CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Dynamic State Representation for Homeostatic Agents

Master's thesis in Complex Adaptive Systems

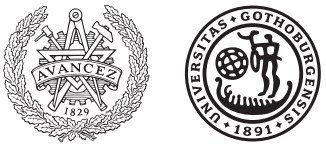
FREDRIK MÄKELÄINEN
HAMPUS TORÉN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Dynamic state representation for homeostatic agents

Fredrik Mäkeläinen
Hampus Torén



Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Dynamic state representation for homeostatic agents
Fredrik Mäkeläinen
Hampus Torén

Supervisor: Claes Strannegård, Computer Science and Engineering
Examiner: Christos Dimitrakakis, Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Dynamic state representation for homeostatic agents
FREDRIK MÄKELÄINEN AND HAMPUS TORÉN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In a reinforcement learning setting an agent learns how to behave in an environment through interactions. For complex environments, the explorable state space can easily become unmanageable, and efficient approximations are needed. The Generic Animat model (GA model), heavily influenced by biology, takes an approach utilising a dynamic graph to represent the state. This thesis is part of the *Generic Animat research project* at Chalmers that develops the GA model. In this thesis, we identify and implement potential improvements to the GA model and make comparisons to standard Q-learning and deep Q-learning. With the improved GA model we show that in a state space larger than 2^{32}, we see substantial performance gains compared to the original model.

Keywords: animat, autonomous agents, reinforcement learning, adaptive architectures, open ai, policy discovery, state representation, DQN, homeostatic agent.

Acknowledgements

We would like to thank our supervisor Claes Strannegård, for making this thesis possible and giving us great support along the way. We also want to thank Christos Dimitrikakis, our examiner, for great feedback, and Rasmus Åkerlund and Dorian Valverde for many interesting discussions. And last but not least a big thanks to our respective families for their support, patience and understanding while we completed this thesis.

Hampus Torén and Fredrik Mäkeläinen, Gothenburg, June 2018

Contents

1
Introduction

In this chapter, we give a brief background for applications and research within artificial intelligence. Furthermore, we introduce the animat model and the Generic Animat model, and their connections to biology. The research question and our goals are then presented, and the last section outlines the remaining chapters of this report.

1.1 Background

In recent years machine learning and artificial intelligence (AI) has taken tremendous leaps forward in many areas. Computers now regularly outperform or match humans on tasks such as digit recognition [1], games [2, 3] and medical imaging analysis [4]. A large part of this advancement stems from the increased computing power available and the progress made regarding neural networks and deep learning. However the above examples are usually highly specialised and require vast amounts of data to train the model on, there are for instance still no robots that can perform basic household tasks close to human level. For robots to achieve such functionality, they must be able to adapt to new environments far beyond their present ability. Building systems that are both flexible and autonomous are the goal of a sub-field of AI called artificial general intelligence.

Using nature as inspiration has in numerous cases proved itself a successful approach towards designing algorithms for solving complicated problems. Nature has a way to find elegant solutions, which might not be optimal but often are good enough. Dorigo et al. [5] introduced the *Ant Colony Optimisation* approach for shortest path problems such as *the travelling salesman*. While Eberhart and Kennedy [6] drew inspiration from bird flocking, fish schooling and swarming theory for the *Particle Swarm Optimisation* algorithm.

This leads us to the *Animat* model introduced by Wilson [7], which also draws inspiration from nature in order to solve the adaptability, learning and survivability problems that animals face. To define this model, Wilson identified four basic characteristics of simple animals, quoting from [7, p. 16-17]

1. *"The animal exist in a sea of sensory signals. At any moment only some signals are significant; the rest are irrelevant."*

2. *"The animal are capable of actions (e.g. movement) which tend to change these signals."*

3. *"Certain signals (e.g. those attendant on consumption of food), or certain signals' absence (e.g. absence of pain) have special status for him."*

4. *"He acts, both externally and through internal operations, so as approximately to optimise the rate of occurrence of the special signals."*

Wilson suggested that the animat model can be used as a way towards artificial general intelligence [8].

The Psikharpax project [9] aims to build an artificial rat, that can learn to navigate and survive in an unknown environment using neural networks and actor-critic models. Yoshida [10] applied reinforcement learning algorithms on homeostatic agents to optimise their survivability.

1.1.1 The Generic Animat model

The Generic animat model proposed by Strannegård et al. [11] is constructed with the goal to build an artificial animal that is capable of performing fundamental animal tasks such as foraging, locomotion, and navigation. It uses generic mechanisms for perception, action, learning, and decision-making, and at its core, there is a dynamic graph built upon the animals environmental perception. Primitive animal behaviour has been successfully modelled using the Generic Animat model [11], as an example, a bee has been modelled that was able to navigate in an environment while collecting nectar.

1.1.2 Research questions

This thesis investigates how the Generic Animat model performs in an environment with a large state space. The analysis is separated into the following steps:

- Analyse the Generic Animat model and identify areas suitable for improvement, using both theoretical and practical approaches.
- Improve and offer suggestions, regarding identified areas.
- Benchmark the Generic Animat model against other established models that could be used in an animat context.

Additionally, a large focus should be put on producing well written and modular code, to provide a good open-sourced basis for continued research regarding the Generic Animat model.

1.1.3 Report outline

In chapter 2, we present brief introductions to reinforcement learning, Q-learning and deep Q-learning. Additionally, we describe the Generic Animat model, perform an analysis of the GA model, and finally describe our additions to the GA model. Chapter 3 contains our work methodology, introduction to the software framework and the design of our experiments. In chapter 4 we present the results of our experiments, which are then discussed in chapter 5. In chapter 5 we also discuss

the limitations and validity of the study, possible future directions for research, and touch upon the societal, ethical and ecological aspects regarding our work. And finally, in chapter 6 we present our conclusions.

2
Theory

In this chapter, the first three sections serve as a brief introduction to concepts used in this thesis regarding reinforcement learning, Q-learning and deep Q-learning. We then continue by describing an *animat* and *The Generic Animat model* as used in this thesis. The next section contains our analysis of The Generic Animat model, and the last section describes our additions to this model.

2.1 Reinforcement learning

The purpose of reinforcement learning is to learn, through interaction, how to behave in an unknown environment. Usually the interactions happen in discrete time steps t, where the learner (agent) selects an action $A_t \in \mathcal{A}$, based on a representation of the environment's state $S_t \in \mathcal{S}$. During the next time step $t+1$ the agent receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and a new state description $S_{t+1} \in \mathcal{S}$, this process is repeated until a terminal condition is fulfilled. In finite Markov decision processes (finite MDPs), the sets $\mathcal{A}$, $\mathcal{S}$ and $\mathcal{R}$ have a finite number of elements, and the random variables R_t and S_t are described by discrete probability distributions that are fully determined by the previous state and action [12]. In this report, we will study finite partially observable Markov decision processes (finite POMDPs), which means that the underlying process is a finite MDP but the agent only receives a partial description of the true state S_t at each time step.

2.2 Q-learning

There are a vast number of possible algorithms that can be used by the agent to learn from its interactions with the environment. A well-known algorithm is called *Q-learning*, which was introduced by Watkins [13]. In Q-learning the action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is updated with respect to the state-action pair (S_t, A_t), the reward R_{t+1} and the new state S_{t+1},

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right], \tag{2.1}$$

where $\alpha \in (0, 1]$ is called the learning rate and $\gamma \in [0, 1]$ is called the discount factor.

At each time step, the agent selects an action according to some policy $\pi(S_t)$. A common policy is called $\epsilon-$greedy [12], which selects a random action with probability ϵ and otherwise selects the action that maximises the action-value function $\max_a Q(S_t, a)$. In a finite MDP, the $\epsilon-$greedy policy ensures complete state space

exploration, which is crucial if the agents are supposed to find an optimal policy. An overview of Q-learning, using lazy initialisation, can be seen in Algorithm 1.

Algorithm 1: Q-learning with lazy initialisation.

Initialise $t = 0$, S_0, A_0 and $Q(S_0, a)$ for all $a \in \mathcal{A}$
repeat
  $t \leftarrow t + 1$
  Observe S_t and R_t
  if $Q(S_t, \cdot)$ *does not exist* **then**
    Initialise $Q(S_t, a)$ for all $a \in \mathcal{A}$
  end
  Update $Q(S_{t-1}, A_{t-1})$ according to update rule (2.1) using the tuple $(S_{t-1}, A_{t-1}, S_t, R_t)$
  Select action A_t according to some policy $\pi(S_t)$
  Perform action A_t
until S_t *is terminal*

2.3 Deep Q-learning

This section serves as an introductory overview regarding concepts in artificial neural networks and their application for reinforcement learning. For more in-depth information, see for example Hertz [14], Sutton and Barto [12], Goodfellow, Yoshua and Courville [15].

Considering that Q-learning uses a matrix $Q(S_t, A_t)$ it is evident that it won't scale well as the state space grows large. One possible workaround for this problem is to replace the Q-matrix with a function approximator such as an artificial neural network. In 2015, a team at Google Deepmind demonstrated how they used such an approach to train an agent to solve classic Atari 2600 games [16]. Their deep Q-network (DQN) agent made use of two novel ideas:

1. To make weight updates more stable, they used two networks: one main network M_{main}, and one target network M_{target}. The idea is that M_{target} should be slower to update, and thus provide a more stable target for training.

2. To reduce correlation between training samples, they used a replay memory RM with capacity N. This memory works as a queue and contains the last N state transitions, which can be randomly sampled to provide training data.

The main network M_{main} can be trained to output the Q-value $M_{main}(S_t, A_t) = Q(S_t, A_t)$. A policy such as $\epsilon-$greedy can be used to select which action to perform. One way to train a neural network is by using backpropagation [14]. For backpropagation to work, we need a way to measure the output error (loss function) in order to compute gradients. The loss function is defined as [16]

$$MSE(S_t, A_t, R_{t+1}, S_{t+1}) = \left[(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))^2 \right]. \quad (2.2)$$

To provide a more stable training target, the M_{target} weights need to update slower then M_{main} weights, this can be accomplished in many ways, and two possible choices are:

1. As Mnih et al. [16], by cloning the weights in M_{main} to M_{target} every C time steps.

2. As described in [17] and equation 2.3, by linear interpolation of the weights in M_{target} towards M_{main}.

$$w_{target_i} = w_{main_i}\tau + w_{target_i}(1 - \tau) \tag{2.3}$$

   where $i \in \{1...\text{number of network weights}\}$ and τ controls the convergence speed.

An overview of the DQN algorithm, using two networks, linear interpolation, and a replay memory, can be seen in Algorithm 2.

Algorithm 2: DQN-learning with target network and replay memory.

Initialise $t = 0$, S_0, A_0
Initialise M_{main}, M_{target} with random weights
Initialise replay memory RM to capacity N
repeat
  $t \leftarrow t + 1$
  Observe S_t and R_t
  Add transition $(S_{t-1}, A_{t-1}, S_t, R_t)$ to replay memory RM
  Uniformly sample a batch B of transitions from RM
  foreach $b \in B$ **do**
    Extract (s, a, s', r) from b
    Predict future Q-value. $Q_{future} = \max_a M_{target}(s', a)$
    Calculate target Q-value. $Q_{target} = r + \gamma Q_{future}$
    Train $M_{main}(s)$ towards output Q_{target}
  end
  According to equation 2.3, update weights for M_{target}
  Select action A_t, according to some policy $\pi(S_t)$ using $M_{main}(S_t)$
  Perform action A_t
until S_t *is terminal*

2.4 Animat

An animat is an artificial animal (contraction of animal-materials) and the word was coined by S.W. Wilson in 1985. A schematic description of an animat agent, as used in this thesis, is given by Figure 2.1.

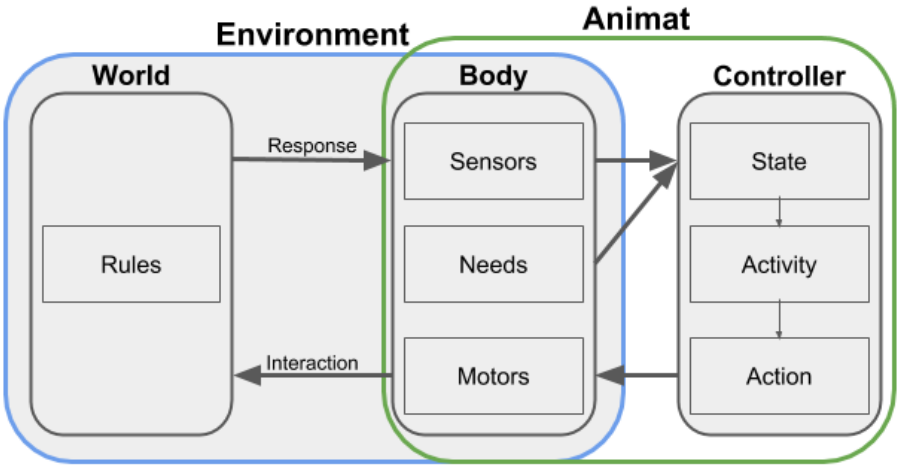


Figure 2.1: The animat and the environment in a reinforcement learning setting.

2.4.1 Body

The *body* is the animat's physical representation. The body has its associated finite sets of variables called *sensors*, *needs*, and *motors*. Sensors and motors take boolean values, whereas needs take values in the real interval $[0, 1]$.

Needs are denoted by natural numbers i. The status of need i at time t is the real value $\iota_i(t) \in [0, 1]$. Intuitively, 0 means death, while 1 means full need satisfaction. Examples of needs and their associated interoceptors could be water (osmoceptors); energy (insulin receptors); protein (amino acid receptors); oxygen (CO_2 receptors); integrity, i.e. freedom from pain (nociceptors); sleep (melatonin receptors); proximity (pheromone receptors); and reproduction (sexual hormone receptors).

Definition 1 (Rewards)**.** For each need i and time $t > 0$, the reward signal $r_i(t)$ is defined as follows:

$$r_i(t) = \iota_i(t) - \iota_i(t-1). \tag{2.4}$$

2.4.2 Controller

The controller is responsible for both learning and decision-making, intuitively, it models the animat's brain. The controller is a function that takes a (physiological) state consisting of sensor values and need values as input and outputs an action, which is immediately executed by the motors.

2.5 The Generic Animat model

In this section we present an overview of the Generic Animat model (GA model), proposed by Strannegård et al. [11], and a description of the subset of the GA model that is used in this thesis.

The controller of a GA agent consist of: a perception graph, a set of experience variables, rules for learning, and finally strategies for decision-making. The time

proceeds in discrete ticks and a GA agent is updated at each tick according to Algorithm 3.

Algorithm 3: The update sequence for a GA agent.

Input: A GA agent
while *Termination criteria not fulfilled* **do**
- The body receives a response from the environment and updates its sensors and needs accordingly
- The controller receives the active sensors and the status of the needs from the body
- The top active nodes are determined
- The global Q-values are determined
- The local Q-values and the reliability are updated
- Formation and forgetting rules are activated
- The top active nodes are determined again
- The global Q-values are determined again
- The action goodness and utility are determined
- An action is selected and sent to the body
- The action is performed by the body
- The world evaluates the interaction

end

2.5.1 Perception graph

In the Generic Animat model the perception graph is used to approximate the state based on the sensory input, an example of a perception graph is given in Figure 2.2.

Definition 2 (Perception graph)**.** A *perception graph* is a graph whose nodes are sensors and binary AND-gates. In this thesis, two nodes that form a conjunction are called the *predecessors* to that conjunction.

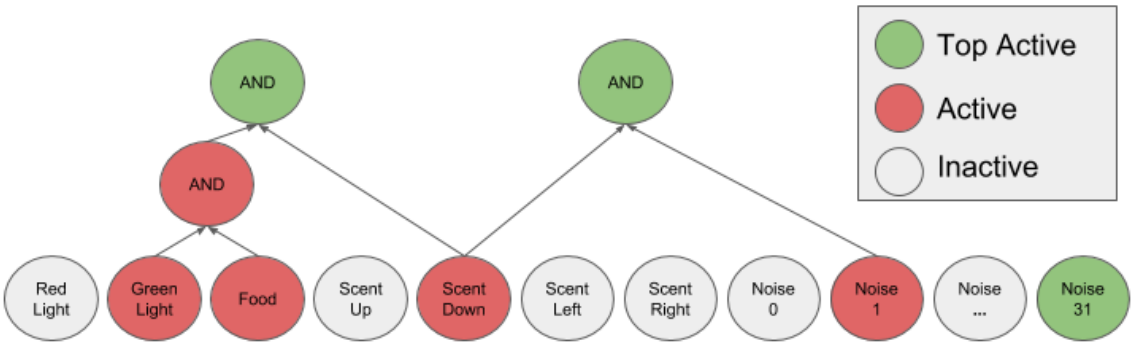


Figure 2.2: A perception graph with 5 active and 3 top active nodes. The lowest layer contains the sensors.

Definition 3 (Perception graph activity)**.** At each time step, the perception graph receives boolean values to its sensors. Those that receive the value *True* are called

active. This activity propagates via the AND-gates within the same tick. A conjunction is *active* if both of its predecessors are active.

We use symbol b for the nodes in the perception graph and B_t for the set of all nodes at time t. The set of all active nodes at time t is denoted by B_t^A.

Definition 4 (Top activity)**.** An active node $b \in B_t^A$ is *top active* if the set of sensors it represents is not a subset to a set of sensors represented by another active node $b' \in B_t^A$.

The set of all top active nodes at time t is denoted by B_t^{TA}. The set of top active nodes represents the state with respect to the structure of the perception graph.

2.5.2 Experience variables

The experience variables can be seen as the GA agents memory, which is based on past experiences.

Definition 5 (Local Q-values)**.** A local Q-value is a real-valued variable $Q_i(b, a)$ that reflects the expected response to need i when performing action a given that node b is top active.

Definition 6 (Reliability)**.** The reliability $Rel_i(b, a)$ is a measure of the stability of the entry $Q_i(b, a)$. It is defined as follows:

$$Rel_i(b, a) = \frac{1}{\sigma_i(b, a) + 1}. \tag{2.5}$$

Here $\sigma_i(b, a)$ is the standard deviation of $Q_i(b, a)$, calculated over the set of all previous values of $Q_i(b, a)$. The standard deviation is calculated using Welford's online algorithm [18].

Definition 7 (Global Q-values)**.** A global Q-value is a real-valued variable $Q_i^{global}(B_t^{TA}, a)$ that reflects the expected response to need i when performing action a given the set of top active nodes. It is defined as follows:

$$Q_i^{global}(B_t^{TA}, a) = \frac{\sum_{b \in B_t^{TA}} Q_i(b, a) Rel_i(b, a)}{\sum_{b \in B_t^{TA}} Rel_i(b, a)}. \tag{2.6}$$

Definition 8 (Surprised)**.** If $Q_i(b, a)$ is updated at $t + 1$ by at least the percentage $\phi_{Surprise}$, for all top active nodes $b \in B_t^{TA}$, the GA agent is *surprised*.

Definition 9 (Combination probability)**.** The combination probability $Comb(b, b')$ contains information about how likely it is that b and b' are top active together.

2.5.3 Learning rules

In this subsection, we present the update rule for the local Q-values and two formation rules that are used to create new nodes in the perception graph.

Definition 10 (Update local Q-values)**.** The update of the local Q-values is based on Q-learning, see Section 2.2, where the main differences stems from the state representation, the set of top active nodes. At $t+1$ the local Q-values are updated for all previous top active nodes $b \in B_t^{TA}$, with respect to the selected action a_t, the received rewards $r_i(t+1)$ and the new top active nodes $b' \in B_{t+1}^{TA}$, as follows:

$$Q_i(b, a_t) \leftarrow Q_i(b, a_t) + \alpha \left(r_i(t+1) + \gamma \max_a \left[Q_i^{global}(B_{t+1}^{TA}, a) \right] - Q_i(b, a_t) \right),$$

where $\alpha \in (0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discount rate.

Definition 11 (Probabilistic merge)**.** At each time step, flip a biased coin. If heads then select two nodes b and b' from B_t with probability proportional to the entries in $Comb(b, b')$ and such that $b'' = b$ AND b' does not yet exist. Then add b'' to B_t.

Definition 12 (Emotional merge)**.** If the GA agent is *surprised* at $t+1$, then select two top active nodes b and b' from B_t^{TA} and add $b'' = b$ AND b' to B_{t+1}.

2.5.4 Decision-making

In this section we present the building blocks that are used by a GA agent to select an action in a potentially multiobjective setting.

Definition 13 (Action goodness)**.** The *action goodness* is defined as

$$G_i(a, t) = \iota_i(t) + \omega Q_i^{global}(B_t^{TA}, a), \tag{2.7}$$

where $\omega \in [0, 1]$ is a constant.

Definition 14 (Utility)**.** The utility is defined as

$$utility(a, t) = \min_i \left[G_i(a, t) \right]. \tag{2.8}$$

Definition 15 (Select an action)**.** The policy is based on ϵ-greedy. Flip a biased coin, if heads then select the action that maximizes $utility(a, t)$, otherwise select a random action.

2.6 Generic Animat model analysis

After inspecting the reliability, the global Q-values and the local Q-value update rule, see Definitions 6, 7 and 10, we made three observations regarding the GA model: the state representation through the top active nodes, the influence of the reliability on the global Q-values, and the initial local Q-values for new nodes.

2.6.1 Top activity

Let us begin with a simple example to illustrate the problem where the local Q-values describe the quality of each node rather than the full state. Consider a GA agent that has the following three sensors: *apple*, an edible object; *fresh*, which is active if the apple is fresh; and *rotten*, which is active if the apple is rotten. Suppose the agent can perform two actions: *eat* and *idle*, and the reward function is defined as; the reward is $+1$ if the agent eats a fresh apple, -1 if it eats a rotten apple and 0 if the agent is *idle*. The optimal policy would be to eat fresh apples and be idle otherwise. The problem immediately stands out: the sensor *apple* will be top active at the same time as the sensors $fresh$ and *rotten*, which represent states with contradicting optimal behaviour. In this simple scenario it could then be impossible for the agent to find an optimal policy, even though the agent receives complete information about each state. This problem can be solved by forming a new node, either $apple \wedge fresh$ or $apple \wedge rotten$, which results in that for each top activity (state representation) there is a coherent response from the environment.

The observation regarding the top activity underlines both the problem and the solution: all sets of top activities (all state representations) a node b can be part of needs to have a coherent response from the environment for action $a \in A$, otherwise conflicting rewards and conflicting global Q-values could prevent the agent from learning a good policy. Three formation rules were designed with this in mind, see Definitions 21, 22 and 23.

2.6.2 Reliability

The reliability's effect on the global Q-values is evident: nodes with high reliability (stable local Q-values) have a more significant impact on the global Q-value. Intuitively the definition of the reliability might make sense, but the effect that the reliability has on the global Q-values is not consistent.

If the local Q-values are negative and the reliability is reduced, the likelihood that action a is chosen increases, but if the local Q-values are positive and the reliability is reduced, the likelihood that action a is chosen decreases. This discrepancy indicates that the reliability might need to be redefined so that it has a consistent effect regardless of the sign of the local Q-values.

The reliability depends on the set of corresponding local Q-values, but so does the local Q-values themselves, which leads to complex and circular dependencies. By using a constant reliability that is set to 1, the circular dependencies in the local Q-values and the discrepancy regarding the action selection are removed.

2.6.3 Initial local Q-values

When a new node is created it is possible to choose any values for its initial local Q-values and reliability values. Depending on the initial local Q-values and the initial reliability for new nodes, the policies, with respect to the affected sets of top activities, are affected accordingly. Ideally, the new node should not affect the current policy, but this is not a simple task since all affected top activities must be taken into account. For the initial local Q-values, two simple and computationally

efficient choices stand out as possible candidates: set to 0 and set to the average of the predecessors local Q-values. Both of those alternatives will affect the current policy, but by using the average of the predecessors it is possible that the effect is not as detrimental as when simply setting the local Q-values to 0. However, the effect will only be temporary no matter the choice of the initial local Q-values.

2.7 Additions to the Generic Animat model

In this section we present our proposed additions to the GA model. The new formation rules aim to create a state representation that make sure that all top activities receives a coherent response from the environment, with regards to the received reward.

2.7.1 Experience variables

The following experience variables are added to support the new formation rules.

Definition 16 (Pair reward)**.** $PairReward_i(b, b', a)$ is the probability that the reward for need i will be positive if action a is performed when b and b' are both top active. If the entry b, b', a have not yet received at least $\phi_{PositiveRewardMerge}$ positive rewards, $PairReward_i(b, b', a)$ is set to 0. At $t{+}1$ the update of $PairReward_i(b, b', a)$ is based on: $r_i(t+1)$, $b \in B_t^{TA}$, $b' \in B_t^{TA}$, the performed action a_t and the set of previous values.

Definition 17 (Reward history)**.** The reward history $RewardHistory_i(b, a)$ is a pair (pos, neg), where pos (neg) is the number of times a positive (negative) reward for need i has been received when doing action a while node b has been active. At $t+1$ the update of $RewardHistory_i(b, a)$ is based on: $r_i(t+1)$, $b \in B_t^A$ and the performed action a_t.

Definition 18 (Positive stable nodes)**.** Based on the entries in $RewardHistory_i(b, a)$, the positive stable nodes $PositiveStable_i(a)$ is a list of all nodes that have received at least $\phi_{PositiveStable}$ positive rewards and no negative rewards for need i and action a.

Definition 19 (Relevant nodes)**.** For each stable node $b \in PositiveStable_i(a)$ all nodes b' that seem to be correlated to b are added to the list of relevant nodes, $Relevant_i(b)$. If at least $\phi_{RelevantUpdates}$ updates have been performed for an entry in $RelevantTransition(b, b'|b'', a)$ and $RelevantTransition(b, b'|b'', a) > p_{Relevant}$, then b'' is added to $Relevant_i(b)$.

Definition 20 (Relevant transition probabilities)**.** The relevant transition probabilities, $RelevantTransition(b, b'|b'', a)$, contains the conditional probability that b' is active given that b'' was active and action a was performed,
where $b \in PositiveStable_i(a')$ and $b' \in Relevant_i(b)$. At $t+1$ the update of $RelevantTransition(b, b'|b'', a)$ is based on: $b' \in B_{t+1}^A$, $b'' \in B_t^A$, the performed action a_t and the set of previous values.

2.7.2 Learning rules

In contrast to the old formation rules, see Definitions 11 and 12, the new formation rules are based on the received reward rather than the local Q-values, and in addition to the top activity they also take the activity into account.

Definition 21 (Positive reward merge)**.** At each time step, flip a biased coin. If heads, then select two nodes b and b' with probability proportional to their entry $PairReward_i(b, b', a)$ and so that b and/or b' have received conflicting rewards, i.e. the entry in $RewardHistory_i(b, a)$ is $(> 0, > 0)$. Then if it does not yet exist, add $b'' = b$ AND b' to B_t.

The positive reward merge creates connections for nodes with conflicting rewards, with the goal that the new node becomes a positive stable node. Entries in $PairReward$ with high probability are more likely to be made first.

Definition 22 (Stable node merge)**.** Suppose a stable node $b \in PositiveStable_i(a)$ is active, $b \in B_t^A$. For $b' \in B_t^{TA}$, if it is not already represented, add $b'' = b$ AND b' to B_t.

The stable node merge make sure that all stable nodes receives a coherent response from the environment, i.e. we isolate them by forming new nodes with the top active nodes.

Definition 23 (Relevant node merge)**.** Suppose a relevant node $b' \in Relevant_i(b)$, is active, $b' \in B_t^A$. For $b' \in B_t^{TA}$, if it is not already represented, add $b'' = b$ AND b' to B_t.

Similar to stable node merge, we also isolate nodes deemed relevant to a stable node.

3
Method

In this chapter, we initially present our work methodology and give an overview of the functionality of the software framework we developed. Then we describe the design and the rules of the animat environment, which are used in two different experiments. Furthermore, we explain why and how we gather statistics during the experiments, and the final two sections describe the configurations of our experiments.

3.1 Work methodology

Since we are two people working together in this thesis, and we have the intention of open-sourcing our work, we chose to follow iterative, lean, agile approaches towards software development. At first, we used a top-down approach [19], where we identified the major modules of the framework and specified their interfaces, this allowed us to get a prototype up and running quickly, and help confirm that the planned development environment and core packages worked well together. Once we established this common ground, we could then proceed to work in parallel when implementing, testing, and deploying the core algorithms.

For actual development, we followed the lean methodology kanban, which was developed by Toyota in the 1970's [20]. The same guiding principles that exist in kanban can also be applied to software development [21]. Problem-solving was approached using the divide and conquer mindset which is very common in software development [22].

For planning and visualisation of tasks and tracking progress we used the online tool Trello (`http://www.trello.com`), and for managing the source code and revision control we used Git (`http://www.git.com`).

3.2 Framework

The framework is developed with the intention to enable easy implementation and evaluation of animat based agents and environments, a more detailed description and installation instructions can be found in appendix A. For this thesis the framework enabled us to: (i) implement and test new environments, (ii) implement and test new agents, (iii) evaluate changes, (iv) collect and present statistics, (v) debug and monitor agents as they learned.

3.3 Environment for the experiments

The environment that will be used by both experiments is presented in this section. The environment is designed so that conflicting rewards are received if the correct nodes have not been created. To be able to increase the state space, we have added a configurable amount of noise to the environment.

3.3.1 World

The environment is built as a bounded 3x3 grid, populated by an agent (cat) and edible food objects (fish), see Figure 3.1. The rules take inspiration from Skinner's theories and implementation of the *Skinner Box*, where an action can generate either reward or punishment but is consistent given observations in the environment [23]. For this environment, a green light indicates that the food is safe to consume while a red light indicates the opposite. For the agent to find an optimal policy, it must learn to only consume food when the green light is active. To increase the state space there are also lights that represent noise, these are activated randomly and do not affect the received reward. The optimal policy is thus straightforward, but the problem lies in finding this policy with all the noise present.

Rules

The rules of the world will now be listed:

- Initially the agent along with three edible objects are placed in separate random locations.
- The edible objects are stationary, and at most there can only be one edible object per tile.
- The agent can move one tile each time step in the directions up, down, left and right.
- The agent can consume an edible object if they are on the same tile and the agent performs the action *eat*.
- When all edible objects are consumed, they are all placed at new random unoccupied locations.
- The red light is activated with probability $p_{red} = 0.25$. If the red light is not activated the green light is activated instead.
- There are N_{noise} noise lights that are activated independently of each other with probability $p_{noise} = 0.25$.
- The agent's energy decays each time step by a constant rate of -0.02.
- If the agent consumes an edible object when the green light is active, its energy increases by 0.3.

- If the agent consumes an edible object when the red light is active, its energy decreases by -0.3.
- If the agent performs action *eat* at an empty tile, the energy decreases by -0.015.
- All movement decreases the energy by -0.01.

Figure 3.1: The figure shows a snapshot of the environment configured for experiment 2. The cat represents the agent, and the fish represent the edible food objects. In the order presented the top bar shows the activity of: the red light, which is inactive; the green light, which is active; and 32 lights representing noise, where four are active. Because of the noise, the state space of this world is greater than 2^{32}.

3.3.2 Body

Sensors

The *body* has the following sensors: a green light sensor, a red light sensor, N_{noise} noise light sensors, a food sensor and four remote perception sensors for food smell. The food sensor is active if the body stands on the same tile as a food object.

We say that the food produces a smell, the combined intensity of the smells from different locations is used to find the direction towards the strongest scent. The strength of the smell, is also attenuated with distance according to the inverse square law [24, p. 726]. So the combined smell of all food f is calculated for each offset position $L \in \{\text{up}, \text{down}, \text{left}, \text{right}\}$ as follows:

$$s(L, f) = \sum_{j=1}^{N} \frac{I_{f_j}}{1 + d(L, f_j)^2}, \tag{3.1}$$

where f_j is a tile with food, N the total number of food currently in the world, $d(L, f_j)$ the Euclidean distance from position L to the location of food f_j and $I_{f_j} = 1$ the scent intensity at the source. The sensor corresponding to the offset position with the highest intensity $s(L, f)$ is set as active.

Needs

An agent has one need: energy. Since the energy is not only affected by the actions but also the constant decay rate, an agent that is not following a good policy can possibly see its energy reach 0 within 40 time steps.

Motors and actions

An agent can perform five different actions: *move up*, *move down*, *move left*, *move right* and *eat*.

3.4 Policy evaluation

To evaluate an agents policy, in the environment described in Section 3.3, we decided to measure: the energy, the episode length, the number of consumed food while the green light was active (green food) and the number of consumed food while the red light was active (red food). Additionally, for the Generic Animat agents, we measure: the number of nodes, the stable nodes and the relevant nodes.

For each agent the data is collected over N_{eval} independent evaluations, each evaluation consist of $N_{episodes}$ succeeding episodes. An episode starts with a training phase that last for $t_{training}$ time steps, learning and exploration are turned on during this phase. A training phase is always followed by a testing phase, where learning and exploration are turned off. The testing phase is evaluated over $N_{testing}$ runs, where each run ends after $t_{testing}$ time steps or if the agent's energy reaches 0. A detailed description of how the data is generated can be seen in Algorithm 4.

3.5 Experiment 1: Generic Animat model

This experiment has been designed to evaluate the new formation rules, the old formation rules, the reliability and the initial local Q-values. The environment described in section 3.3 will be used with $N_{noise} = 10$ noise lights. In total, eight agents will be tested separately, four of them will be using the new formation rules and the remaining four agents will use the old formation rules. Two versions of the reliability will be studied: as described in definition 6 and always set to one. For new nodes, two versions of the initial local Q-values will be assessed: set to zero or set to the average of its predecessors values. All eight agents have the following learning parameters in common: $\epsilon_{start} = 1.0$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.1$, $\alpha = 0.05$, $\gamma = 0.9$ and $\omega = 1.0$.

The data will be collected as described in Section 3.4, using the following parameters: $N_{eval} = 200$, $N_{episodes} = 50$, $t_{training} = 200$, $N_{testing} = 20$ and $t_{testing} = 100$. After each training phase the number of nodes will be measured for all agents, and

Algorithm 4: An overview of how data is collected during the evaluation of one agent. The example statistics s_{train} and s_{test}, are included to show how statistics that are based on the training and the testing phase are gathered, respectively.

Initialise the environment and the agent.
For each statistic, i.e. $s \in \{s_{train}, s_{test}\}$, initialise a matrix, $\mathbf{A}^s$, with dimension $N_{eval} \times N_{episodes}$.
For each statistic that is based on the test episodes, i.e. s_{test}, initialise vectors, $\mathbf{b}^s$, of length $N_{testing}$.
for *i=1,2,...,*N_{eval} **do**
  Reset the environment (place the agent and the food at new random locations and set the agents energy to one ($\iota_{energy}(0) = 1$)).
  Reset the agent (remove all new nodes and set the local Q-values for the sensors to zero).
  for *j=1,2,...,*$N_{episodes}$ **do**
    Reset the environment.
    Turn on learning and exploration.
    for $t_{training}$ *time steps* **do**
      The agent learns.
    end
    Store data in $A^s_{i,j}$, for the statistics that are based on the training episodes, i.e. s_{train}.
    Turn off learning and exploration.
    Clear the vectors $\mathbf{b}^s$.
    for *k=1,2,...,*$N_{testing}$ **do**
      Reset the environment.
      $t = 0$
      while $t < t_{testing}$ *and* $\iota_{energy}(t) > 0$ **do**
        $t \leftarrow t + 1$
        The agent exploits its current policy.
      end
      Store data in b^s_k, for the statistics that are based on the test episodes, i.e. s_{test}.
    end
    Calculate averages over the entries in the vectors $\mathbf{b}^s$ and store them in $A^s_{i,j}$.
  end
  For each statistic s and episode j, use the entries in $\mathbf{A}^s$ to calculate the mean and standard deviation over all evaluations $i \in \{1, 2, ..., N_{eval}\}$ and store them in μ^s_j and σ^s_j, respectively.
  Plot $\mu^s_j \pm \sigma^s_j$ over the total number of training steps $j \cdot t_{training}$.
end

for the New Animat agents, the stable nodes and the relevant nodes will be gathered as well. Two statistics will be gathered after each testing phase: the energy, and the number of consumed food while the red light was active (red food).

New Animats

An agent that exclusively uses the new formation rules, see Definitions 21, 22 and 23, will be called *New Animat*. The parameters for the new formation rules are set to: $\phi_{PositiveStable} = 20$, $\phi_{RelevantUpdates} = 20$, $p_{Relevant} = 0.90$ and $\phi_{PositiveRewardMerge} = 10$. The difference between the New Animat agents lies with the configuration of the initial local Q-values and the definition of the reliability. In Table 3.1, the configuration for each agent can be seen.

Agent	Reliability	Initial $Q_i(b, a)$
New Animat 1	Always one	$\frac{1}{2}\sum_{b' \in \{b_1, b_2\}} Q_i(b', a)$
New Animat 2	As Def. 6	$\frac{1}{2}\sum_{b' \in \{b_1, b_2\}} Q_i(b', a)$
New Animat 3	Always one	0
New Animat 4	As Def. 6	0

Table 3.1: The configuration of the reliability and the initial local Q-values for each agent that uses the new formation rules in the first experiment. Here b_1 and b_2 are the predecessors to a new node b.

Old Animats

An agent that exclusively uses the old formation rules, see Definitions 11 and 12, will be called *Old Animat*. The parameters for the old formation rules are set to: $\phi_{Surprise} = 0.05$ and the probability for the *probabilistic merge* will be $p = 0.01$. The configuration of the reliability and the initial local Q-values, for each Old Animat agent, can be seen in Table 3.2.

Agent	Reliability	Initial $Q_i(b, a)$
Old Animat 1	Always one	$\frac{1}{2}\sum_{b' \in \{b_1, b_2\}} Q_i(b', a)$
Old Animat 2	As Def. 6	$\frac{1}{2}\sum_{b' \in \{b_1, b_2\}} Q_i(b', a)$
Old Animat 3	Always one	0
Old Animat 4	As Def. 6	0

Table 3.2: The configuration of the reliability and the initial local Q-values for each agent that uses the old formation rules in the first experiment. Here b_1 and b_2 are the predecessors to a new node b.

3.6 Experiment 2: Benchmark

Five agents are compared to each other in this experiment: New Animat, Old Animat, DQN, Q-learner and Random. The environment described in section 3.3 is used with $N_{noise} = 32$ noise lights.

The data will be collected as described in section 3.4, with the following parameters: $N_{exp} = 20$, $N_{episodes} = 50$, $t_{training} = 200$, $N_{testing} = 20$ and $t_{testing} = 100$. If applicable for a specific agent, we will after each training phase record the number of nodes in its perception graph, or in the Q-learner's case, the number of rows in its Q-matrix (states visited). After each test phase, we gather four different statistics: the energy level, the episode length, the amount of food consumed while the red light was active (red food), and the amount of food consumed while the green light was active (green food). Additionally, for one of the New Animat agents, a log containing when and by which rule nodes were created is saved for later analysis.

New Animat

The New Animat agent exclusively use the new formation rules, see Definitions 21, 22 and 23. The parameters for the new formation rules are set to: $\phi_{PositiveRewardMerge} = 10$, $\phi_{PositiveStable} = 15$, $\phi_{RelevantUpdates} = 20$, $p_{Relevant} = 0.65$. And the New Animat agent uses the following learning parameters: $\epsilon_{start} = 1.0$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.01$, $\alpha = 0.05$, $\gamma = 0.9$ and $\omega = 1.0$. The initial local Q-values are set to the average of its predecessors and the reliability is always 1.

Old Animat

The Old Animat agent exclusively use the old formation rules, see Definitions 11 and 12. The parameters for the old formation rules are set to: $\phi_{Surprise} = 0.04$ and the probability for the *probabilistic merge* will be $p = 0.02$. And the agent is configured with the learning parameters: $\epsilon_{start} = 1.0$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.01$, $\alpha = 0.05$, $\gamma = 0.9$ and $\omega = 1.0$. The initial local Q-values are set to 0 and the reliability as described by Definition 6.

DQN

The DQN agent replaces the Q-matrix by a function approximator in the form of a neural network, as described in Section 2.3 and by Algorithm 2. It is implemented on top of `Keras` [25] and configured to use `TensorFlow` [26] as backend.

The optimal policy in this environment, i.e. which action to take for a given observed state is straightforward, since the actions giving a positive reward are linearly separable. We can therefore use a simple topology for the network consisting of just one input layer and one output layer. The hyperparameters used were found through repeated manual trials and chosen to provide a good balance between training time, stability and performance. There was no exhaustive grid search done to try and find the optimal parameters, and it is therefore very likely that better parameter settings exist for this experiment. In the end, we settled on the following parameters: *Replay memory capacity = 1000*, *Replay batch size = 32*, *Optimizer = Adam*, *Activation = Sigmoid*, *Loss function = MSE*, $\epsilon_{start} = 1.0$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.01$, $\alpha = 0.005$, $\gamma = 0.9$ and $\tau = 0.1$.

In total, this two-layered network has 200 weights to train: The input layer with 39 input features+bias, and an output layer with five outputs. For a more in-depth explanation of the parameters, please see the

`Keras documentation` (`https://keras.io`), and the source code accompanying this thesis (`https://gitlab.com/fredrikma/aaa_survivability.git`).

Q-learner

The Q-learner agent is based on ordinary Q-learning, where each unique set of active sensors has an entry in the Q-matrix. Since the state space is vast, it is implemented using lazy initialisation according to Algorithm 1. It uses the following learning parameters: $\epsilon_{start} = 1.0$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.01$, $\alpha = 0.05$, $\gamma = 0.9$.

Random

This agent will select an action randomly at each time step.

4
Result

In this chapter, we present the results of our two experiments. The first experiment investigates the performance of the New Animat agents and the Old Animat agents, using different configurations of the reliability and the initial local Q-values. The second experiment compares a New Animat agent and an Old Animat agent to a deep Q-learning agent (DQN), a standard Q-learning agent (Q-learner) and an agent (Random) that selects actions randomly.

4.1 Experiment 1: Generic Animat model

In this section we present the results of the first experiment. The New Animat agents and the Old Animat agents are compared separately, where the mean and standard deviation of each statistic is shown in a plot. The order of the configurations of the reliability and the initial local Q-values are the same for the New Animat agents and the Old Animat agents, see Table 3.1 and 3.2, i.e. the only difference between New Animat 1 and Old Animat 1 is the set of formation rules that are being used.

4.1.1 Energy

The energy can be seen in Figure 4.1. After $\sim$ 1000 trained time steps, all of the New Animat agents have a higher energy compared to all of the Old Animat agents. Initially the agents 1 and 2 have a higher energy compared to the agents 3 and 4, additionally there are no noticeable differences when comparing agents 1 to 2 and 3 to 4, when looking at either the New Animat agents or the Old Animat agents.

From Sub-figure 4.1a we see that New Animat 1 and New Animat 2 stabilises at $\sim$ 0.94 after 3000 trained time steps. New Animat 3 and New Animat 4 converge towards $\sim$ 0.94 after 4000 time steps, but, compared to New Animat 1 and New Animat 2, they have a larger standard deviation throughout this time frame.

After $\sim$ 3000 time steps all of the Old Animat agents starts to perform similar to each other, where the energy increases slowly and reaches $\sim$ 0.74 at the end of this time frame, see Sub-figure 4.1b.

4.1.2 Number of nodes

In Figure 4.2 the number of nodes can be seen. All of the New Animat agents stabilise around 220 nodes, which is about half of the nodes that are formed by the Old Animat agent at the end of the time frame. There are no noticeable differences

when comparing agents 1 to 2 and 3 to 4, when looking at either the New Animat agents or the Old Animat agents.

From Sub-figure 4.2a we see that after $\sim$ 1500 time steps, New Animat 1 and New Animat 2 create new nodes at a faster pace compared to New Animat 3 and New Animat 4. After 4000 time steps all of the New Animat agents have stabilised around 220 nodes, but the standard deviation has not converged to zero for any of them.

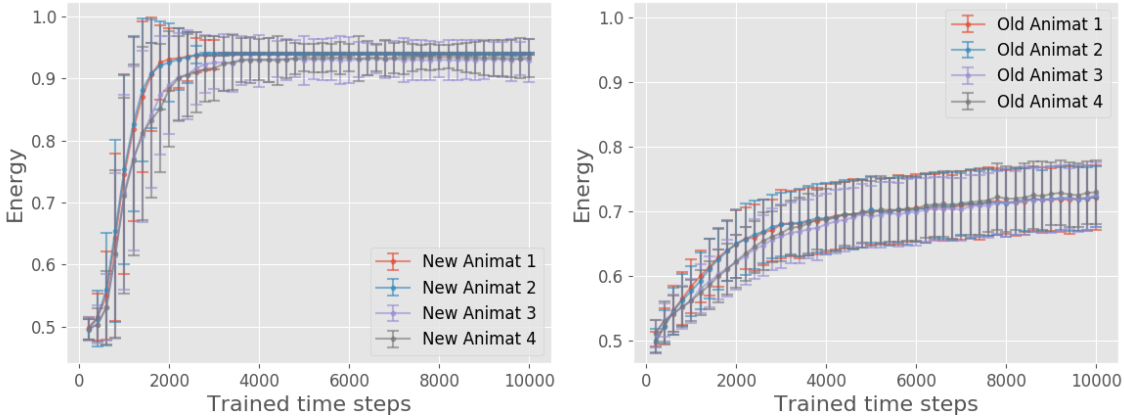
After 2000 time steps, Old Animat 3 and Old Animat 4 starts to create more nodes compared to Old Animat 1 and Old Animat 2. The number of nodes increases steadily for all Old Animat agents, at the end of this time frame New Animat 1 and New Animat 2 have $\sim$450 nodes and New Animat 3 and New Animat 4 have $\sim$ 490 nodes.

4.1.3 Red food

The number of consumed food, while the red light was active (red food), can be seen in Figure 4.3. All New Animat agents have stopped consuming red food after 4000 steps but none of the Old Animats have stopped consuming red food within the time frame. Initially the agents 1 and 2 consume more red food compared to the agents 3 and 4, additionally there are no noticeable differences when comparing agents 1 to 2 and 3 to 4, when looking at either the New Animat agents or the Old Animat agents. Notice that for New Animat 3 and 4 the red food decreases the first 1000 steps but then starts to increase again before finally stabilising at zero. After 2000 time steps all of the Old Animat agents starts to perform similar to each other, at the end of the time frame they consume $\sim$ 0.25 red food.

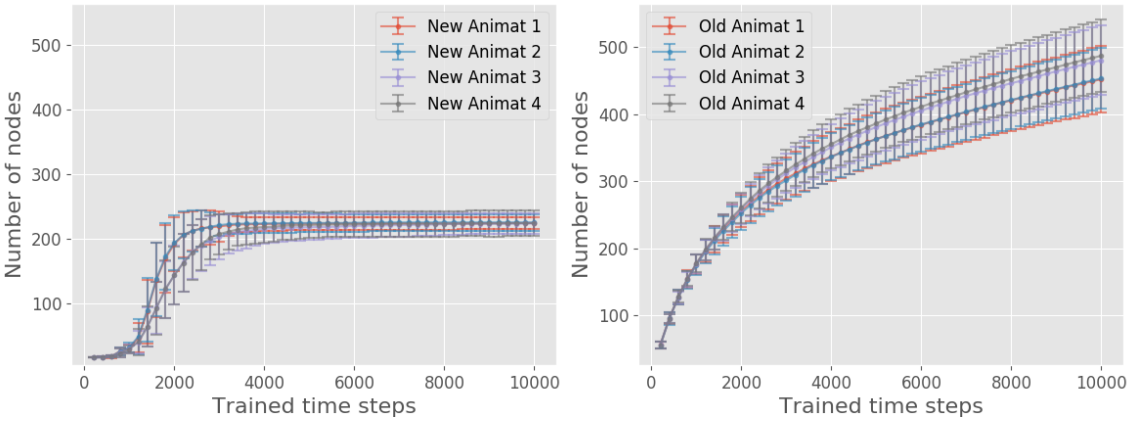
4.1.4 Stable and relevant nodes

In the Figures 4.4 and 4.5 the stable and relevant nodes can be seen. A fraction with mean 1 and standard deviation 0, means that the node was present in all independent evaluations for that agent. For all agents the stable node *Green Light* $\wedge$ *Center FOOD* converges to one with zero standard deviation within the first 3000 time steps. The nodes *Center FOOD* and *Green Light* eventually converges to 1 with zero standard deviation for all agents as well. The relevant nodes are added at a faster pace for New Animat 1 and New Animat 2 compared to New Animat 3 and New Animat 4. The remote perception nodes are not always added for any of the agents. There are no noise nodes present for any agent. The *Red Light* node is added to the set of relevant nodes for all agents, but it is with a mean that is less than 0.1.



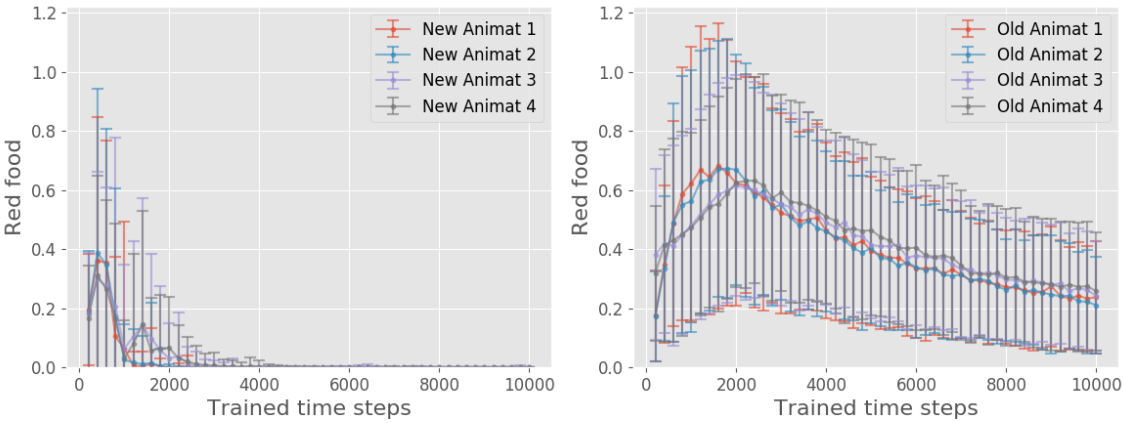
(a) The New Animat agents. **(b)** The Old Animat agents.

Figure 4.1: The mean energy along with the standard deviation is presented for the New Animat agents and the Old Animat agents. After $\sim$ 1000 time steps all of the New Animat agents have a higher energy compared to all of the Old Animat agents. Notice that, within each type of agent, the energy is similar when comparing agent 1 to agent 2 and agent 3 to agent 4. New Animat 1 and New Animat 2 stabilise at $\sim$ 0.94 after approximately 3000 steps, New Animat 3 and New Animat 4 reaches a similar mean but they have a larger standard deviation. Initially Old Animat 1 and Old Animat 2 have a slightly larger energy compared to Old Animat 3 and Old Animat 4. After $\sim$3000 time steps the Old Animat agents perform similar to each other, where the energy increases slowly and ends up at $\sim$ 0.74. The configurations of the agents can be seen in Table 3.1 and 3.2.



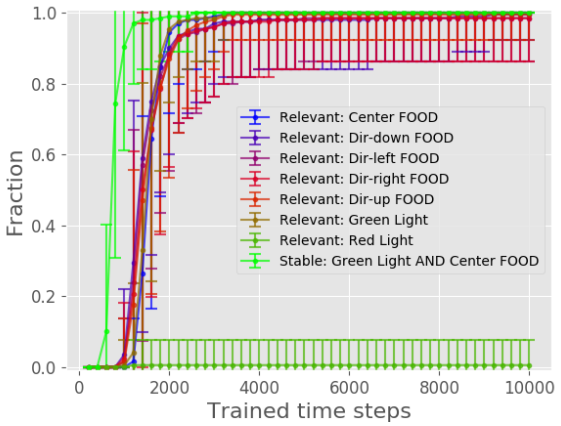
(a) The New Animat agents. **(b)** The Old Animat agents.

Figure 4.2: The mean number of nodes along with the standard deviation is presented for the New and Old Animat agents. All of the New Animat agents stabilise around 220 nodes, but the standard deviation does not converge to zero for any of them. There are no noticeable differences when comparing agents 1 to 2 and 3 to 4, when looking at either the New Animat agents or the Old Animat agents. For all Old Animat agents the number of nodes steadily increases, Old Animat 1 and Old Animat 2 reaches $\sim$ 450 nodes, and Old Animat 3 and Old Animat 4 reaches $\sim$ 490 nodes after 10000 time steps. The configurations of the agents can be seen in Table 3.1 and 3.2.

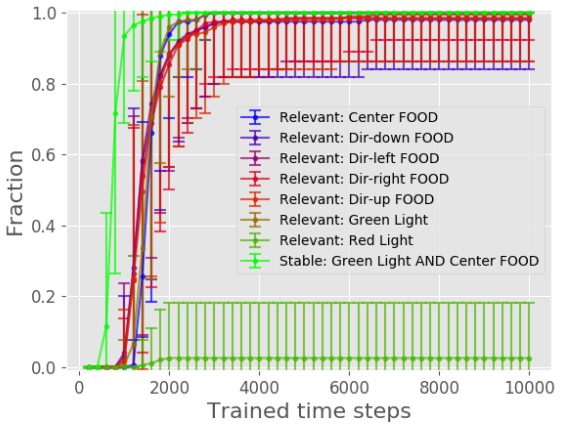


(a) The New Animat agents. **(b)** The Old Animat agents.

Figure 4.3: The mean red food along with the standard deviation is presented for the New and Old Animat agents. All New Animat agents have stopped consuming red food after 4000 time steps, but the Old Animat agents do not stop consuming red food during this time frame. Initially the agents 1 and 2 consume more red food compared to the agents 3 and 4, additionally there are no noticeable differences when comparing agents 1 to 2 and 3 to 4, when looking at either the New Animat agents or the Old Animat agents. Notice that for New Animat 3 and 4 the red food temporarily increases after 1000 time steps. After 2000 time steps all of the Old Animat agents start to perform similar to each other, at the end of the time frame they consume ~ 0.25 red food. The configurations of the agents can be seen in Table 3.1 and 3.2.

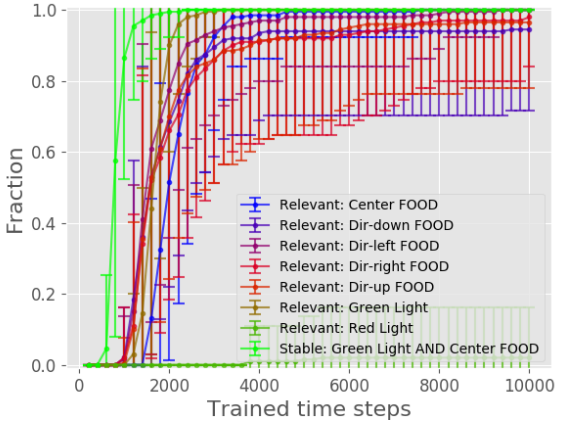


(a) New Animat 1.

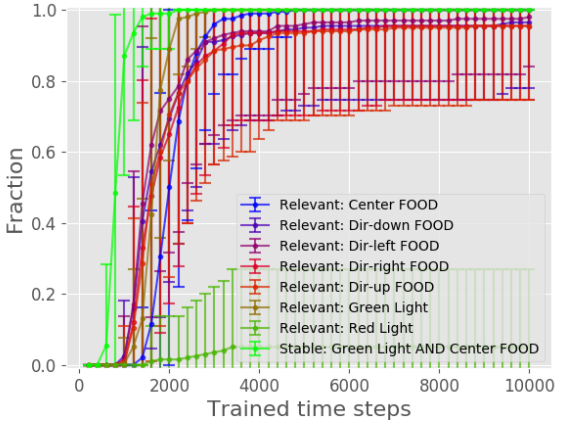


(b) New Animat 2.

Figure 4.4: The mean and standard deviation of the fraction of stable and relevant nodes for New Animat 1 and New Animat 2, if a node is added by all independent evaluations the mean is one and the standard deviation is zero. There is no significant difference between the two agents. The stable node, *Center FOOD* and *Green Light* reaches a fraction of 1 with zero standard deviation after 3000 time steps. The remote perception nodes are not always added and the *Red Light* node is added to less than 5% of the evaluations. The configurations of the agents can be seen in Table 3.1



(a) New Animat 3.



(b) New Animat 4.

Figure 4.5: The mean and standard deviation of the fraction of stable and relevant nodes for New Animat 3 and New Animat 4, if a node is added by all independent evaluations the mean is one and the standard deviation is zero. There is no significant difference between the two agents. The stable node reaches a fraction of 1 with zero standard deviation after 3000 time steps, both *Center FOOD* and *Green Light* are eventually always added as well. The remote perception nodes are not always added and the *Red Light* node is added to less than 10% of the evaluations. The configurations of the agents can be seen in Table 3.1

4.2 Experiment 2: Benchmark

In this section, we present the results of the second experiment. For each of the five tested agents, we show the mean and standard deviation of each statistic in a plot. For one of the New Animat agents, we also show a partial log over its node formation, see Listing 4.1.

4.2.1 Energy

Figure 4.6 shows the energy as a function of the number of trained time steps. Here it is clear that New Animat and DQN reach a higher energy level compared to the other agents. The energy for DQN increases quickly and is higher than New Animat during the first 2000 time steps, but then New Animat stabilises at a level that DQN is unable to reach in this time frame. Old Animat's energy is slowly improving over time, and it performs better than both Q-learner and Random. However, it fails to reach the performance levels of New Animat or DQN. Q-learner and Random show similar energy performance with no noticeable improvement in this time frame.

4.2.2 Number of nodes

Figure 4.7 shows two different statistics: the total number of nodes created by New Animat and Old Animat, and the total number of unique states visited by the Q-learner. New Animat's node count stabilises around 700 nodes after 3000 time steps. The number of nodes for Old Animat increases steadily, and at the end of this time frame, Old Animat has roughly twice as many nodes as New Animat. The number of unique states visited by the Q-learner increases linearly with the number of time steps.

4.2.3 Episode length

In Figure 4.8 the episode length is shown. DQN has an episode length close to 100 after 1000 time steps, and after 2000 time steps so does New Animat. Even though DQN quickly approaches a mean episode length of 100, the standard deviation does not stabilise to zero during this time frame. On the other hand, the standard deviation for New Animat reaches zero as soon as the mean reaches 100. The episode length increases steadily over time for Old Animat, but it never reaches the same level as New Animat or DQN. Q-learner and Random show no noticeable improvement in this time frame.

4.2.4 Green food

Figure 4.9 shows how much food the agents consumed while the green light was active (green food). Consuming green food is the only way to gain energy, and is therefore an important performance measure. Initially, DQN consumes more green food compared to New Animat but after $\sim$1800 time steps New Animat starts to outperform DQN with regards to both number of consumed green food and stability.

Old Animat show an increase in the number of green food consumed with time and performs better than both Q-learner and Random agent. However, it fails to reach the performance levels of New Animat and DQN. Q-learner and Random agent show similar performance with no noticeable improvement in this time frame.

4.2.5 Red food

Figure 4.10 shows the amount of food the agents consumed while the red light was active (red food). One crucial part of a good policy is to avoid eating food while the red light is active, and here it is clear that both New Animat and DQN stops eating red food after ∼1000 time steps. Q-learner and Random are consistently eating the same amount of red food, while Old Animat consumes an increasing amount of red food during this time frame.

4.2.6 Node formation

For one of the New Animat agents in the experiment, Listing 4.1 shows a partial log for which point in time nodes were created and the specific rule used.

```
t = 585  : Formed AND_1  : 'Green Light' AND 'Center FOOD' : (Reward based merge)
t = 745  : Formed AND_2  : 'AND_1' AND 'Dir-left FOOD'     : (Reward based merge)
t = 869  : Formed AND_3  : 'AND_1' AND 'Noise 25'          : (Reward based merge)
... (Reward based merges) ...
t = 979  : Formed AND_9  : 'AND_1' AND 'Noise 7'           : (Stable node merge)
t = 979  : Formed AND_10 : 'AND_1' AND 'Noise 18'          : (Stable node merge)
t = 979  : Formed AND_11 : 'AND_1' AND 'Dir-up FOOD'       : (Stable node merge)
t = 979  : Formed AND_12 : 'AND_1' AND 'Noise 14'          : (Stable node merge)
t = 979  : Formed AND_13 : 'AND_1' AND 'Noise 28'          : (Stable node merge)
... (Stable node merges) ...
t = 1337 : Formed AND_39 : 'AND_1' AND 'AND_7'             : (Stable node merge)
t = 1749 : Formed AND_40 : 'Dir-right FOOD' AND 'AND_32'   : (Relevant node merge)
t = 1749 : Formed AND_41 : 'Dir-right FOOD' AND 'AND_24'   : (Relevant node merge)
t = 1749 : Formed AND_42 : 'Dir-right FOOD' AND 'AND_19'   : (Relevant node merge)
... (Alternating: Stable node merges followed by Relevant node merges) ...
t = 9276 : Formed AND_648: 'Center FOOD' AND 'AND_647'     : (Relevant node merge)
```

Listing 4.1: For one of the New Animat agents, this listing shows the timing for when the three active node formation rules are invoked. At time = 585 the rule for positive reward (definition 21) is activated and forms the node
Green Light ∧ *Center FOOD*. At time = 979 this node has been identified as stable, and the rule for stable node merge (definition 22) is activated. At time = 1749 the first relevant node is identified, and the rule for relevant node merge (definition 23) is activated. Between time = 1749 and time = 9276, we then see alternating usage of the rules for stable node merge and relevant node merge.

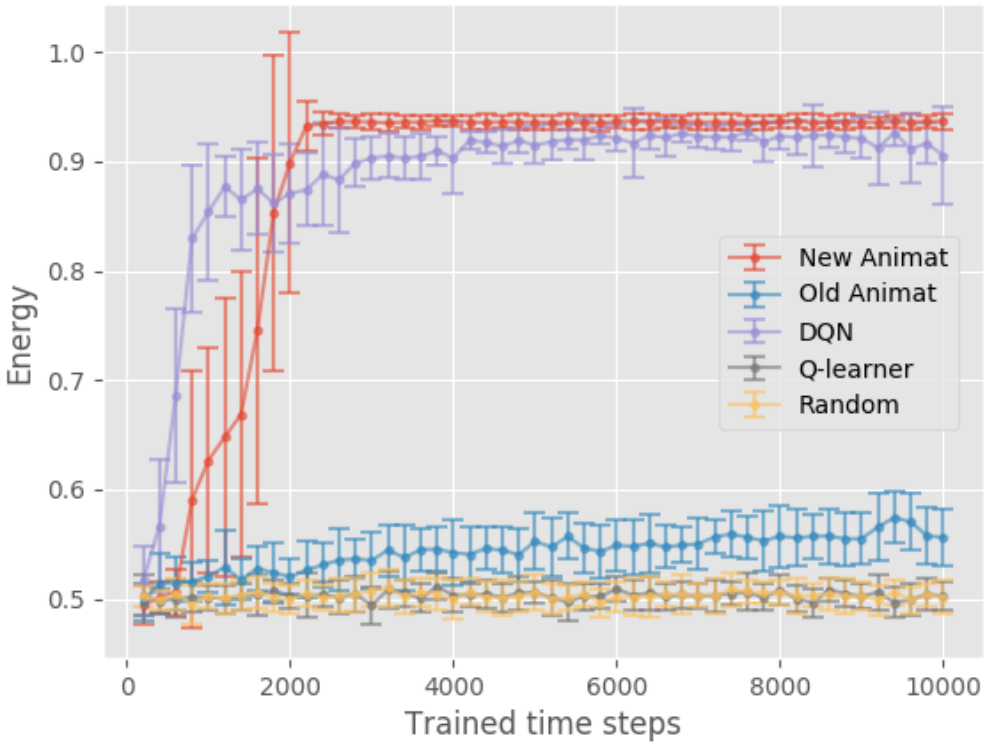


Figure 4.6: For each agent, the mean and standard deviation of the energy with respect to the number of trained steps can be seen. New Animat and DQN have the highest performance, and Old Animat performs slightly better then Q-learner and Random. Q-learner and Random have similar performance to each other in this time frame. For DQN the energy increases rapidly from the very beginning but is soon overtaken by New Animat's performance in this time frame. For New Animat the energy increases rapidly after $\sim$ 1000 steps and then stabilises after $\sim$ 2200 steps.

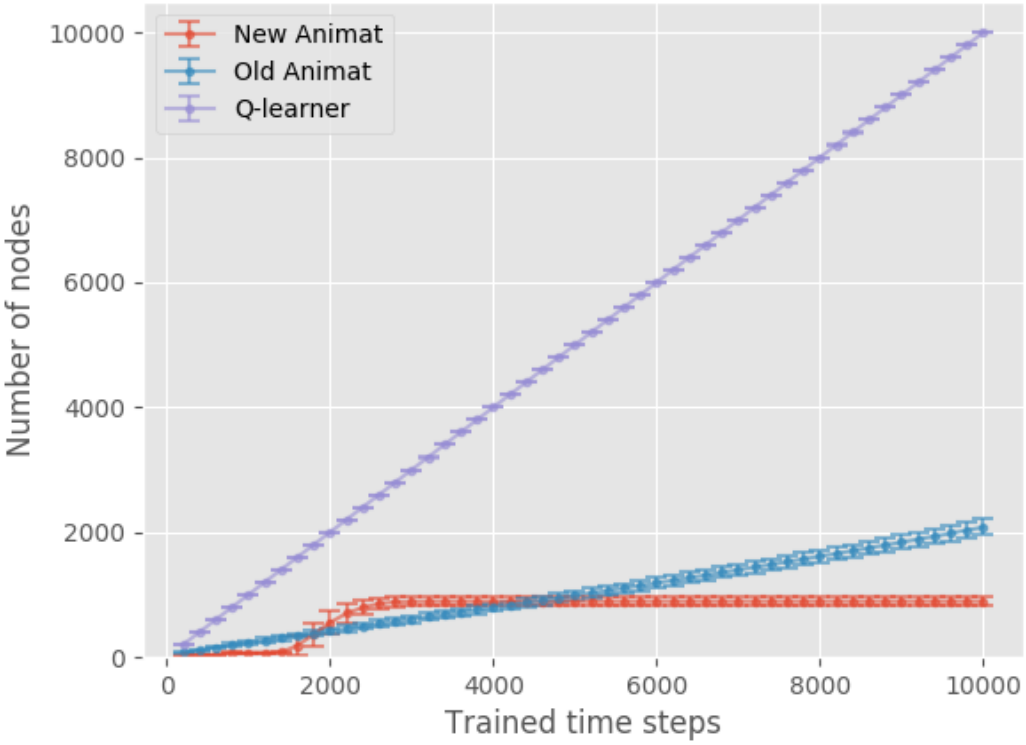


Figure 4.7: The mean and standard deviation for the total number of nodes for New Animat and Old Animat, and the total number of unique states for Q-learner, with respect to the number of trained steps. New Animat stabilises around ~ 700 nodes, whereas the number of nodes keeps increasing for Old Animat. At the end of this time frame, Old Animat has created twice as many nodes as New Animat. The number of unique states for Q-learner increases linearly with the number of trained time steps.

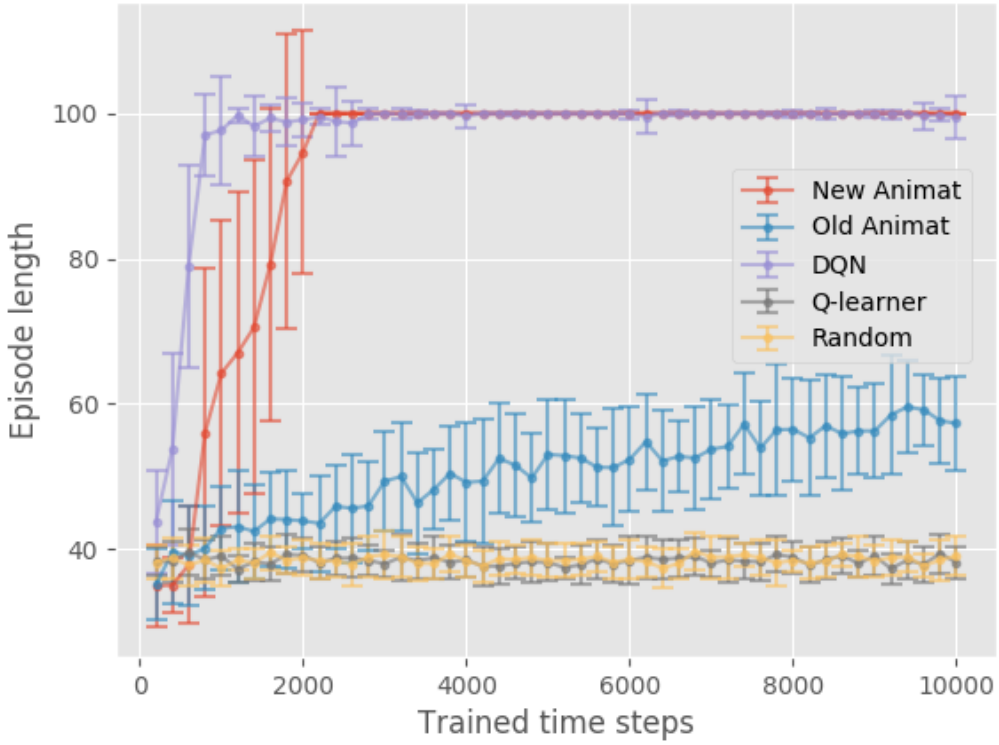


Figure 4.8: The mean and standard deviation of the episode length with respect to the number of training steps. Both New Animat and DQN finds a policy that enables them to survive for the complete duration of a test episode, which is 100 time steps. After $\sim$ 2200 steps the mean reaches 100 and the standard deviation goes to zero for New Animat. The episode length for DQN increases rapidly in the beginning and the mean reaches 100 the first time around 1700 time steps but small fluctuations of the standard deviation can be seen throughout this time frame. The episode length for Old Animat improves with the number of training time steps, and while it does not reach New Animat's or DQN's performance, it still outperforms both Q-learner and Random. The Q-learner and Random agent show no noticeable improvement in this time frame.

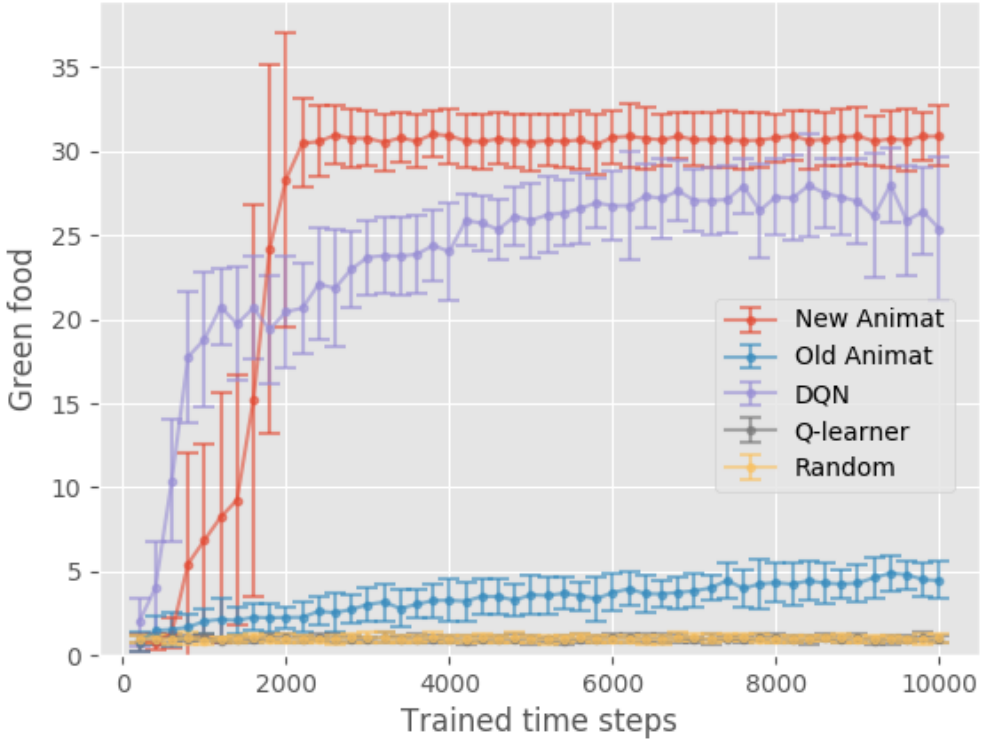


Figure 4.9: The mean and standard deviation of the number of food eaten while green light was active, this is the only way to gain energy. Initially, DQN consumes more green food than New Animat, but after 1800 time steps New Animat starts to perform better then DQN. The mean stabilises around 31 green food for New Animat, whereas for DQN the mean does not stabilise during this time frame. Old Animat performance is well below New Animat and DQN, but it is better than both Q-learner and Random. The Q-learner and Random agent show no noticeable improvement in this time frame.

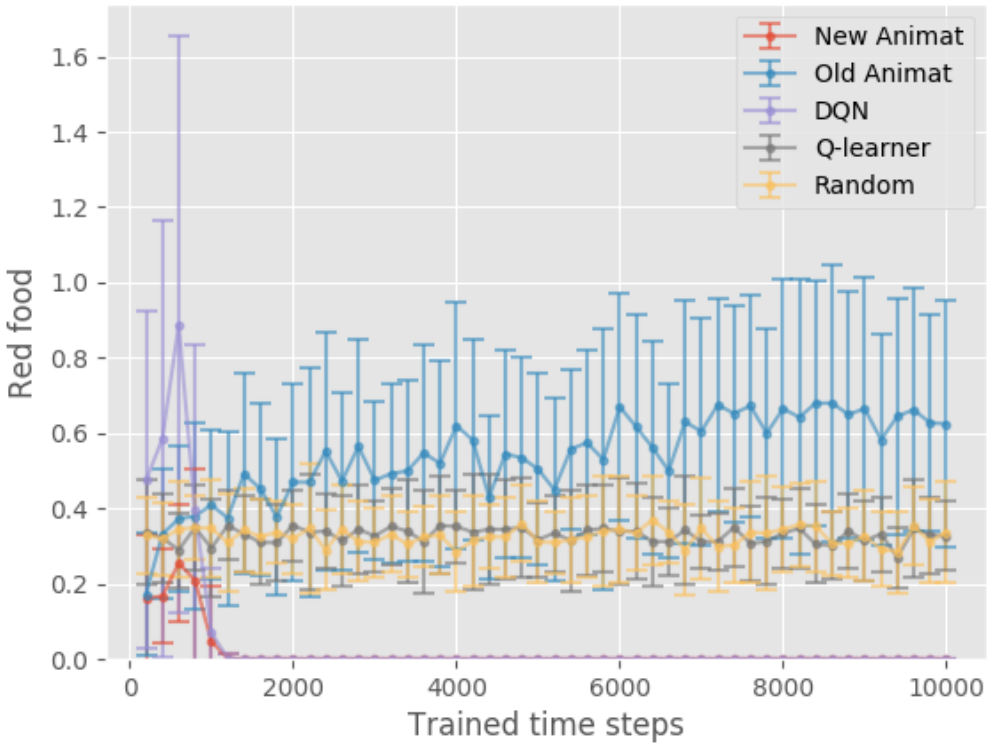


Figure 4.10: The mean and standard deviation of the number of food eaten while red light was active. It is clearly visible that New Animat and DQN are the only agents that learns to not eat food while the red light is active, and that this part of their respective policy is found after $\sim$ 1000 steps. Old Animat eats more red food compared to both Q-learner and Random.

5
Discussion

In this chapter, we discuss the results of the experiments, limitations and validity, future work and societal aspects regarding our work.

5.1 Formation rules

Even though the Old Animat agents formed twice as many nodes, it is evident from both experiments that the New Animat agents outperform the Old Animat agents. The New Animat agents exclusively use the new formation rules, see Definitions 21, 22 and 23; and the Old Animat agents use the old formation rules, see Definitions 11 and 12. We will now discuss the formation rules in more detail.

5.1.1 New formation rules

From Listing 4.1, we can see that the new formation rules are applied in the anticipated order: (i) positive reward merge (Definition 21), (ii) stable node merge (Definition 22), (iii) relevant node merge (Definition 23).

From Figure 4.2a we can see that New Animat 1 and New Animat 2 have stabilised in the number of nodes after ~ 3000 time steps. We can also see from Figure 4.1a and 4.3a, that the policy for those agents stabilises around that time, and from Figure 4.4, we see that the fraction of stable and relevant nodes stabilise around 3000 time steps as well. This indicates that the top activities need to have a coherent response if a GA agent should be able to find a good policy, and we can see that all of our new formation rules play an important role to achieve this goal.

The environment is designed so that the conjunction *Green Light* $\wedge$ *Center FOOD* is a stable node with respect to the action *eat*, i.e. the reward will always be positive if the action *eat* is performed while that node is active, see Section 3.3. By studying the Figures 4.4 and 4.5, we see that the node *Green Light* $\wedge$ *Center FOOD* becomes a stable node for all New Animat agents. We know that the agent will reach food if it moves in the direction indicated by its remote perception nodes (Dir-left/right/up/down FOOD). As anticipated, we see that they start being added to the set of relevant nodes shortly after the stable node has been identified, but they are not always added and they are not the only nodes that are being added.

A reason the remote perception nodes are not always added to the set of relevant nodes, could be because the agent learns to exploit hidden information in the environment. Since the remote perception nodes show the next nearest food when the agent stands on food, it is possible for the agent to use this information to find the

most likely direction to a wall. The environment is bounded, so if the agent moves into a wall it will simply stay where it is. Therefore, given that *Center FOOD* and *Red Light* is active, it is better to move in any direction other than the active remote perception node, since it is likely that a wall is present there. If the agent moved into the wall then the *Center FOOD* node will be active the next time step, and maybe also *Green Light*. Due to this exploit, the action that was used to walk into the wall will now be updated without the corresponding remote perception node being active in the previous time step, which make it less likely that the relevant transition probability becomes greater than the threshold $p_{Relevant}$, see Definitions 19 and 20.

To assure that the remote perception nodes receive a coherent response from the environment it is important to separate their top activities. All nodes in the set of relevant nodes will eventually be completely separated from each other, see definition 19. If only three remote perception nodes are added to the relevant nodes, the fourth remote perception node will be isolated from the remaining remote perception nodes and the effect is the same as if it was added, with a difference being that less nodes have been created. In Figure 4.2a we see that the number of nodes for New Animat 1 and New Animat 2 seems to stabilise after 3000 time steps, but the standard deviation is constant rather than moving towards 0. From the energy and red food, Figures 4.1 and 4.3, it seems that an optimal policy has been found by New Animat 1 and New Animat 2 after 3000 time steps, which indicates that the top activities are sufficiently separated by then, even though the remote perception nodes are not always added to the relevant nodes.

From Figures 4.4 and 4.5 we can see that after the remote perception nodes have started to be added to the relevant nodes the nodes *Center FOOD*, *Green Light* and *Red Light* are added as well. This tells us that those nodes are added with respect to the relevant nodes rather than the stable node. The nodes *Center FOOD* and *Green Light* are most likely added after the agent finds a good policy, since when a good policy has been found the agent will stop performing the action *eat* except when both *Center FOOD* and *Green Light* are active. Every time the agents has consumed green food one of the remote perception nodes will be active and eventually the threshold, $p_{Relevant}$, will be reached and *Center FOOD* and *Green Light* are added to relevant nodes. The *Red Light* node is probably added with respect to *Center FOOD*, due to the exploit mentioned earlier where the agents utilised the wall to stay at the same location, given that *Center FOOD* and *Red Light* was active.

5.1.1.1 Limitations

The new formation rules rely on the existence of positive rewards in the environment, i.e. the new formation rules will not be able to create any nodes in an environment where the reward function is defined without any positive rewards. However, the Generic Animat model focus on homeostatic agents, where the rewards are defined as the difference in each need. If a need reaches zero the agent dies, therefore, in the homeostatic setting, it is reasonable to expect that there should exist interactions that leads to an increasing need and thus positive rewards.

The positive stable nodes, see Definition 18, identifies nodes that exclusively

receives positive rewards as stable nodes. If we in the investigated environment were to remove the green light sensor from the observation space, the new formation rules would not work since no node would be identified as a stable node. The reason for this is because the reward history is based on the set of active nodes, see Definition 17, so even if the node *Red Light* $\wedge$ *Center FOOD* is top active, each of the predecessor nodes will update their corresponding reward history as well. It is therefore crucial that all potential stable nodes are either represented by a single sensor or a conjunction of several sensors, which can be assured by including the complement of each sensor.

From the first experiment, we saw that none of the noise sensors was added as relevant nodes. However, if we would increase the activity of the noise nodes, they too would eventually be added to relevant nodes as well. To find causation is not a trivial task, and with the simple strategy that we use to find relevant nodes, we know that in the limit of infinity all noise nodes will have been added to the relevant nodes. This is true since the probability is greater than zero that a noise node can be activated, prior to a stable or relevant node, so that the threshold $p_{Relevant}$ is eventually reached.

5.1.2 Old formation rules

The Old Animat agents mainly create nodes through its rule for emotional merge (see Definition 12), and since this rule randomly selects the top active nodes to connect, it is likely that many of the new nodes created by the Old Animat agents are based on noise.

When studying the energy and red food, see Figures 4.1b and 4.3b, we can see that Old Animat 1 and Old Animat 2 performs slightly better than Old Animat 3 and Old Animat 4 the first 3000 time steps. However, after 3000 time steps, the difference becomes negligible between using the average initial local Q-values and initial local Q-values set to zero. This indicates that, in the beginning, useful nodes are created, but after 4000 time steps noise might be added to a larger extent.

The old formation rules are built exclusively on the top active nodes, and one problem with this approach is that new nodes can easily become unnecessarily complex since the set of top active nodes describes the current state to its highest level of detail. As time move on it is therefore expected that new nodes are more likely to be based on an increasing amount of sensors, but a node that is based on many sensors is less likely to become active. For example, in this experiment, the probability that a node based on three noise nodes becomes active is $0.25^3 \approx 0.016$. This fact is most likely the reason why the difference between the definitions of the initial local Q-values is so small after 3000 time steps, even though we from Figure 4.2b can see that we continuously create new nodes.

The performance with regards to red food, see Figure 4.10, is worse for Old Animat compared to both Q-learner and Random. The fact that the amount of red food increases as Old Animat learns to consume more green food (see Figure 4.9), underlines the problems where information propagates through the top activities. Some nodes can be top active together with both green food and red food. Therefore the information about how good it was to eat green food is propagated to the red

food and vice versa. However, the policy found by Old Animat regarding energy, episode length and green food is better than Q-learner and Random (see Figures 4.6, 4.8, 4.9. This indicates that even with a perception graph that is not optimally designed, the state representation can initially help the agent to find a better policy.

5.2 Reliability

From the first experiment, it is clear that the reliability set to be constant one or by using Definition 6, have little to no effect on the agent's policy. So with these results and an identified possibility for contradicting behaviour of the current reliability (see Section 2.6), we recommend setting the reliability to one rather than keeping the current version.

5.3 Initial local Q-values

Comparing New Animat 1 and New Animat 2 to New Animat 3 and New Animat 4 in the first experiment, it is clearly beneficial to use the average of the predecessors local Q-values for new nodes. In Figure 4.3a, we can see that the number of Red food increases temporarily for New Animat 3 and New Animat 4, which happens at the same time as nodes start being created by relevant node merge, see Figure 4.2a and 4.5. This shows that for this environment, it is possible for an agent to temporarily forget its current policy when creating new nodes if the initial local Q-values are set to zero.

5.4 Model comparison

In Figures 4.1a and 4.6 we see that New Animat has similar energy performance in both experiments. We also see that even though the state space between the experiments is vastly different, a good policy is still found around ~ 2200 time steps in both experiments.

In the second experiment we have seen that DQN quickly improves with time, but it does not reach the same level of performance as New Animat, see Figures 4.6, 4.8, 4.9 and 4.10. However, performance alone does not tell the complete story. Considering that the environment and rules are designed with an animat perspective, and adapted to suit the Generic Animat model with its boolean sensory inputs, there are two points worth mentioning. Compared to the Generic Animat model a DQN can be considered more general since: (i) it is not limited to boolean values as input, and in theory could handle continuous state spaces directly, and (ii) it does not rely on complicated node formation rules. But compared to the perception graph a neural network can be considered: (i) more of a black box approach, and might not be suitable for tasks where explainability for actions are of high importance, and (ii) more rigid in its layout, in that it is not very easy to change the topology without having to retrain the complete network.

Since the state space is larger than 2^{32} it is unlikely that the same state is visited more than once within a time frame of 10000 time steps, this is confirmed in Figure 4.7, where we see linear growth in the Q-learner's unique states visited. It is therefore expected that the performance of Q-learner should be close to Random, which has been confirmed in all presented statistics (see Figures 4.6, 4.8, 4.9 and 4.10). So we can conclude that the state space in the second experiment is too large for an ordinary Q-learning agent to learn a good policy within 10000 time steps.

5.5 Future work

In this thesis, valuable contributions to the Generic Animat model are the three rules developed for making the correct node connections, which enables the model to converge towards a policy with the potential to maximise a need. We do this by identifying and forming the nodes contributing to positive rewards, and isolating them by connecting them to all other nodes that can be top active simultaneously with these nodes. Another approach could be to only use the stable and relevant nodes in the top activity rather than making sure they are entirely isolated. This way it might be possible that for each need, we can filter out irrelevant nodes from the top activity, and then base the update of the local Q-values and the selected action on this trimmed top activity instead. And if the current reliability concept were made binary, it could be used to apply this filtering.

We believe this approach is well worth exploring further since it has potential to significantly reduce the number of nodes that are formed, which in turn would reduce the computational complexity of the model and make the perception graph simpler to understand. It might also make sense from a neurological perspective, considering how the brain process, filter and store information. According to McNab and Klingberg [27], the human brains *basal ganglia* plays a big part in filtering out irrelevant information, and making sure only the relevant parts are stored in the brains working memory. And Hélie et al. [28], argue that one important part of the *basal ganglia* is to train connections between posterior *cortical* areas and frontal *cortical* regions that are responsible for automatic behaviour after extensive training.

It is also essential to evaluate the GA model in more scenarios, and there are plenty of well established POMPD benchmarks available [29]. But since the GA model relies on boolean sensory input (observation space) and discrete output (action space) the number of readily available scenarios shrinks considerably, and if changes are made we need to pay special consideration so that the complexity remains intact for results to be comparable.

5.6 Societal, ethical and ecological aspects

When considering the societal, ecological and ethical aspects with regards to the *animal* part of an animat, we have in this thesis only dealt with artificially simulated animals. However, the animat model draws inspiration from biology, which has real, living creatures, and thus we consider the ethical aspects of our simulations. Regarding animals, an estimation is that on a yearly basis up to 100 million [30]

vertebrae animals are used for research. And the ethical aspects of animal testing are a subject often debated and with public opinion shifting over time, also it is not uncommon with threats against researchers involved in animal experiments [31].

In their book *The Principles of Humane Experimental Technique*, Russell and Burch (1959) [32], the authors propose a concept involving 3 R's when dealing with animal experiments. These R's stand for replacement, reduction and refinement, and can be interpreted as:

- Replacement: if possible considering the research goal, the animal in question should be replaced whenever possible by non-animal methods such as simulations or other innate models. If a non-animal approach is not possible, an animal considered less sentient, or that has a lower pain perception should be used.
- Reduction: the aim should be to use as few animals as possible to complete the experiment, taking into consideration the welfare of the individual animals.
- Refinement: by using suitable methods, pain, suffering and distress should be minimised during the animals lifetime.

In our environment and experiments, we have to express the agent graphically somehow, we could have used a simple symbol like a circle for this, but we chose to represent it with an image of a cat. We used a cat since we wanted to emphasise the biological connection of the animat model, and a cat is something to which we can easily relate. However, we did design the environment and its rules not to promote what could be considered unethical or cruel methods. For example, we specify the reward functions as a change in need, and cues from the environment are simple observations. We could have implemented an environment closely resembling the Skinner box [23] together with its shock generator and electrocuting grid.

When viewing the Generic Animat research project as a step towards artificial general intelligence (AGI), there are also some issues to address. Recently an open letter [33] regarding potential autonomous weapons development was published, and the current round in the competition "The general AI challenge" [34] is currently about *Solving the AI race*. Research regarding autonomous systems has the potential for significant societal impact in many areas, and as always there are two sides of a coin. Progress can be used to do good, but often the opposite is also true, and technology can potentially be misused for more nefarious or unethical purposes. However, we argue that for the work done in this thesis no special precaution or concern needs to be taken since: (i) the research are on an elementary level (ii) animats live in a simulated, contained, environment with no possibility for interaction with the outside world.

6
Conclusion

In this thesis, we have studied the Generic Animat model proposed by Strannegård et al. [11] and identified three possible areas for improvement. The first regarding the state representation using the top activity and Q-global calculation. The second regarding the reliabilitys effect on choosing the optimal action to take in any given state. And the third involving the effect initial local Q-values can have on a currently learned policy when creating new nodes.

Solutions have been implemented and validated in a simple yet noisy environment, where it was clear that the solutions implemented for this *New Animat*, substantially increased the performance compared to the original Generic Animat model. It was also shown that in an environment with a state space large enough to render ordinary Q-learning infeasible, the New Animat has comparable or better performance to a DQN agent.

In addition, a framework, suitable for continued research within reinforcement learning in an animat context has been developed and open-sourced.

Bibliography

[1] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes, "Rmdl: Random multimodel deep learning for classification," *arXiv preprint arXiv:1805.01890*, 2018.

[2] "Dota 2," 2017. [Online]. Available: https://blog.openai.com/dota-2/

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[4] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, 2016.

[5] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, Nov 1995, pp. 1942–1948 vol.4.

[7] S. W. Wilson, "Knowledge growth in an artificial animal," in *Adaptive and Learning Systems*. Springer, 1986, pp. 255–264.

[8] ——, "The animat path to ai," *In Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, pp. 15–21, 1991.

[9] J.-A. Meyer, A. Guillot, B. Girard, M. Khamassi, P. Pirim, and A. Berthoz, "The psikharpax project: Towards building an artificial rat," *Robotics and autonomous systems*, vol. 50, no. 4, pp. 211–223, 2005.

[10] N. Yoshida, "Homeostatic agent for general environment," *Journal of Artificial General Intelligence*, 2017.

[11] C. Strannegård, N. Svangård, D. Lindström, J. Bach, and B. Steunebrink, "The animat path to artificial general intelligence," in *Proceedings of the Workshop on Architectures for Generality and Autonomy 2017*, 2017. [Online].

Available: http://cadia.ru.is/workshops/aga2017/proceedings/AGA_2017_Strannegard_et_al.pdf

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. A Bradford Book, 1998.

[13] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[14] J. A. Hertz, *Introduction to the theory of neural computation*. CRC Press, 2018.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[17] "Reinforcement learning w/ keras + openai: Dqns," 2017. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c

[18] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[19] H. Mei, F. Chen, Y.-D. Feng, and J. Yang, "Abc: An architecture based, component oriented approach to software development," *Journal of Software*, vol. 14, no. 4, pp. 721–732, 2003.

[20] Y. Sugimori, K. Kusunoki, F. Cho, and S. Uchikawa, "Toyota production system and kanban system materialization of just-in-time and respect-for-human system," *The International Journal of Production Research*, vol. 15, no. 6, pp. 553–564, 1977.

[21] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*. IEEE, 2013, pp. 9–16.

[22] J. Bentley, "Programming pearls: algorithm design techniques," *Communications of the ACM*, vol. 27, no. 9, pp. 865–873, 1984.

[23] S. McLeod, "Skinner-operant conditioning," 2017. [Online]. Available: https://www.simplypsychology.org/operant-conditioning.html

[24] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1997.

[25] "Keras: The python deep learning library," 2018. [Online]. Available: https://keras.io

[26] "Tensorflow: An open source machine learning framework for everyone," 2018. [Online]. Available: https://www.tensorflow.org

[27] F. McNab and T. Klingberg, "Prefrontal cortex and basal ganglia control access to working memory," *Nature neuroscience*, vol. 11, no. 1, p. 103, 2008.

[28] S. Hélie, S. W. Ell, and F. G. Ashby, "Learning robust cortico-cortical associations with the basal ganglia: An integrative review," *Cortex*, vol. 64, pp. 123–135, 2015.

[29] "The pomdp page - partially observable markov decision processes," 2018. [Online]. Available: http://www.pomdp.org/index.html

[30] "Alternatives to animal testing gaining ground," 2010. [Online]. Available: http://articles.baltimoresun.com/2010-08-26/health/bs-hs-animal-testing-20100826_1_consumer-products-animal-testing-experiments

[31] B. Huggett, "When animal rights turns ugly," *Nature biotechnology*, vol. 26, no. 6, pp. 603–606, 2008.

[32] W. M. S. Russell and R. L. Burch, *The Principles of Humane Experimental Technique*. Methuen London, 1959.

[33] "Autonomous weapons: an open letter from ai & robotics researchers," 2018. [Online]. Available: https://futureoflife.org/open-letter-autonomous-weapons

[34] "General ai challenge," 2018. [Online]. Available: https://www.general-ai-challenge.org

[35] "Pypi - the python package index," 2018. [Online]. Available: https://pypi.python.org/pypi

[36] "Gym," 2018. [Online]. Available: https://gym.openai.com

A

Development framework

A.1 Installation

This section describes the steps necessary to setup a minimal development framework.

1. Virtual machine and OS installation (optional).
2. Native packages.
3. Git configuration.
4. Python virtual environment setup.
5. Installation of backend for Keras (optional).
6. Project retrieval and installation.
7. Running an example.

If you don't have access to a native Linux environment we recommend running Ubuntu 16.04 LTS (`https://www.ubuntu.com/download/desktop`) under VMware Workstation Player 14 (`https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html`).

Install the needed native packages:

```
animat@ubuntu:~$ sudo apt-get install git python3-pip python3-tk python3-venv
    graphviz
```

Configure git:

```
animat@ubuntu:~$ git config --global user.name "John Doe"
animat@ubuntu:~$ git config --global user.email johndoe@example.com
```

Retrieve the project:

```
animat@ubuntu:~$ git clone https://gitlab.com/fredrikma/aaa_survivability.git
```

Create and activate a Python 3 virtual environment:

```
animat@ubuntu:~$ python3 -m venv my_animat_env
animat@ubuntu:~$ source my_animat_env/bin/activate
(my_animat_env) animat@ubuntu:~$
```

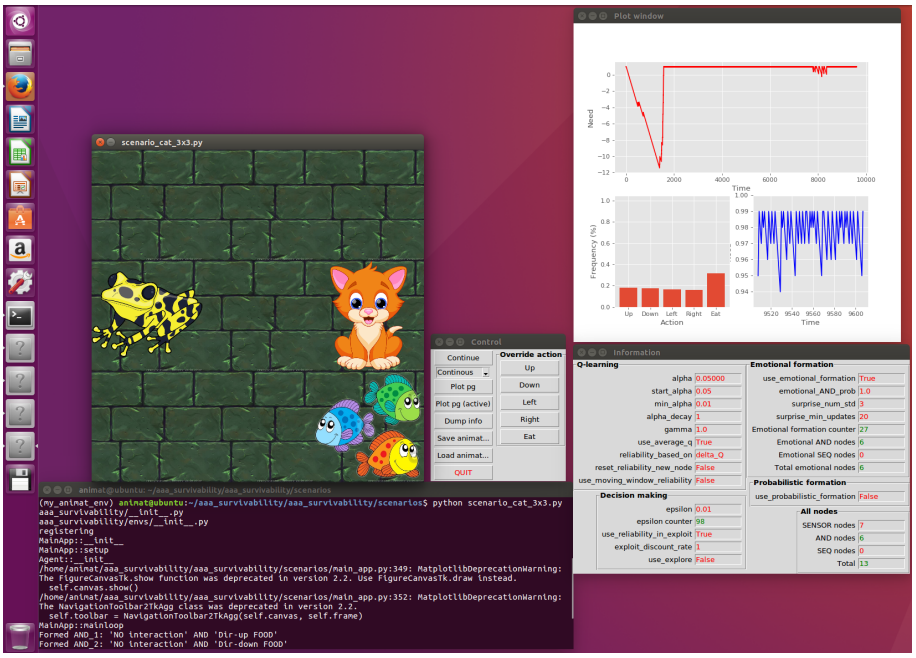


Figure A.1: The example scenario for Cat3x3 running in the framework. The upper left part shows the 3x3 tiled environment with an agent (cat) and two edible objects (fish and frog). The small middle window contains the scenario controls, while the two windows to the right shows real-time statistics and configuration parameters.

Install TensorFlow as backend for Keras, this is optional if you already have one installed. Follow `https://www.tensorflow.org/install/install_linux` or:

```
animat@ubuntu:~$ pip install tensorflow
```

Install the project:

```
(my_animat_env) animat@ubuntu:~$ cd aaa_survivability/
(my_animat_env) animat@ubuntu:~/aaa_survivability$ pip install -e .
...
Installing collected packages: aaa-survivability
  Running setup.py develop for aaa-survivability
Successfully installed aaa-survivability
```

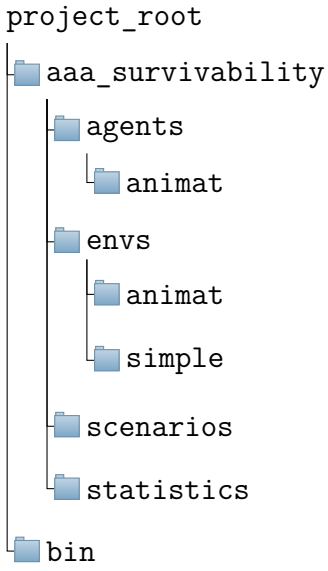
Running an example:

```
(my_animat_env) animat@ubuntu:~/aaa_survivability$ cd aaa_survivability
(my_animat_env) animat@ubuntu:~/aaa_survivability/aaa_survivability$ cd scenarios/
(my_animat_env) animat@ubuntu:~/aaa_survivability/aaa_survivability/scenarios$
    python scenario_cat_3x3.py
```

If everything worked well something similar to figure A.1 should show.

A.2 Folder structure

The project is structured according to the Python Package Index [35] guidelines, and the overall folder structure can be seen below.



- `aaa_survivability` and sub-folders contains the python source code.
- `aaa_survivability/agents` contains the implementation for various agents.
- `aaa_survivability/envs` contains environments which follow the OpenAI gym interface [36].
- `aaa_survivability/scenarios` contains runnable programs, both with and without gui
- `aaa_survivability/statistics` contains runnable programs to calculate statistics for the various scenarios.
- `bin` contains shellscripts to easily run scenarios, generate statistics and save out the results.

A.3 Graphical framework

The graphical framework for which an example can be seen in Figure A.1, is created to enable easy control and measurement of an animat based agents performance in a chosen environment. Some of the things possible to do in this framework includes

- Continuous or single-step time.
- Export and visualization of the perception graph.
- Saving and loading of an animat.
- Action override.
- Display of *need* and *action* statistics.
- Display of the animat configuration and statistics.

A.4 Python code

The code is written in a modular way so that testing agents against new scenarios should be pretty straightforward. For example configuring and running an animat agent in the Cat 3x3 environment (appendix B.2) with a GUI, can be accomplished with a few lines of code

```
from aaa_survivability.scenarios.main_app import MainApp

if __name__ == '__main__':
    config_agent = {'use_probabilistic_merge': False,
                    'probabilistic_merge_prob': 0.01,
                    'use_emotional_merge': False,
                    'use_reward_based_merge': True,
                    'prob_reward_based_merge': 1.0,
                    'reward_based_threshold': 3,
                    'use_stable_node_merge': True,
                    'stable_threshold': 20,
                    'use_average_q': True,
                    'start_alpha': 0.05,
                    'gamma' : 0.9,
                    'alpha_decay': 1}

    config = {}
    config['Agent'] = config_agent

    app = MainApp('cat_3x3-v0', config, animat_baseline=True)
    app.mainloop()
```

B

Environments

During development and testing, we created a few environments to help test basic functionality and make sure that our implementation of the models was correct. Here we will give brief descriptions of those environments.

B.1 Eat/Drink

This is a very simple environment to help us verify that the Q-learning part of the models is working on a basic level. The agent has two actions to choose from, $a \in$ {Eat, Drink}. The observation space consists of one sensor which is always active. If the agent eats it receives a positive reward, and if it drinks it receives a negative reward. The goal for the agent is thus to only eat and refrain from drinking. Some statistics from an agent in this environment can be seen in figure B.1.

B.2 Cat 3x3

The environment in figure B.2, is a more complex environment which contains basic vision sensors and actions for movement. It is designed to test more parts of a model compared to the eat/drink environment. The world is tile-based of size 3x3, and there are two edible food objects of different types that the agent can interact with. One healthy food in the form of fish, and one poisonous food represented by a frog. The agent has 5 actions to choose from, $a \in$ {Move Up, Move Down, Move Left, Move Right, Eat}.
The observation space consists of:

- Current tile sensor for fish, active if the agent stands on a fish.
- Current tile sensor for frog, active if the agent stands on a frog.
- Current tile sensor other, active if the agent is not standing on a fish or a frog.
- Four sensors giving the direction to the fish object {Up, Down, Left, Right}, that are not mutually exclusive. If the fish is located diagonally from the agent, two sensors will be active.

The rewards are given as:

- Eat fish: $r = 0.5$

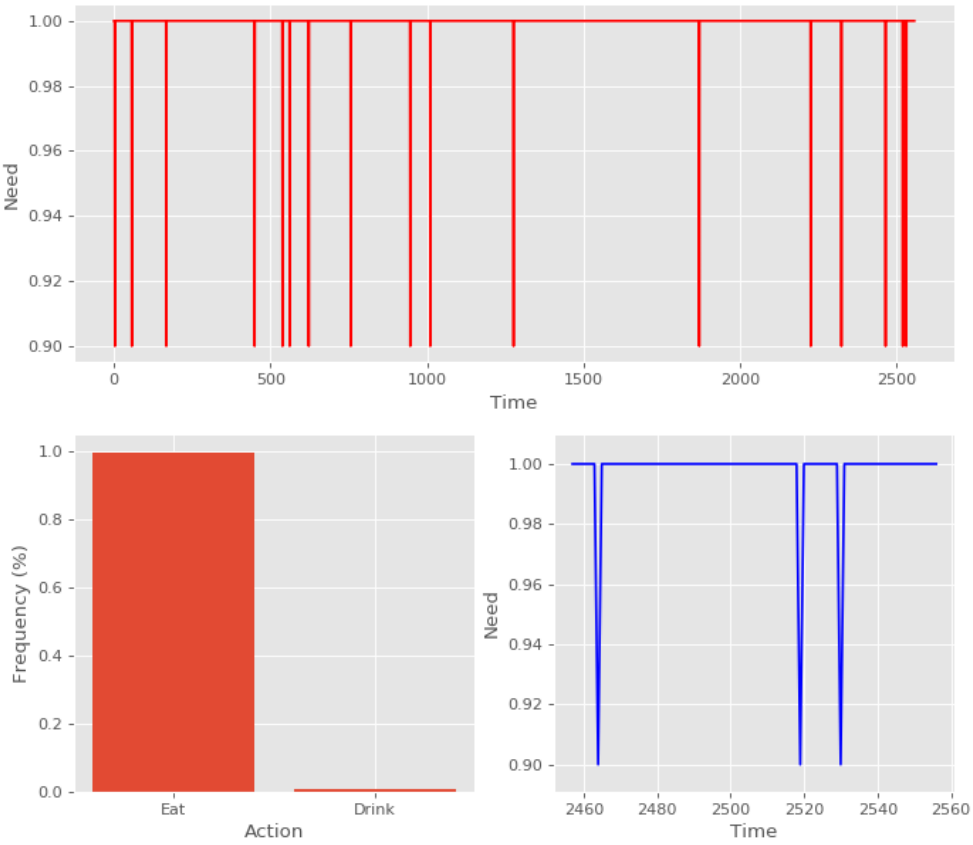


Figure B.1: The statistics for an agent in the eat/drink environment. We can see that the agent has found a policy that enables it to sustain its need over time. Exploration is active, and that explains the dips in the need curve.

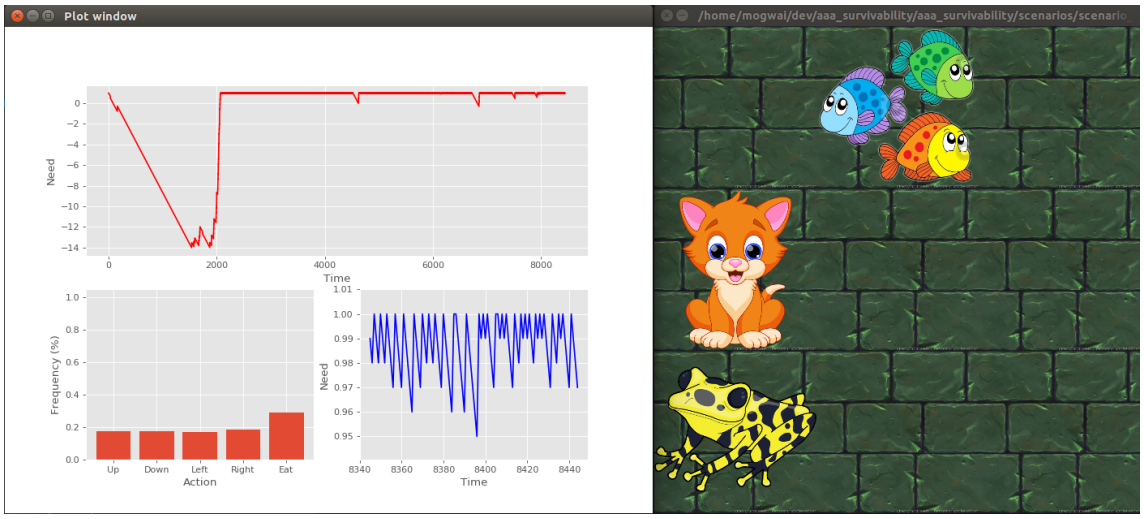


Figure B.2: The environment designed to test most parts of a model. From the statistics on the left side, we can see that the agent (cat) has found a policy that enables it to sustain its need over time. Exploration is active, and that explains the dips in the need curve.

- Eat frog: $r = -0.2$
- Eat something else: $r = -0.01$
- Movement: $r = -0.01$

The goal for the agent is thus to learn how to navigate towards fish and then eat it, and refrain from eating any frogs.

B.3 Remote perception

This environment which can be seen in figure 3.1, is used to test the sensors for remote perception as described in section 3.3.2 and equation 3.1. The world is tile-based of configurable size (default 20x20) and contains a configurable amount (default 100) of healthy food represented by fish. Eating is automatic in this environment, and fish are consumed as soon as the agent occupies the same tile as a fish. When all fish are consumed, they all respawn at new random locations. For any tile, there is a maximum of one fish occupying it. The agent has 4 actions to choose from, $a \in \{\text{Move Up, Move Down, Move Left, Move Right}\}$.
The observation space consists of:

- 4 sensors giving the direction toward strongest fish smell {Up, Down, Left, Right}.

The rewards are given as:

- Consume fish: $r = 0.2$
- Movement: $r = -0.001$

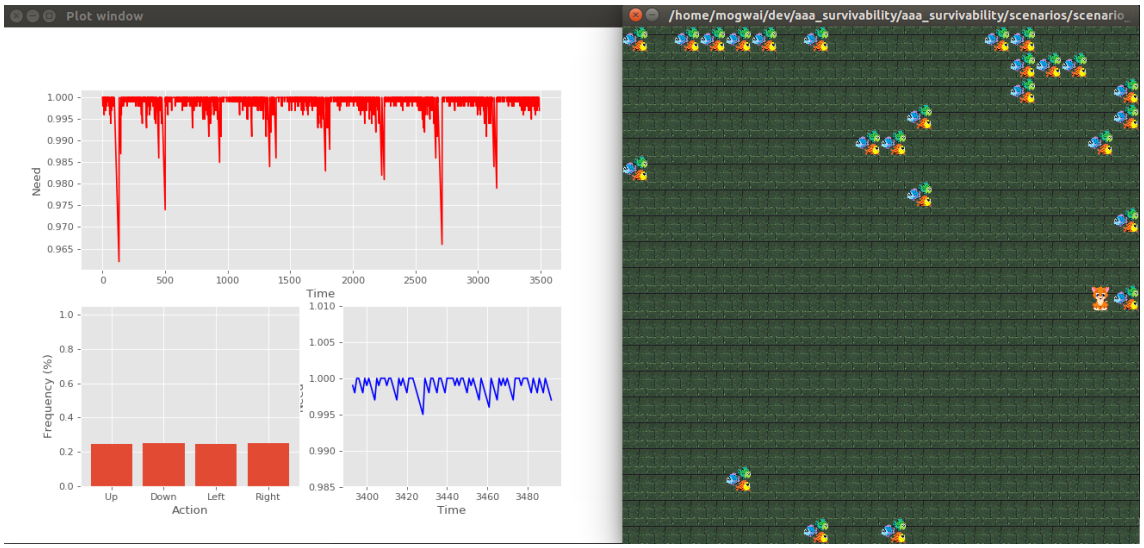


Figure B.3: The environment designed to test the remote perception. From the statistics on the left side, we can see that the agent (cat) has found a policy which enables it to survive indefinitely.

The goal for the agent is thus to learn how to navigate towards fish by using its 'smelly vision'.

B.4 Yoshida 1

Figure B.4 show an environment inspired by one described in Yoshida [10]. The world is bounded, tile-based of size 3x3, and just as in Cat 3x3 (section B.2) the agent can interact with two edible food objects. One healthy in the form of fish, and one poisonous represented by a frog. The differences compared to the Cat 3x3 environment lies in the agent's observation space and goal. The agent in this scenario has an energy level $E \in [0, 100]$ and the goal is to keep this energy level as close as possible to the target energy $E_{target} = 60$. This energy is translated into a need for the agent according to equation B.1.

There are 44 sensory inputs in total, 20 of these represents the discretised energy level, and the other 24 are constructed so that absolute positions for the fish, frog and agent can be computed. The sensors might not seem very animat like but are chosen so that we can recreate the observation space used by Yoshida [10] and reach a similar state space complexity. This sensory input makes the environment extremely hard for The Generic Animat model since there are few opportunities for transfer learning.

For a more in-depth explanation of this environment, please see the source code accompanying this thesis (`https://gitlab.com/fredrikma/aaa_survivability.git`).

$$\text{need} = \max(1 - \frac{|E_{target} - E|}{E_{target}}, 0) \tag{B.1}$$

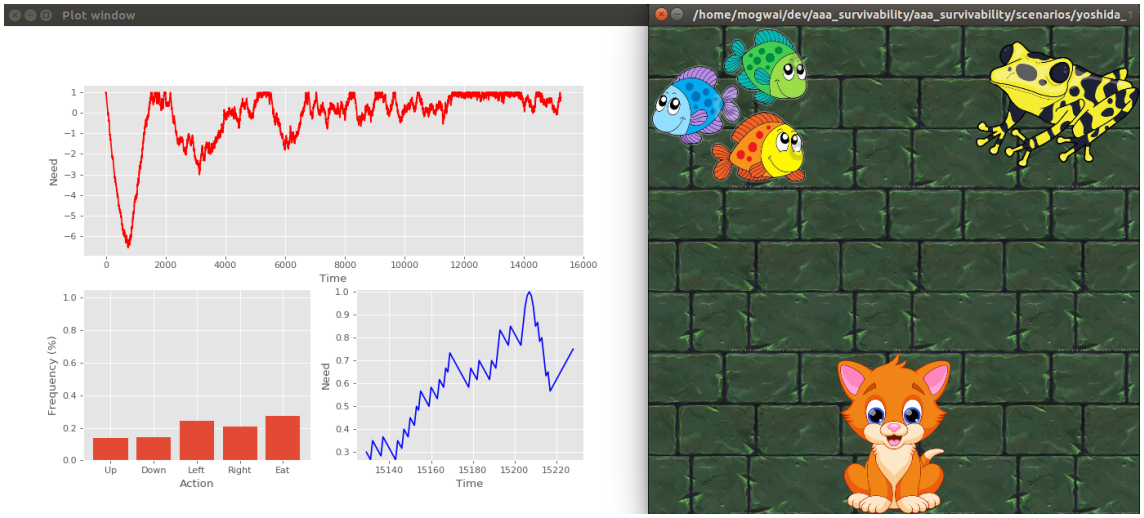


Figure B.4: The Yoshida 1 environment which offers few opportunities for transfer learning, and from the statistics on the left side we see that the agent is struggling to find a good policy.