have an option presented on the website where this could be done. Just adding the possibility to save the written code as a text file on the desktop, and being able to load text files on the website would have been an easier approach to the problem.

5.5.7 Continuous work towards educational guidelines

To increase the educational factor of the application, it could be changed to become more similar to a real game, meaning adding different levels, or achievements, where the difficulty gets harder as the student progresses. This does not only motivate the student to keep on using the game but also adapts better to the student's success factor by customizing the level of difficulty better. Another way to adapt to the student's knowledge could be by marking each tutorial with a difficulty level, which gives a better understanding in which games are too difficult, or too easy at the time.

The texts in Argama could be worked on more than they have been. By re-reading them several times, asking teachers which approach to use when explaining programming, or even having students from the target group read them and give their opinion on what needs more clarification, it would be possible to improve the texts in regards to how pedagogical they are, and making sure the information that is conveyed is understood by the user.

In the current version of the application, disabilities are not considered because of the limited time for this project. This is important to acknowledge before introducing the game in schools in order to not exclude anyone. The main things to add would be alt-tags to all the images and elements in order to make it easier for people with speech synthesis. Implementing a way of enlarging the text to make it easier for students with visual impairments to read would also have to prioritized.

Adding how the game plan should look like, either after each step in the tutorial, or at the end is another improvement that could be implemented. It does not have to be displayed directly, but the user could for example press a button and the game plan could then appear. Displaying the correct game plan could help the user to understand what to do if they really do not understand how to write the game plan when reading the tutorial, or if they are stuck and do not really know where something went wrong. Giving them the possibility to find the game plan, also makes it possible to compare what the game was intended to look like, and how the user actually created it, giving them the opportunity to learn from the mistakes.

6
Conclusion

The purpose of this project was to develop a web application for students where they can create a game by writing a game plan in plain English. The application should use modern coding practices and principles, and contain similar functionality as GameChangineer while being more user-friendly. In addition, the goals of the project were to:

- Develop the application by using modern practices and principles.
- Design for user experience.
- Make the content pedagogical.
- Generate easily understandable error messages.
- Successfully interpret game descriptions, including synonyms and errors.

Argama gives students the possibility to write a game plan in plain English and generate a game from it. Moreover, the application mainly uses modern coding practises and principles, such as modern praxis for HTML and JavaScript, as well as GF for defining the controlled natural language. However, the modern practices are lacking for some parts of the application, for example, HTTP is used instead of HTTPS.

The application contains similar functionality as GameChangineer, however, the functionality is not as complex. The game plan has to be written in the exact way as it is specified in the tabs, in other words it does not interpret synonyms, or errors.

Design-wise the website is less cluttered and easier to understand than GameChangineer according to the result from the user tests. The color scheme is neutral without colors that are too bold, and several design patterns are incorporated into the design in order to achieve a better user experience. The application takes educational aspects into consideration and interprets game plans to the extent of the defined CNL. However, the result does not contain functioning error messages, which makes it hard to say whether they are easily understandable or not.

Implementing generation of error messages, as well as further development of the controlled natural language could be further investigated in future projects. Using more modern principles and practices, such as HTTPS for the application server, and making the website more responsive are other possible improvements. With that said, the project outcome still fulfills the purpose, and therefore also the main goal of the project.

Bibliography

[1] GameChangineer. GameChangineer. URL https://gc.ece.vt.edu/k2c/.

[2] Stärkt digital kompetens i skolans styrdokument. Technical report.

[3] Michael S Hsiao. Automated Program Synthesis from Object-Oriented Natural Language for Computer Games. 2018. doi: 10.3233/978-1-61499-904-1-71.

[4] R Goebel, J Siekmann, and W Wahlster. Lecture Notes in Artificial Intelligence 5972 Edited by Subseries of Lecture Notes in Computer Science. Technical report.

[5] Norbert E Fuchs and Rolf Schwitter. Specifying Logic Programs in Controlled Natural Language. *Workshop on Computational Logic for Natural Language Processing*, abs/cmp-lg:16, 1995. URL http://arxiv.org/abs/cmp-lg/9507009.

[6] Krasimir Angelov and Aarne Ranta. Implementing controlled languages in GF. In Norbert E. Fuchs, editor, *International Workshop on Controlled Natural Language*, pages 82–101, Berlin, Heidelberg, 2009. Springer.

[7] Aarne Ranta. *Grammatical Framework Programming with Multilingual Grammars*. CSLI Publications, 2011.

[8] GF Resource Grammar Library: Synopsis Introduction. URL https://www.grammaticalframework.org/lib/doc/synopsis/.

[9] Grammatical Framework Tutorial. URL https://www.grammaticalframework.org/doc/tutorial/gf-tutorial.html.

[10] Aarne Ranta. The GF Resource Grammar Library. Technical Report 2, 2009.

[11] Jörg Krause. *Introducing Web Development [electronic resource]*. 2016. ISBN 9781484224991.

[12] Amirali Sanatinia and Guevara Noubir. On GitHub's Programming Languages. 3 2016. URL http://arxiv.org/abs/1603.00431.

[13] Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world., . URL https://getbootstrap.com.

[14] Bootstrap. Introduction · Bootstrap, . URL https://getbootstrap.com/docs/4.3/getting-started/introduction/.

[15] Bootstrap. Alerts · Bootstrap, . URL https://getbootstrap.com/docs/4.0/components/alerts/.

[16] Peter Gasston. *The modern Web [electronic resource] : multi-device Web development with HTML5, CSS3, and JavaScript*. 2013. ISBN 9781593274870.

[17] Bootstrap. Text · Bootstrap, . URL https://getbootstrap.com/docs/4.3/utilities/text/.

[18] Processing Exhibition. URL https://processing.org/exhibition/.

[19] Sinan Ascioglu. Games. URL https://www.openprocessing.org/curation/25/.

[20] Itch.io. Top Games made with Processing. URL https://itch.io/games/made-with-processing.

[21] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach: International Edition*. 2013. ISBN 9780273775638.

[22] Jörg Krause. Protocols of the Web. In *Protocols of the Web*, chapter 1, pages 1–25. 2016.

[23] Oracle. HttpServer (Java Http Server). URL https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html.

[24] Effie Lai-chong Law, Virpi Roto, Marc Hassenzahl, APOS Vermeeren, and Joke Kort. Understanding, scoping and defining user experience: a survey approach. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 719–728, 2009. doi: 10.1145/1518701.1518813.

[25] Jesse James Garret. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders Publishing, 2 edition, 2011. ISBN 9780321683687.

[26] Jennifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. 2010. ISBN 1449379702, 9781449379704.

[27] Jeff Rubin and Dana Chisnell. *Handbook of usability testing: how to plan, design, and conduct effective tests*. 2nd ed. edition, 2008.

[28] Hidekuni Tsukamoto, Yasuhiro Takemura, Yasumasa Oomori, Isamu Ikeda, Hideo Nagumo, Akito Monden, and Ken Ichi Matsumoto. Textual vs. visual programming languages in programming education for primary schoolchildren. In *Proceedings - Frontiers in Education Conference, FIE*, 2016. ISBN 9781509017904. doi: 10.1109/FIE.2016.7757571.

[29] Robert Heininger, Loina Prifti, Victor Seifert, Matthias Utesch, and Helmut Krcmar. Teaching how to program with a playful approach: A review of success factors. In *IEEE Global Engineering Education Conference, EDUCON*, 2017. ISBN 9781509054671. doi: 10.1109/EDUCON.2017.7942846.

[30] Daisuke Saito, Hironori Washizaki, and Yoshiaki Fukazawa. Work in progress: A comparison of programming way: Illustration-based programming and text-based programming. In *Proceedings of 2015 IEEE International Conference on Teaching, Assessment and Learning for Engineering, TALE 2015*, number December, pages 220–223. IEEE, 2016. ISBN 9781467392266. doi: 10.1109/TALE.2015.7386047.

[31] M. Paralič and E. Pietriková. Learning by game creation in introductory programming course: 5-Year-long study. In *ICETA 2014 - 12th IEEE International Conference on Emerging eLearning Technologies and Applications, Proceedings*, pages 391–396. IEEE, 2015. ISBN 9781479977406. doi: 10.1109/ICETA.2014.7107617.

[32] Liam Doherty and Vive Kumar. Teaching programming through games. In *International Workshop on Technology for Education, T4E'09*, 2009. ISBN 9781424455058. doi: 10.1109/T4E.2009.5314120.

[33] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1):68, 2000.

[34] Thomas W Malone and Mark R Lepper. Making learning fun: A taxonomic model of intrinsic motivations for learning. *Aptitude, learning, and instruction*, 3:223–253, 1987.

[35] Marina Papastergiou. Digital Game-Based Learning in high school Computer Science education: Impact on educational effectiveness and student motivation. *Computers and Education*, 2009. ISSN 03601315. doi: 10.1016/j.compedu.2008.06.004.

[36] Jane McGonigal. *Reality is broken: Why Games Make Us Better and How They Can Change the World JANE*. 2011. ISBN 9781101475492. doi: 10.1075/ni.10.1.03bro.

[37] S. Mininel, F. Vatta, S. Gaion, W. Ukovich, and M. P. Fanti. A customizable game engine for mobile game-based learning. In *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2009. ISBN 9781424427949. doi: 10.1109/ICSMC.2009.5346386.

[38] Richard Van Eck. Digital Game-Based Learning: It's Not Just the Digital Natives Who Are Restless. *EDUCAUSE Review*, 2006. ISSN 15276619. doi: 10.1145/950566.950596.

[39] Yun Ling, Huamao Gu, and Xun Wang. An Ontology-Based Development Framework for Edutainments. In *2008 International Seminar on Business and Information Management*, volume 1, pages 343–346. IEEE, 2008.

[40] Alexander Soska, Jürgen Mottok, and Christian Wolff. Playful learning in academic software engineering education. In *IEEE Global Engineering Education Conference, EDUCON*, 2015. ISBN 9781479919086. doi: 10.1109/EDUCON.2015.7095992.

[41] Fredrik Johansson. Presentation av pedagogiska texter påwebben, 2008.

[42] E-Nämnden. Vagledningen 24-timmarswebben 2.0. Technical report, Stockholm, 2003. URL https://www.uka.se/download/18.12f25798156a345894e2d7d/1487841932569/24timmarsvägledningen.pdf.

[43] Tommy Sundström. *Användbarhetsboken: bästa sätten att göra fungerande webb*. Studentlitteratur, 2005. ISBN 9789144037431.

[44] TTSpråket. Allmänna skrivregler. URL https://tt.se/tt-spraket/skriv/.

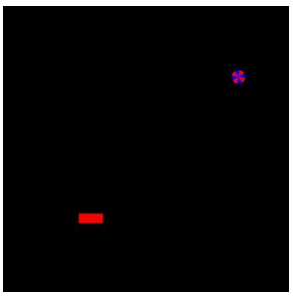
[45] Christopher G. Lasater. *Design patterns [electronic resource]*. 2007. ISBN 1598220314.

A

First user test

Användartest: Gamechangineer

Gamechangineer har i USA testats på fourth-graders och i middle school, det vill säga i åldrarna 9-13 år. Detta test bör göras med ett barn i den åldern, eller med en äldre person som har bra datorvana och kunskaper i engelska men som inte är kunnig i programmering.



I testet ska programmet Pong skapas. I Pong finns en boll som studsar och en "sten" (brick) spelaren kan flytta fram och tillbaka. Om bollen nuddar stenen studsar den. Spelet är över om bollen nuddar nederkanten.

Försök att styra användaren så lite som möjligt under testet, egentligen ingenting förrän det behövs hjälp (bra om den provar själv först). Be användaren förklara vad den gör och tänker under hela processen.

Testet

Börja med att förklara för användaren om att det här är ett test för att utvärdera den här hemsidan. Uppmuntra användaren till att säga vad de tänker och att det inte finns något rätt eller fel svar.

Öppna startsidan.

Notera: Vad gör användaren? Bläddrar ner, klickar på något, frågar något?
Svar: 1. Kollar runt på sidan. 2. Bläddrar ner och kollar över innehållet lite snabbt. 3. Börjar läsa överst, "Frågar om vi gjort sidan" 4. Börjar läsa scrolla runt lite, frågar om hen måste signa in. 5. Bläddrar ner, kollar runt lite. 6. Frågar om sign in. "Definitivt en backendare som gjort den här sidan". 7. Scrollade ner lite. Sedan klickar de på "try it out".
Fråga: Vad är ditt första intryck?
Svar: 1. Helt okej.

2. Den ser okej ut.

3. Rörig, hinner inte läsa texten som rör sig, inte särskilt snygg sida.

4. "Det är lite grejer här och där, väldigt mycket på första sidan"

5. Lite rörig.

6. Plottrig. Den skulle må bra av övermenyer.

7. Oklart och otydligt. "Try it out" är otydlig vad den gör och liten. De rörande elementen är störande. För mycket text, alldeles för mycket text till 9-13 åringar.

Fråga: Be användaren berätta med egna ord om vad den tror hemsidan är till för.

Svar:
1. Vet inte.

2. Man ska göra ett spel.

3. Lära sig om objektorientering och göra spel, skall vara kul.

4. "Verkar som man kan göra spel." Läste main idea.

5. Inte en aning.

6. Man ska kunna skriva ner regler för ett spel på ren engelska och så kommer den att skapa ett spel för det.

7. Otydligt vad hemsidan är till för!

Be användaren gå vidare till spelsidan. Vänta och se om användaren hittar, annars visa knappen "Try it out" på vänstersidan.

Förklara samtidigt vad hemsidan är till för och vad ni ska göra, det vill säga "programmera" ett spel med bara vanligt språk. Berätta att ni ska göra spelet Pong, förklara vad det är om användaren inte redan vet.

Notera: Vad gör användaren? Bläddrar ner, klickar på något, frågar något?

Svar:
1. Klickar direkt på try it out-knappen.

2. Frågar var man går vidare.

3. Börjar läsa.

4. "oh shit" börjar läsa till vänster (blå ruta).

5. Scrollar massa, mycket förvirrad. Tycker knappen var dold, tror man måste logga in. Tycker inte om rörlig text.

6. Trycker på logga in med google. Väljer sitt konto på google.

7. Tyckte de va lite för få characters. Skrev in: "There is a ball. There are two bricks. When the ball hits one brick the ball will fly away." Man orkar ej läsa error meddelanden. Ändrade sin kod till "There is one ball. Ther bricks are the characters. The ball is positioned between character object one and object two.

Fråga: Vad är ditt första intryck?

Svar:
1. "Förstår inte så mycket."

2. "Man ska nog välja en karaktär."

3. Mycket text, hade varit gött med en lättsammare text. Förklarade för mycket direkt från börja, hade varit bättre att få information stegvis.

4. "Ser ut som en hemsida".

5. Inte rent, väldigt mycket text.

6. Också plottrig. Väldigt "90-talswebb". Inga marginaler, blir väldigt ihoptryckt. Vänsterpanelen med quickguide är bra men behöver kanske inte vara där hela tiden.

7. Error meddelanden är otydliga, det är ej gjort för 9-13 åringar? Alldeles för mycket information.

Be användaren att börja skapa sitt spel. Försök att *hjälpa till så lite som möjligt* i processen.

Den färdiga "koden" finns längst ner i detta dokument. Tanken är att användaren inte ska se den färdiga koden utan att skriva den själv och med din hjälp.

Om användaren fastnar, försök förklara pedagogiskt genom att dela upp spelet i tre delar; "karaktärer" och dess placering, rörelse och vem som gör den, samt kravet för game-over. Gör en sak i taget, när du förklarat något så låt användaren fortsätta själv. Uppmuntra användaren till att läsa informationen som finns på sidan. Ställ ledande frågor istället för att bara säga svaret rätt ut.

Notera: Börjar användaren med att fylla i titel och namn? Går det utan problem?

Svar:
1. "Förstår ingenting". Börjar med att läsa. Börjar med att fylla i namn, sen titel.

2. Det går bra!

3. Inget problem med titel eller namn. 4. Inga problem med titel eller skapare. 5. Ja. 6. Fyller i titel och namn utan problem. 7. Fyllde i titel men glömde fylla i namn.
Notera: Hur lätt/svårt är det för användaren att skapa sitt spel?
Svar: 1. Svårt att börja. 2. Börjar med att kolla exempel. 3. Svårt fastnar mycket, tar lång tid innan hen börjar kunna skriva fungerande text. 4. Var tvungen att visa execute knappen. 5. Lite förvirrad till en början, kollar runt på sidan. Försöker dra upp djuren. Börjar sedan skriva. 6. Börjar skriva "korrekta" engelska meningar i rimlig ordning och med ett logiskt tänkande. Provkör. Får fel. "Tydliga felmeddelanden". Men det fanns inget sätt att veta hur man ska skriva från början. Hittar "guiden" längst ner på sidan (lexikonet). Försöker att rätta till statements utifrån felmeddelanden. Funderar mycket. Försöker att lägga till väggar. Nytt felmeddelande. Går inte att lägga väggar runt kartan. Går över till "maptextboxen". Gillar map keys. "Varför står inte saker i bokstavsordning?". Bygger en karta i ascii. "Kan man inte få måla istället, typ med en palett så man kan placera ut karaktärer?". Att map keys var kvar medan man skrev uppskattades. Får ett felmeddelande om att blank spaces måste representeras som - när man använder ascii. Fixar det, får felmeddelande om att asciin är för bred för kartan, tar bort några tecken och testar igen. Får tips om att frångå väggar. Skriver nu en sak i taget och provar om det fungerar. Förstår principen och fixar att göra Pong. Lyckas skriva väldigt logiska meningar. Försöker lägga in att bollen ska öka fart vid studsar, "fysikmotorn lämnar lite att önska". "Hade velat kunna definiera exakt var paddeln ska ligga". 7. När man väl kom igång så blev det lätt.
Notera: Vad gör användaren när den fastnar? Försöker den att leta efter information själv eller frågar den dig?
Svar: 1. Räcker med små hintar och så provar hen och kommer framåt. 2. Frågar om hjälp, testar. 3. Ställer många frågor innan hen börjar. Gillar inte begränsningen av available characters. När hen kommit igång läser mycket i instructions rutan. 4. Börja läsa felmeddelanden, formulerar om meningar, blir tydligare. Är fast vid sin

terminologi. Började ställa frågor efter ett tag.

5. Använder sig av hjälpen som kommer upp under, kollar runt lite. Läser felmeddelanden.

6. Letar information själv. När det blir mycket felmeddelanden försöker användaren utforska olika lösningar. Snurrar in sig i ett svårt problem när det går att lösa mycket enklare.

7. De löste det mesta själva. Får flika in med hjälp när de behövde det.

Notera: Hur tycker du att det går för användaren? Smidigt eller fastnar den mycket? Skriv från din egen synvinkel.

Svar:
1. Tycker det går väldigt smidigt, ibland svårt med stavningen på vissa ord. Men förstår hur allt hänger ihop.

2. När hen förstår ett koncept så provar den själv och kommer framåt. Svårt med nya grejer. Får ett felmeddelande om corner istället för border, löser det med lite hjälp.

3. Fastnar ganska så mycket, har problem med storleken på rutan man skriver in text i. Har svårt att lösa att bollen studsar mot brick och kanterna. Testar att använda Map textrutan, blir väldigt förvirrad. Gav upp på att använda kartan.

4. Förstår tanken men tar tid att få fram sina idéer.

5. Lite segt, hänger inte riktigt med i tänket. Två meningar först, läser sedan felmeddelandet och försöker förstå felet. Fastnar på att det behöver specificeras mycket, "inte så enkelt". Fastnar i hur man måste deklarerar objekten. Läser felmeddelanden, inser att man måste säga var de ska befinna sig. Svårt att förstå hur det hänger ihop och vad som ska skrivas. Förstår inte hur noga man behöver förklara. Förstod inte och gav tillslut upp pga tidsbrist efter 20 min.

6. Fastnar mycket. När användaren fått tips om hur tankesättet ska vara så gick det enkelt.

7. Jag tyckte att det löste det väldigt bra, kreativa trots att hemsidan såg ut som den gjorde.

När användaren gjort klart spelet så kan du avsluta med att ställa följande frågor.

Fråga: Känns det som att du lärt dig något nytt under det här testet?

Svar:
1. Lärde mig verktyget.

2. Det var enklare än jag trodde.

3. Man ska vara tydligare när man skriver, referera inte med "it". Upprepa namnet för tydlighet.

4. "Ja, detta var jätteroligt".

5. Nej.

6. Njae, inte så mycket i och med att jag kan programmera. Känns dock som ett smidigt sätt att göra det på, så det är principiellt lärorikt. Som programmerare blir vissa saker uppenbara, exempelvis varför bollen försvinner ur plan. Men man kan nog lära sig mycket om man inte haft koll på det innan.

7. De lärde sig inte så mycket nytt. Men man måste vara väldigt noga med vad man säger åt den att göra.

Fråga: Var det kul? Hade du velat göra ett annat spel eller bygga på fler funktioner på Pong?

Svar:
1. Ja. Kan tänka mig göra nåt annat.

2. Ja, det var det väl. Kan tänka mig att göra ett annat.

3. Hade velat använda fler av figurerna.

4. Vill göra mer funktioner, t.ex att bollen blir snabbare.

5. Man behöver ha lite mer tid på sig, och det borde finnas exempel. Exempel spel hade varit lättare så att man kan skriva om det, annars svårt att veta vart man ska börja tänka.

6. Ja! Hade kunnat tänka mig göra något annat.

7. Ja. Hade kunnat lägga till lite funktioner. Kanske, hade varit kul att göra lite annorlunda grejor.

Fråga: Vad var det svåraste med att använda hemsidan?

Svar:
1. Vet inte.

2. I början och att lära sig allt, sen var det enkelt.

3. Texten med errors var svårt att förstå, rutan var förvirrande. Hantera kartan

4. Att man måste vara så noggrann med sin syntax, var hjälpsam med att förklara vad som man kunnat mena.

5. Rörig, fruktansvärt lite som man förstår. Måste vara tydligare, tydligare med att förklara vad det går ut på. Exempel - ett spel skulle kunna se ut såhär.

6. Det svåraste var att det inte var uppenbart, jag visste vad jag ville göra men inte vilket språk jag skulle använda för att det skulle parsas.

7. Jätteotydliga instruktioner. Men ingenting som va riktigt svårt. Sidan känns slarvig.

Fråga: Fanns det något du ville göra men inte kunde?

Svar:
1. Näe, tyckte inte det.

2. Nej.

3. Kände inte att saker saknades för uppgiften som utfördes. 4. Click and drag för att lägga till saker i spelet (Dra från alla objekt som fanns direkt till spel rutan) 5. Jättemycket, kunde ingenting. 6. Jag hade velat ha ett tydligt sätt att interagera med canvasen från början. Typ som en inforuta som förklarar på vilka sätt du kan använda den. Jättebra definition på karaktärerna men ingen på till exempel borders. 7. Hade varit kul att kunna använda ljud.
Fråga: Hade du velat att det skulle fungera annorlunda på något sätt?
Svar: 1. Nej. 2. Inget jag tänker på. 3. Ändra förklaringarna när man gjort fel. Det var för mycket text i början "words to consider in your text" skulle komma tidigare så man hittar det tidigare. Hade velat se hur koden ser ut i slutändan. 4. Se ovan. 5. Exempel. 6. Främst ett smidigare UI. Förstår tanken med snabb feedback men det är ganska grötigt. Hade velat förstå vad som händer på ett tydligare sätt. Bra felmeddelanden. Logiken bra, användarvänligheten sämre. Vet inte om jag är så förtjust i "understood with 91% certainty", lite för mycket info möjligtvis. Kanske bättre med varningar vid låg procentandel. Notes är lite mycket info per default. Hade velat se det som en kompileringskonsoll. 7. Spelet borde vara i mitten och större och textrutan borde vara längst ner.
Fråga: Några andra tankar?
Svar: 1. Nej. 2. Nej. 3. Inga andra tankar 4. Det var roligt, "tänk att jag var så skillad" 5. Näe. 6. Häftigt koncept. 7. Nej.
Hur lång tid tog det?:

1. 30 min.

2. 30 min.

3. 50 min.

4. 30 min.

5. Gav upp efter 20 min.

6. 45 min.

7. 38 min.

Ett facit:

There is one ball near the top.
There is 1 fast brick near the bottom.

The player controls the brick.
When right arrow is pressed, the brick moves right.
When left arrow is pressed, the brick moves left.
The ball starts out moving in a random direction at 3 pixels per frame.
When the ball reaches the top border, it reverses direction.
When the ball reaches the right border, it reverses direction.
When the ball reaches the left border, it reverses direction.
When the ball collides with the brick, it reverses direction.

When the ball reaches the bottom border, the game is over.

B

Mock ups

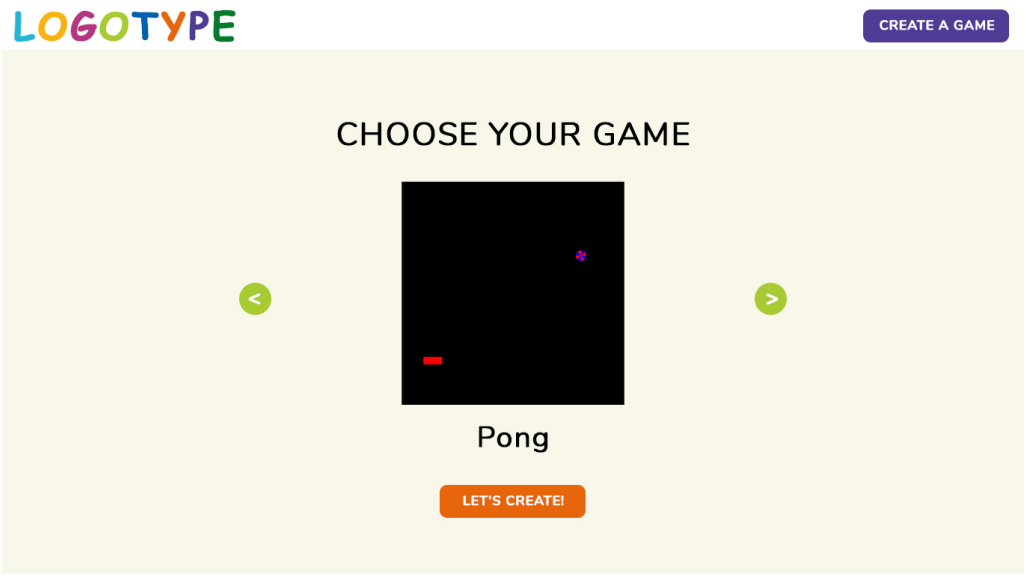


Figure B.1: The mock up for the index view.

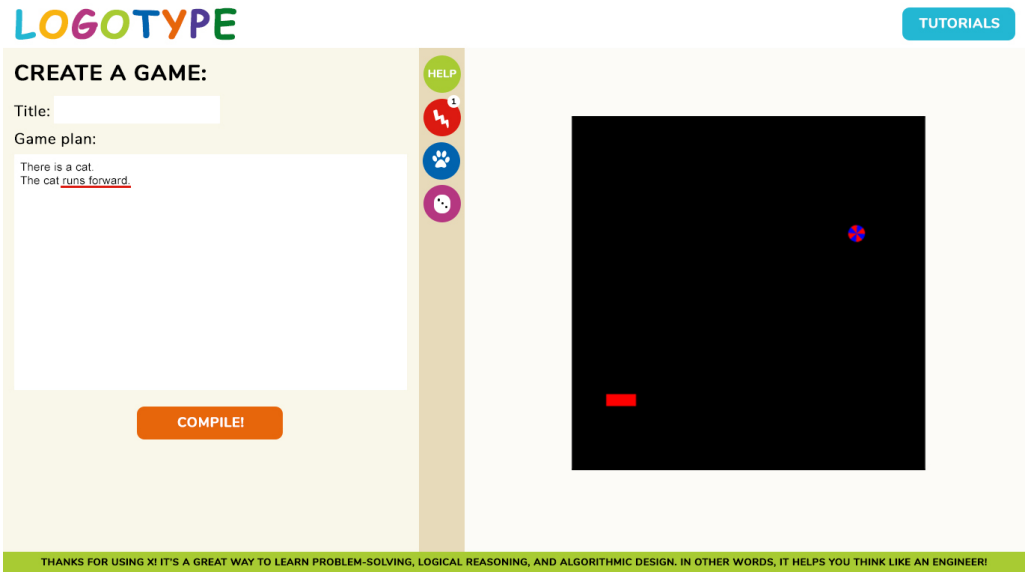


Figure B.2: The mock up for the game view.

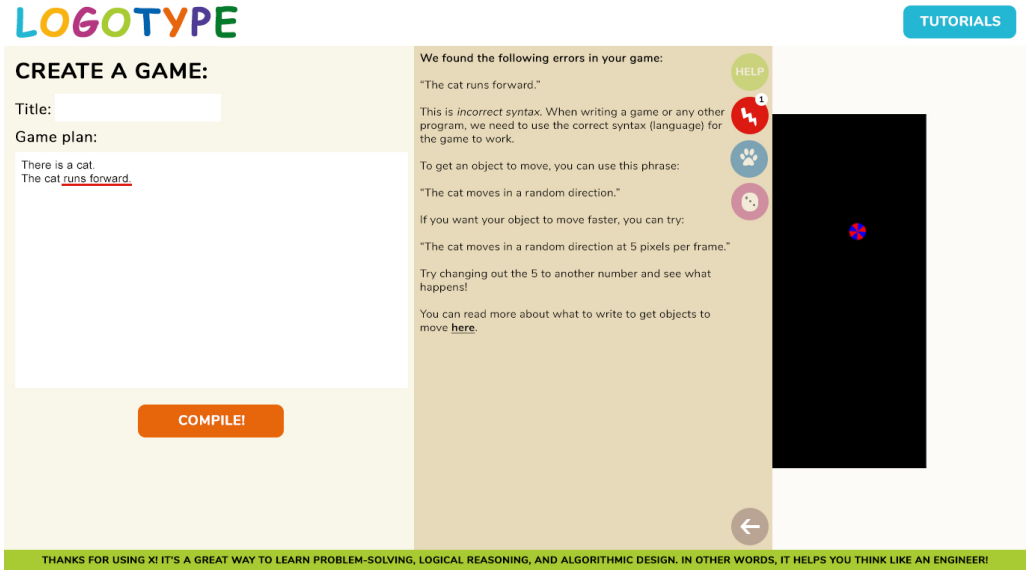


Figure B.3: The mock up for the game view when the *Error* tab is open.

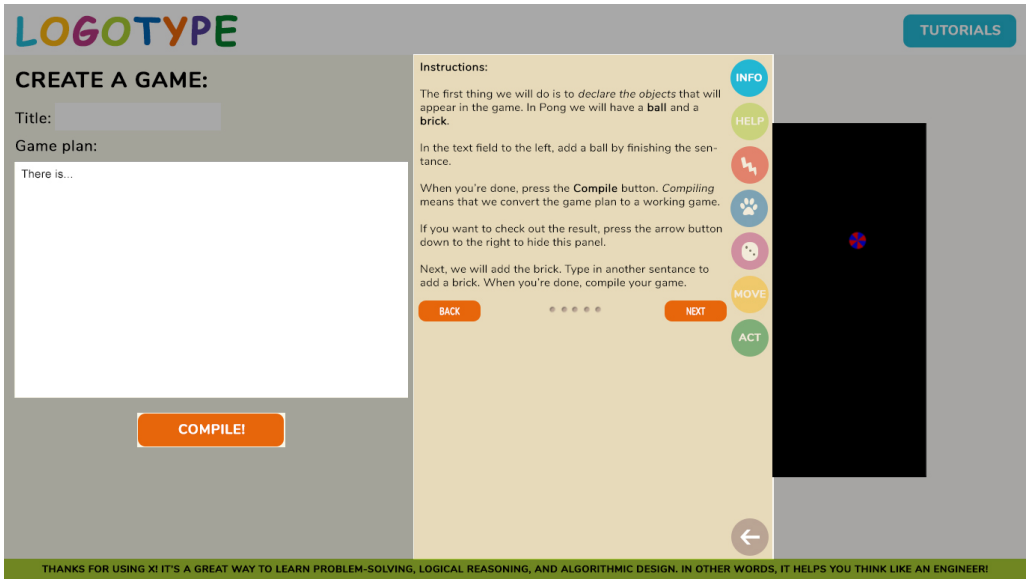


Figure B.4: The mock up for the game view with the first page on the tutorial for Pong.

C

The website

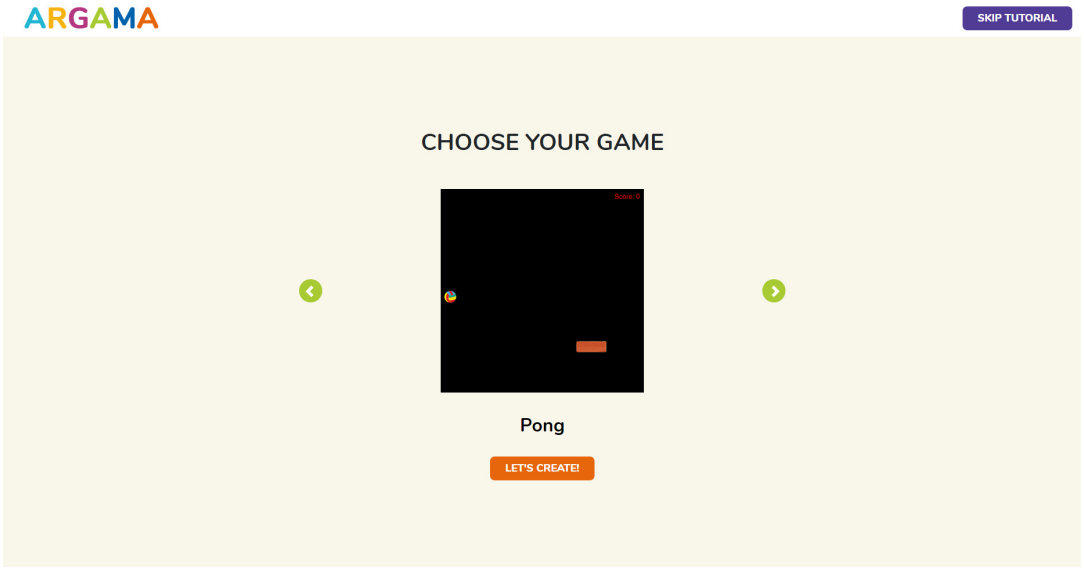


Figure C.1: The index view of the website.

Figure C.2: The websites game view.

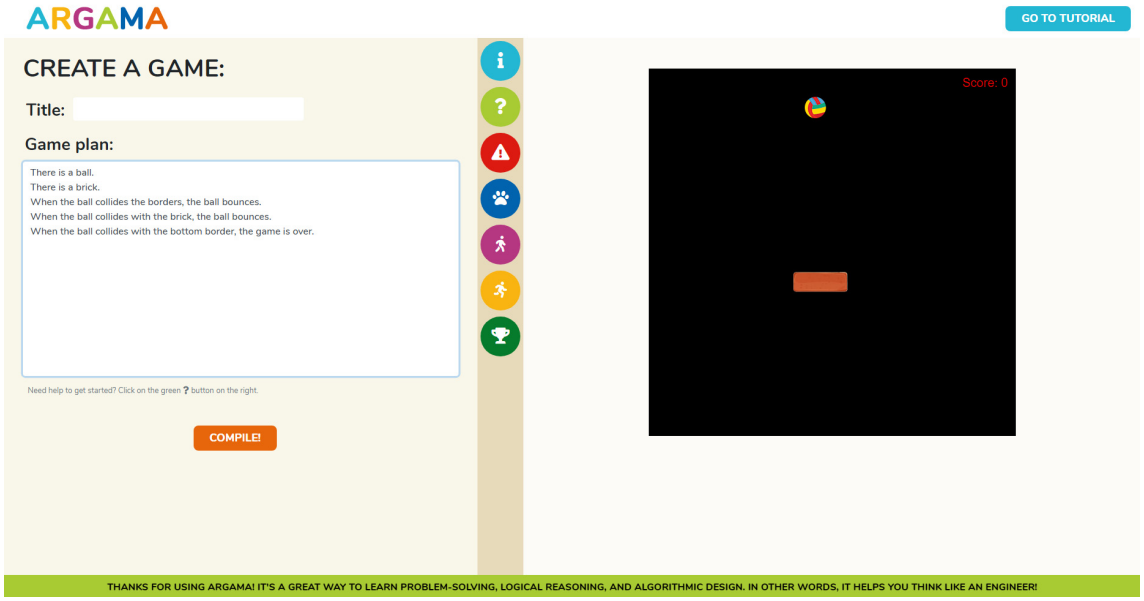


Figure C.3: The game view when the game Pong is created.

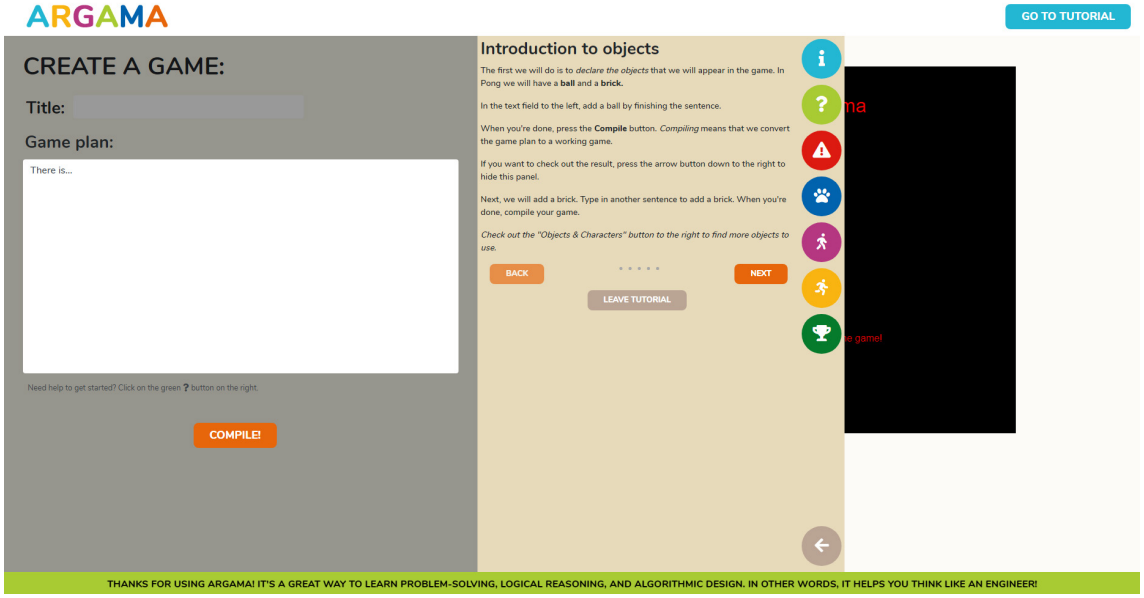


Figure C.4: The game view with the first page of the tutorial for Pong open.

Figure C.5: The game view with the *Help* tab is open.

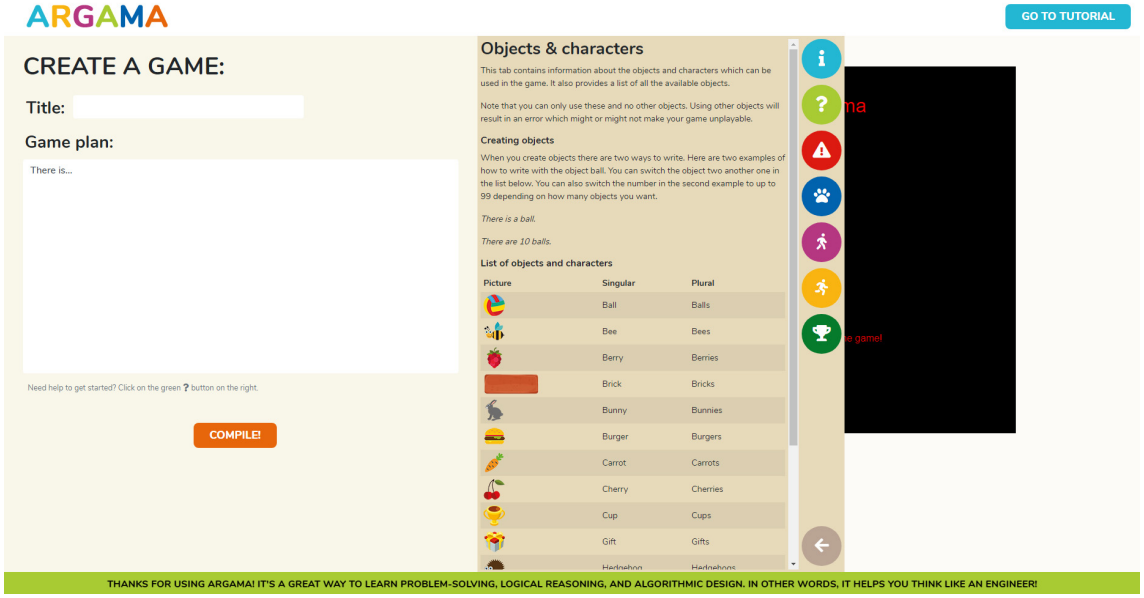


Figure C.6: The game view with the *Objects & Characters* tab is open.

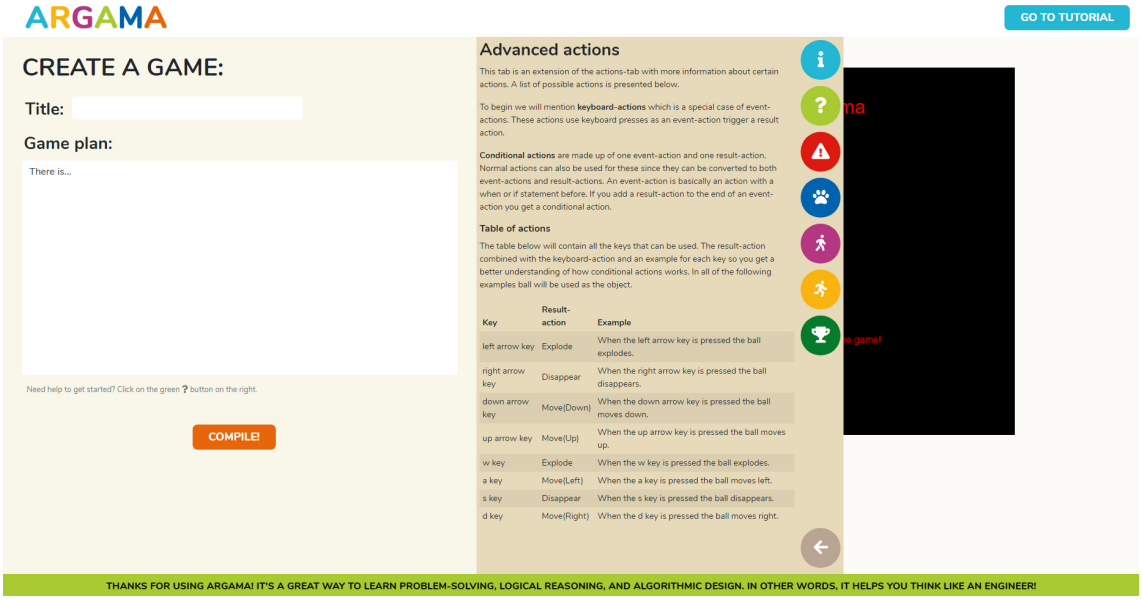


Figure C.7: The game view with the *Advanced actions* tab is open.

D

Second user test

Användartest: Argama

I testet ska programmet Pong skapas. I Pong finns en boll som studsar och en "sten" (brick) spelaren kan flytta fram och tillbaka. Om bollen nuddar stenen studsar den. Spelet är över om bollen nuddar nederkanten.

Försök att styra användaren så lite som möjligt under testet, egentligen ingenting förrän det behövs hjälp (bra om den provar själv först). Be användaren förklara vad den gör och tänker under hela processen.

Testet

Börja med att förklara för användaren om att det här är ett test för att utvärdera den här hemsidan. Uppmuntra användaren till att säga vad de tänker och att det inte finns något rätt eller fel svar.

Öppna startsidan.

Notera: Vad gör användaren? Bläddrar ner, klickar på något, frågar något?
Svar: 1. Användaren tryckte på create game. 2. Jag förklarar att vi inte ska använda tutorialen. Klickar in på skip tutorial. 3. Kollar runt, läser på sidan. Bläddrar runt bland spelen. Går in på tutorialen för pong. Förvirrande att den bytte bild på startsidan. Lite förvirrande att den heter tutorial. 4. Användaren klickade på tutorial, och klagar på engelskan. 5. Klickar runt på pilarna och kollar vad det finns för olika tutorials
Fråga: Vad är ditt första intryck?
Svar: 1. Känns som om jag har gjort något sånt här förr. 2. Tydligt uppbyggt med titel och spelplan. Översta symbolerna i menyn är väldigt tydliga. Resten av dem är lite otydliga men om man håller över så syns det vad alla är till för. 3. Ren och snygg. 4. Fin, choose your game kanske ska stå något annat, så man vet att man ska skapa ett spel.

5. Känns som att hemsidan är mer inriktad mot barn, men ändå ska användas också av ikke barn.
Fråga: Be användaren berätta med egna ord om vad den tror hemsidan är till för.
Svar: 1. Vet inte, göra ett spel typ. 2. Skapa ett spel på något sätt. 3. Visste det sedan innan, skapa ett eget spel. 4. Att göra ett spel. 5. Att man kan skapa spel, via att följa tutorial eller testa sig fram själv.

Be användaren gå direkt vidare till spelsidan (inte via tutorial). Vänta och se om användaren hittar, annars visa knappen "Skip tutorial" på vänstersidan.

Förklara samtidigt vad hemsidan är till för och vad ni ska göra, det vill säga "programmera" ett spel med bara vanligt språk. Berätta att ni ska göra spelet Pong, förklara vad det är om användaren inte redan vet.

Notera: Vad gör användaren? Bläddrar ner, klickar på något, frågar något?
Svar: 1. Kollar in på objects efter snabbt ha "svävat" med pekaren för att se vad de olika flikarna är och betyder. "Frågar ska jag skriva det där?" Skriver in "There is a ball" utan att trycka compile skriver användaren sedan in "There is a brick". Användaren går sedan in bland actions. "Användaren verkar ha svårt att inse att "compile" innebär kör spelet ett namn byte hade kunnat vara aktuellt för att förbättra detta. Användaren har inte lagt en punkt mellan meningarna så när compile trycktes så gick de ihop. Inte helt självklart att det ska vara en punkt efter meningarna dvs att dessa är line-breakers. Efter användaren varit fast lite så tipsar jag om att informationen i Help är tydligare och då navigerar användaren dit. Efter ett tag lyckas användaren ganska kvickt utan fel att skriva ihop kollision för att bollen rör sig vänster, kollision för stenen och för väggarna. Användaren fastnade lite vid actions för att röra bricks men lyckas navigera till advanced actions för att sedan därifrån ganska snabbt komma igång med att göra så att den kunde röra sig med piltangenterna åt höger och vänster. Det existerade ett fel i "help" där det stod hits istället för collides vilket förvirrade användaren. 2. Börjar kolla runt, läser först info, sedan help. 3. Skriver i titeln. 4. Klickade på tutorialen, började följa den påpekade att skippa den. Började skriva in saker och testa att kompilera.

5. Börjar att läsa igenom de olika flikarna, går uppifrån och ner.
Fråga: Vad är ditt första intryck?
Svar: 1. Ser tydligt ut vad man gör, är tydligt på gubbarna vad de gör för något. 2. Snygg design. Lätt att hitta vad man ska göra. 3. Ren och snygg, fattar ingenting. 4. Färgerna är olika i varje element (divs). Men de färgglada inslagen är jättefina, ikonerna är jättefina. Compile knappen borde vara större. 5. Vet inte helt vad de sakerna i mitten är för.

Be användaren att börja skapa sitt spel. Försök att *hjälpa till så lite som möjligt* i processen.

Den färdiga "koden" finns längst ner i detta dokument. Tanken är att användaren inte ska se den färdiga koden utan att skriva den själv och med din hjälp.

Om användaren fastnar, försök förklara pedagogiskt genom att dela upp spelet i tre delar; "karaktärer" och dess placering, rörelse och vem som gör den, samt kravet för game-over. Gör en sak i taget, när du förklarat något så låt användaren fortsätta själv. Uppmuntra användaren till att läsa informationen som finns på sidan. Ställ ledande frågor istället för att bara säga svaret rätt ut.

Notera: Börjar användaren med att fylla i titel och namn? Går det utan problem?
Svar: 1. Användaren har inte fyllt i namn innan användaren början skriva i textrutan. 2. Börjar fylla i titel utan problem. 3. Fyller i titel, hittar instruktionerna. Går utan problem. 4. Fyller inte i titel. 5. Skippar att fylla i titel helt.
Notera: Hur lätt/svårt är det för användaren att skapa sitt spel?
Svar: 1. Det börjar ganska sakta då användaren läser mycket i object/actions. Användaren får dock sen väldigt snabbt ihop kollision med vägen och brick när den väl kommit vidare lite. 2. Det går ganska lätt. Provar att skapa en boll utan problem. Testar att sätta rörelse på bollen, funkar. Får testa lite olika sätt att få border bounce att funka, använder hjälp och ser att det ska skrivas på ett exakt sätt. Lägger in en brick och sätter kontroller på den utan problem.

3. Lite svårigheter, men lägger till sin egna touch, men majoriteten av beskrivningen blir exakt enligt "facit" tack vare tabsen. 4. Gick ganska snabbt när flikarna började användas. 5. Kommer väldigt snabbt igång med att skapa en ball och en brick, tar lite tid innan hen kollar vidare i de andra tabsen än "help"
Notera: Vad gör användaren när den fastnar? Försöker den att leta efter information själv eller frågar den dig?
Svar: 1. Användaren försöker fråga på vilket jag svarar "försök själv innan du frågar om hjälp". Navigerar sig till de olika flikarna för hjälp. 2. Använder sig av de olika knapparna och läser exemplen där. 3. Läser på i menyerna, framförallt i listorna med actions och karaktärer som finns för att förstå hur man skapar pong. Fick tips om att advanced actions-taben fanns och även att man inte kan skriva it (ska vara the ball istället). 4. Skriver in "there is a ball." Undrar hur man får bollen att röra på sig. Hittar actions skriver in "the ball wanders." ändrar detta sedan till "the ball moves left." sen till "the ball moves randomly". Undrar ifall man behöver skriva in alla borders för att få den att studsa. Lyckas efter att ha tittat i flikarna att få bollen att studsa på kanterna. 5. Läser noggrant igenom "help" för att se om hen kan hitta info där, kollar sedan i olika tabsen för att se om hen kan komma vidare där.
Notera: Hur tycker du att det går för användaren? Smidigt eller fastnar den mycket? Skriv från din egen synvinkel.
Svar: 1. Användaren tar mycket tid för "research" men fastnar egentligen inte jättemycket. 2. Går väldigt smidigt även om användaren inte är van vid verktyget. 3. Smidigt att hitta till menyn och förstå hur han ska komma igång. Fastnar på små saker såsom att det inte finns någon möjlighet att låta en karaktär flytta sig självmant i sidled. Kommer ändå igång bra med att hitta hjälpen och läser på för att förstå hur actions etc. fungerar. 4. Går bra för personen men verkar inte ha fullt fokus. Frågar först kollar sedan på flikarna. Går ganska bra generellt men jobbigt utan felmeddelanden och vissa funktionaliteter. 5. Går bra för personen, strukturerar sin kod väl, separerar mellan olika "områden" så som att deklarerar objekt, player control, interaktion mellan objekten och end conditions. Hakar upp sig på de specifika meningarna som behövs.

När användaren gjort klart spelet så kan du avsluta med att ställa följande frågor.

Fråga: Känns det som att du lärt dig något nytt under det här testet?
Svar: 1. Ja, men kan inte riktigt sätta grepp på exakt vad. 2. Lärt mig verktyget. 3. Ja. 4. Nej inte egentligen. 5. Nja, not really
Fråga: Var det kul? Hade du velat göra ett annat spel eller bygga på fler funktioner på Pong?
Svar: 1. Nej. 2. Ja, det var roligt. Kul att man ser direkta resultat. 3. Ja, det hade jag faktiskt kunnat tänka mig. 4. Det var kul, kan utveckla det som man vill typ. 5. Det var kul, väldigt intressant. Hade velat göra mer om det fanns möjlighet.
Fråga: Vad var det svåraste med att använda hemsidan?
Svar: 1. När det stod fel. 2. Lite hämmande att vara tvungen att skriva exakt som det står i hjälpen, hade varit nice att kunna skriva med synonymer. 3. Att den kraschade. 4. Lite få exempel, så visste inte alltid exakt hur man skulle skriva. Men annars fanns det bra hjälpmedel. 5. man vet inte riktigt exakt vilka “key-words” som fungerar. Ville ha lika bra beskrivet som det är för advanced actions för borders också, även att hitta text om att randomly finns.
Fråga: Fanns det något du ville göra men inte kunde?
Svar: 1. Ja, flytta var stenen börjar. 2. Nej, inget jag kommer på på rak arm. 3. Randomly fanns inte med, fick hjälp med den. Att det finns en färdigt spel, av den enklaste varianten som ett simpelt exempel program. 4. Nej, jag provade inte mer eller kanske bestämma vart allt hamnar i början.

5. Känns begränsat, hade velat ha fler reglar för kollision, kunna sätta startposition.
Fråga: Hade du velat att det skulle fungera annorlunda på något sätt?
Svar: 1. Annat språk, gärna svenska. 2. Hade varit nice om borders inte var "låsta" som standard, eller om det gick att ändra i alla fall. Vore bra om man när man har öppet en tab och trycker på samma knapp igen, så stängs tabsen. 3. Nej, det hade kunnat vara kul precis som det är. 4. Ne asså typ inte, bra att man inte slängs in i det direkt och har en förstasida. 5. Direkt när kommer till hemsidan hade det varit nice med någon introduktion om vad hemsidan är för något.
Fråga: Några andra tankar?
Svar: 1. Nej. 2. Nej, coolt arbete. 3. Kan vara bra att ha en spara funktion om man vill jobba under en längre tid med. 4. Nej, men bra jobbat. 5. Kan vara bra ur en pedagogisk synpunkt att man behöver definiera score så man lär sig att man behöver skriva ner allt i programmering och inte kan anta att saker finns.
Hur lång tid tog det?: 1. 26 min 2. 15 min 3. 20 min 4. 25 min 5. 40 min

There is a ball.
There is a brick.
The ball moves randomly.
When the right arrow key is pressed the brick moves right.
When the left arrow key is pressed the brick moves left.
When the ball collides with the border the ball bounces.
When the ball collides with the brick the ball bounces.

When the ball collides with the south border the game is lost.