



Universiteit
Leiden
The Netherlands

Data Science and Artificial Intelligence

Crossy Road AI:
How will the chicken cross the road?

Lex Janssens

Supervisors:

Matthias Müller-Brockhausen & Thomas Moerland

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

www.liacs.leidenuniv.nl

24/07/2023

Abstract

In this thesis, we have done research on deep learning reinforcement learning agents in a simplified self-made version of the game *Crossy Road*. The game is played autonomously using a deep reinforcement learning algorithm called Proximal Policy Optimization (PPO). The environment is an infinite world, with the agent observing states consisting of a portion of the world relative to the agent. The main objective is to show the different effects of the size of the portion of state representation. We have conducted three experiments, highlighting the difference between the use of step rewards as well as the use of a double state representation clearly exposing various objects in the environment separately. The training shows that for all agents, agents using a larger state representation perform on average better. Agents with a smaller state representation get stuck behind obstacles more. Yielding heuristic rewards for actions helps the agent progress, manipulating moving forwards and rewarding patience over exploration. The use of a double-state representation for an agent with a relatively small single-state representation is beneficial. Double-state representations for agents with a relatively large single-state representation expose the problem of the curse of dimensionality. Furthermore, the evaluation shows that the standard deviation for all agents is high and that some portions of the world are harder to navigate. This thesis exposes us to carefully think about the size of a state representation for a given environment, as well as the implications of providing the agent a reward for a specific action.

Contents

1	Introduction	1
2	Related work	2
3	Background	4
3.1	Frogger vs. Crossy Road	4
3.2	Proximal Policy Optimization	6
3.2.1	Policy Gradient Method	6
3.2.2	Trust Region Method	7
3.2.3	Proximal Policy Optimization	7
3.3	SHAP	8
4	Methods	9
4.1	The environment	9
4.1.1	Static layers	10
4.1.2	Non-static layers	11
4.1.3	Custom	14
4.2	World generation	14
4.3	State space	16
4.4	Action space	17
4.5	Transition function	17
4.6	Reward function	19

4.7	Initial state distribution	19
4.8	Reinforcement learning model	20
5	Results	21
5.1	No step reward and single-value cell representation	21
5.2	With step reward and single-value cell representation	25
5.3	With step reward and double-value cell representation	27
5.4	Comparison	30
5.5	SHAP analysis	32
6	Discussion	33
7	Conclusion	35
	References	37
8	References images	37
A	Symbol definitions	38
B	Hyperparameter values	39
C	World generation algorithm	41
D	Training plots	42
D.1	No step reward and single-value cell representation	42
D.2	With step reward and single-value cell representation	43
D.3	With step reward and double-value cell representation	44

1 Introduction

Reinforcement learning [SB18] is an active topic in the field of Artificial Intelligence (AI). Reinforcement learning is learning from experience: an agent and an environment communicate with each other to gain feedback as to how it performs a certain task. From the perspective of the agent, looking towards an environment is capturing an observation: a state. A state contains information about the current setting of the environment, for instance, where certain objects are or what certain variables are. Based on this information, the task of the reinforcement learning agent is to take the optimal action when given this state information. Intuitively, one might think that better choices can be made with more information, but this is tricky. The agent does not know what these values specifically mean, and which are relevant for its current choice of action. It has to be learned that some values are more relevant to the situation than other values. Furthermore, to increase the number of values representing a state is to increase the state space. This introduces the curse of dimensionality [BM03], which states that increasing the size of a state representation, is to exponentially increase the number of parameters that have to be learned.

Knegt *et al.* [KDW18] wondered the same: how to reduce the complexity of a state representation, thus reducing the complexity of the state space. For their research, they used a vision grid to represent a state in the Game of Tron. In this game, an agent is guiding a light cycle against an opponent in a 10×10 grid world. The player must also avoid hitting any walls or light trails from either the opponent or themselves whilst doing this. Furthermore, a vision grid is a grid relative to the agent of which it can observe values. In their paper, the agent is the center of the grid, where the grid has a dimension of $n \times n$, where n is a positive odd number. They used multiple vision grids to denote different features of observation, and multiple sizes for said combination of vision grids (3×3 and 5×5). There are separate vision grids for light trails of the player itself, light trails by the opponent, and walls. These are binary grids, denoting the presence of a feature relative to the agent with a value of 1, whilst a value of 0 denotes the absence. They showed that, in comparison to using the whole grid as a state representation, vision grids state representations reduce the state space significantly, allow for better learning, and in most cases allowed for better performance. They have shown that the larger vision grid outperformed the smaller vision grid, and rarely the full grid outperformed the larger vision grid. However, there was not a lot of insight into the effects of the vision grids other than performance. The reduction of the state space is a significant part of the performance increase, however, we missed a clear explanation as to why these sizes are used and what advantages and/or disadvantages the agent has with its vision.

In this thesis, we want to experiment with various sizes of vision grids as well as cell representations, along with the reward shaping of actions. We ask ourselves the following research question:

What are the effects of reinforcement learning agent performance using different observation space complexities for the game Crossy Road?

For this, we have implemented a self-made simplified version of *Crossy Road* as a sequential decision problem where the agent gets to choose an action for each new game state. This environment is in itself infinitely large, and highly dynamic with obstacles to avoid and attributes to use to their advantage. In this environment, it can be observed what the effects of different vision grids

are, not only in a metric of performance but also in behavior. We will perform three experiments where the effects of observation space complexities are central. The first experiment show the overall comparison in performance between agents using different vision grid sizes with no applied heuristic reward for actions. The second experiment however does apply a heuristic in the reward function such that patience is valued over moving around and moving forwards is promoted over moving backwards. Here, the question rises: *Has this heuristic impact on the reinforcement learning agent performance because of different utilization of the vision grid?* Lastly, to represent cells distinctively, we experimented with a more complex representation of cells in order to find out: *Is this increased complexity for representing cells distinctively worth the performance difference?*

Results have shown that the more vision an agent has, the better the performance. The smaller the vision grid, the more the agent gets stuck. However, the larger the vision grid is, the more the problem of poor generalization arises due to the curse of dimensionality. Reward shaping for actions improves the performance of the agent. A state representation of multiple vision grids resolves the problem of getting stuck as much, but due to the added dimension, this is not a good trade-off for agents with larger vision grids. The importance of the number of values an observation is represented with to a reinforcement learning agent could truly be limiting its performance. Using the explainable AI method Shapley, the agents showed that cells contributing to their path are of the largest importance when it comes to deciding upon action, with cells on the said path further away having less importance than cells close by relative to the agent.

Our contributions include a Gym [BCP⁺16] environment which represents a simple version of *Crossy Road*. This environment includes various options for adjusting the vision grid size, hyperparameters in the environment, reward function, and an easy structure to change any layers and/or section generation. There are options for recording an episode as well. With the extensive feedback from the environment, we contribute extensive evaluations of agents with various vision grid sizes in the settings of: using a single vision grid without step reward; using a single vision grid with step reward; and using a double vision grid with step reward. Evaluations include but are not limited to, the way the agent terminated the environment and where it was at that particular moment. During training as well as during evaluation, a shift between the way agents terminated the environment and where that happened can be observed.

We will first start by going over related work in section 2, then continue with background knowledge in section 3. Additionally, we will extensively explain the methods in section 4. The results are shown in section 5 along with appendix D. Lastly, we will reflect on our work in section 6 and end with a conclusion in section 7.

2 Related work

Research by Knecht *et al.* [KDW18] made use of vision grids in reinforcement learning agents. In this research, opponent modeling is used in the Game of Tron using reinforcement learning. The Game of Tron is played on a grid, where there is a player present as well as an opponent. This research suggested making use of relative vision grids for the agent to represent only a partial view of the grid as representation. The game grid was subdivided into three binary grids, one to denote the player's

position, one to denote the opponent's position, and one to denote the walls present in the vision. This reduced the state space tremendously. They showed that vision grids increase the learning speed along with an increase in performance when compared to using the full grid as a state representation.

The thesis by van der Velde [vdV18] experimented with Frogger with a single multi-layer perceptron as well as a double multi-layer perceptron, where one is meant for the Road section and one for the Logs section. Along with that, they experimented with a single-action network and compared the performance. Single-action networks are networks that yield an approximation of the Q-value of a state-action pair. As input, they have used a vision grid of the world. Instead of using cells, they have used blocks of 30x30 pixels to denote the value of that cell in the state space. The vision grid reaches out 4 cells forward, and 2 steps both left, right, and downwards. Both approaches managed to get passed the Road section, but could not get across the Logs section. For both single-action, single multi-layer perceptron, and double multi-layer perceptron, the performance was not comparable to human performance.

The thesis of Maijers [Mai19] continued with this project, implementing 2-step Q-learning and 4-step Q-learning [PW96] with two different reward functions: action-based reward functions and distance-based reward functions. They showed that this does affect the percentage of Road section completion, but does not improve the win rate. They have shown that multi-step Q-learning with an action-based reward function performs significantly better than a distance-based reward function. They noted that the agent has little patience on the Logs section and that not penalizing doing nothing in the reward function could potentially help.

Emigh *et al.* [EKB⁺14] also experimented with Frogger. They have used two different kinds of representations to describe the game state. The first is a 'holistic' feature construction, which targets agent locations globally across the entire environment, and a 'local' feature construction, which is comparable to the vision grid concept, along with additional information about whether spots have been filled with frogs or not. They argue that this way of state representation by stating the curse of dimensionality has a great impact on the learning process [Bel10]. Along with this, they used Q-learning along with nearest neighbor action value approximation to let the agent learn. This is a way to evaluate unvisited states by approximation concerning the action values of neighboring states. They showed that there both methods of feature construction succeed in navigating the frog, with 'holistic' feature construction making metric learning greatly improve the learning performance. Furthermore, they showed that with metric learning, irrelevant features were eliminated or decreased in importance when making action value updates.

Reward shaping is steering the agent's behavior via a certain heuristic, by giving rewards for sub-objectives. If the objective function is to maximize the reward received, the agent will explore ways to do so. By reward shaping, a heuristic is added to reduce this exploration by stating what sub-objectives are steering in the right direction [BHV⁺14]. For example, an additional reward could be to keep the largest distance from other agents [DKG11]. Overall, learning in environments where there are sparse rewards yielded, reward shaping helps to speed up the learning process. Providing additional penalties or additional rewards can expose the directed behavior of an agent. Additionally, rewards and penalties lower the training time for learning a certain task [NHR99].

PPO agents have been shown to learn complex behavior in elaborate environments [HTS⁺17]. The observation space consists of two parts, where the first part is focused on the sensors of the agent, including velocity, acceleration, and gyroscope but also values of each joint angle on the legs and torso. The second part is environmentally based, where just a part relative to the position of the agent is known. They show that an agent can show locomotion, which is sensitive to reward shaping. Agents must navigate various terrains, including bumpy terrain, jumping over obstacles, slaloming walls, and jumping over gaps. They show that, by extensive reward shaping, agents utilizing a PPO model can possess the skill of locomotion.

Previous research on Atari games with Deep Reinforcement Learning shows the method of using convolutional network [MKS⁺13]. An Atari emulator is used to interact with a game, where a state is a literal frame of the game, and the output is an action emulated in the game. They showed that a convolutional neural network, trained with a variant of Q-learning, taking in the raw pixel values of a frame can be used as a value function to estimate future rewards. Whilst Atari games are visually fairly simple games, they did outperform humans for three out of six games, and show state-of-the-art results in six of seven games.

A study by Beechey [BS3] aims to explain reinforcement learning decisions using Shapley. For one, they showed how Shapley values are applied to policies. They argue that Shapley values for feature importance are not a standalone explanation of the performance of an agent, but rather a tool to describe the currently chosen action based on given features. This is true for both a value estimation function as well as a policy function. The paper extends upon this problem by proposing a new method using Shapley to identify feature contributions to the performance of the agent. This is done by observing the consequences of leaving one feature out and quantifying the performance change. They state to have found meaningful explanations of performance that match human intuition via this method.

3 Background

3.1 Frogger vs. Crossy Road

The game we will be exploring is *Crossy Road*, released by *Hipster Whale* in 2014 [Wha14]. This game is a successor to the arcade game *Frogger*, released in 1981, developed by Konami and manufactured by Sega [Kon81].

Starting with *Frogger*, the objective is to guide the agent, in this case, a frog, to each spot on the other side of the grid-based environment. A visualization can be seen in figure 1. The agent starts at the bottom and has to work its way up to the top, considering two large tasks. First of all, the agent has to cross a five-layer road section, where vehicles from left and right with various speeds cross by. The agent must not get hit, or it will lose a life (out of a total of five). Separated by a safe layer, the next section is a log section, where the agent must navigate itself to one of the empty spots. There are logs, turtles, and alligators the agent can stand on. Turtles appear and disappear from time to time, so moving swiftly is obligatory. Alligators can be stepped on at any point, except for the head. For all objects, the agent moves along with the objects, meaning it can float off-screen.

The process of filling an empty spot with a frog has to be repeated five times to move on to the next level. There is a time limit for navigating the frog to the other side, which is usually set to 60 seconds. Any time unused will be added to the total points. Furthermore, occasional spawning flies be caught which yields additional points. The game ends when the player has no more lives left.

Crossy Road is fairly similar to *Frogger*, and can be seen in figure 2. It is seen as an endless adaptation to *Frogger*. The objective is slightly different: get as far as possible without colliding with vehicles and utilizing objects. The grid-based environment is not just two sections, but an infinite amount of sections. New parts of the environment get generated the further you get. *Crossy Road* adopts the two sections *Frogger* provided, namely the Road section and the Logs section. A new section is added in this adaptation, which is a Rail section. This section contains a high-speed train that rushes by at certain intervals. The agent gets warned via a light signal. Sections can be of variable length based on their type, chance, and progression. For Logs and Road sections, this average lies around three layers. For the Rail section, this average is usually lower. The further the agent progresses, the more difficult the environment gets. This affects vehicle and object speeds, the density of vehicles and objects, and the number of layers a section contains. The agent only has one life, and it loses it in collision with vehicles, jumping into the water, or moving off-screen. A time limit also exists in the original game, making sure that the agent does not wait too long before moving on. The agent must reach a new layer to reset that time limit. Visually in the game, the environment moves slowly off-screen when no actions are performed. Once the agent gets too close to the bottom of the screen, the environment ends. The game contains golden coins, which are randomly distributed in the environment and can be picked up by the agent. This does not add any points to the score but does increase the player’s in-game currency to unlock new characters.



Figure 1: The game *Frogger*. The goal of this level-based game is to get five frogs in each gap at the top of the level, whilst starting at the bottom avoiding cars and utilizing logs. Source: [con23]



Figure 2: The game *Crossy Road*. The goal of this endless game is to get across as many layers as possible, by avoiding cars and trains and utilizing logs. Source: [Wha14]

We will be using *Crossy Road* as a baseline for this environment. We will be building the environment ourselves since that provides us with more information about the positions of the objects and allows more control over the environment. Previous research of a DQN network on Atari games shows the use of convolutional networks [MKS⁺13], but we feel like this approach for the actual game *Crossy Road* is a lot harder. The environment we will be building is similar but does differ in the fact that there are no coins to be picked up and a possible time limit works differently. Furthermore, world generation will deviate from the game, primarily since there are no exact numbers as to how the world is generated in the original game. Difficulty works slightly differently too, mainly because it only increases the number of layers per section, but does not interfere with the speeds or density.

3.2 Proximal Policy Optimization

3.2.1 Policy Gradient Method

Proximal Policy Optimization (PPO) [SWD⁺17] wants to strike a balance between easy implementation, sample efficiency, and easy tuning. It is a policy gradient method, which means it learns online and directly from experience. PPO does not have a replay buffer, since it leads to unjustified large policy updates, which could potentially destroy the current policy. The experience is only used once, before being discarded, meaning that this model is less sample efficient than models like a deep Q-learning network (DQN) [MKS⁺13]. PPO, as stated, is a policy gradient method. It uses a policy gradient method, which is defined as

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log\pi_\theta(a_t|s_t)\hat{A}_t]$$

where π_θ is the policy network, $\hat{\mathbb{E}}_t$ the expectation at time t , $\log\pi_\theta(a_t|s_t)$ the logarithm of selecting action a in s at time t , and \hat{A}_t the advantage. There are two parts to this objective function. The first part is $\log\pi_\theta(a_t|s_t)$. These probabilities are generated using a neural network. This is a policy network π_θ , which takes in the state as the input and generates action probabilities as the output. The second part is \hat{A}_t , which is the estimated advantage when taking some action a given some state s . To calculate the advantage, discounted rewards and baseline estimates are needed. The discounted rewards are the reward function, which is defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where γ is the discount factor and r_t at time step t . k represents the future expected steps along with the expected rewards. This reward function sums up the discounted rewards from the current episode. This function does exactly that. With a discount factor $\gamma = 0.99$, the reward function rewards current rewards more than rewards it will get later.

The baseline estimate is the estimate of the discounted reward from time t . This is estimated by another neural network, namely a value function or baseline network, which takes in a state and produces $V(s)$: the estimate of the discounted return from current time t . This is a noisy estimate. Putting them together, \hat{A}_t is defined as

$$\hat{A}_t = G_t - V(s)$$

or the advantage is the discounted rewards minus the baseline estimate. Or, "how much better is the action that the agent took than expected?"

Now, it knows what reward it received from that state, namely the reward from the action taken from the previous state. This now becomes a supervised learning problem, where $V(s)$ should be as closely predicted to the reward r observed. This neural network, as well as the policy neural network, are updated accordingly to the error of the value function. Coming back to the original objective function, yielding a higher \hat{A}_t yields a higher a_t , thus providing more chance to an action being picked that has a higher advantage. This works vice versa: an action yielding a disadvantage, will lower a_t and thus be picked less.

3.2.2 Trust Region Method

There is a problem, however, which we mentioned before: it cannot update on a dataset of experience, but rather learn from online experience. This has to do with the fact that the experiences in such a dataset are chronologically unrelated. The way \hat{A}_t is calculated, is by observing the reward the agent making action a from a state s to yield s' . Using a random state s to calculate the advantage over, yields unrelated results and cause the value function to be completely wrong, thus training falsely.

The solution for this is a Trust Region Method, which in this case is Trust Region Policy Optimization (TRPO) [SLM+15]. It makes sure that the new policy is not too far off from the old policy, i.e. the updates are not too significant. It uses the following objective function:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$$

subject to $\hat{\mathbb{E}}[\text{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)]] \leq \delta$

where $\pi_{\theta_{old}}$ denotes the old policy, KL denotes the Kullback-Leibler divergence, which is a type of statistical distance, and lastly δ denotes the value to which the network is constrained for updating. This objective function consists of two parts. The first part is in essence the same as the first part of the objective function L^{PG} , but instead of applying the *log*, it is turned into a fraction. The purpose of this part of the objective function remains the same: a policy gradient method [Sch17]. The second part is a constraint that the update of the policy is not too significant, i.e. the new policy lies closely to the old policy, for which we know the policy performs well. Adjusting δ allows us to tune the balance between exploration and exploitation, by allowing newer policies to make more or less errors than the old policies. TRPO has a so-called surrogate objective function. This means that instead of optimizing the model to yield the most cumulative rewards, it optimizes the policy updates since this is simpler and more efficient.

3.2.3 Proximal Policy Optimization

The problem that TRPO faces, is that the objective and the constraint are separate. These both serve different purposes and require different optimizations. The functions alone do not function as an objective function, but rather the balanced combination is what makes TRPO powerful.

PPO [SWD+17] introduces a new way of a surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

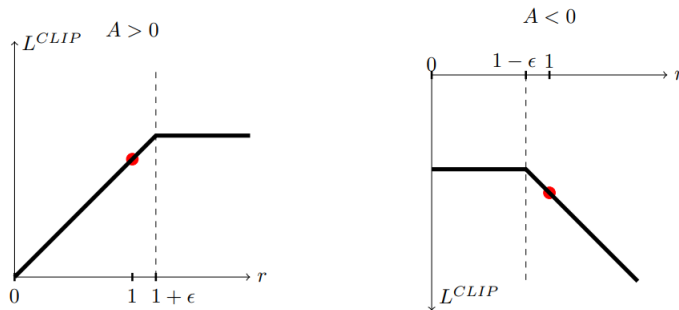


Figure 3: The way the surrogate objective function L^{CLIP} functions. Here you can see both advantages $A > 0$ and $A < 0$. The red dot shows the starting point for the optimization ($r = 1$) [SWD+17]

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the hyperparameter clip range $\epsilon = 0.2$ that allows for smaller or larger updates, and 'clip' the function that limits values at a certain extent. First of all, it uses an expectation. So it going to compute this objective over some batches. Then, it takes the minimum of two terms: the normal policy gradients objective, which is seen in TRPO, and a clipped version of the normal policy gradients objective. Figure 3 shows the way the surrogate objective function works. For a positive advantage, thus the action was better than expected, it limits at $1 + \epsilon$. This limits too big updates to the policy if the current action was too good. For a negative advantage, thus the action as worse than expected, it limits at $1 - \epsilon$. Similarly, if the action was too bad, it does not get updated too significantly in one update. Note how the other sides of the graphs (i.e. $r < 1$ for the left graph and $r > 1$ for the right graph), are not clipped. You could argue this is a safeguard for this function: the policy only ends up on either side when the action was less probable (left) or more probable (right) but made the policy function worse. This then acts like an undo function, making the exception that a bigger policy update is allowed here to 'fix' the bad policy update.

3.3 SHAP

It is important to find out why a model makes certain predictions. Especially for models with more complexity, interpretability helps understand but also correct the models. SHapley Additive exPlanations (SHAP) is an evaluation method that aims to explain machine learning models using a game theoretic approach [LL17]. The purpose of SHAP values is to explain the contribution of each feature in the input for a model's prediction. SHAP values are established via cooperative game theory where it can distribute the predicted value over all different features, by assigning an impact value compared to a baseline reference. To understand a model using SHAP values, not only the predicted output can be evaluated with these impact values, but also the output not predicted. This makes it interesting in a setting of reinforcement learning to find out what features were responsible for choosing or not choosing a certain action given a certain state, where the state is a collection of features.

For the evaluation of a policy network given a collection of values provided by a vision grid, it can be observed whether values closer or further relative from the agent are more important or less

important to the output respectively. We assume that this is the case, and such an assumption would look like the heatmap in figure 4 on the left. This figure shows values closer to the agent (provided in a lighter color) are more important than values further away from the agent (provided in a darker color). Furthermore, such an evaluation per time step can also expose whether some (combination of) action(s) is preferred by being less dangerous. This should then expose a combination of lighter colored cells, relatively close to each other and suggesting a certain direction, which implicitly is a combination of actions, which can be seen in the same figure on the right.

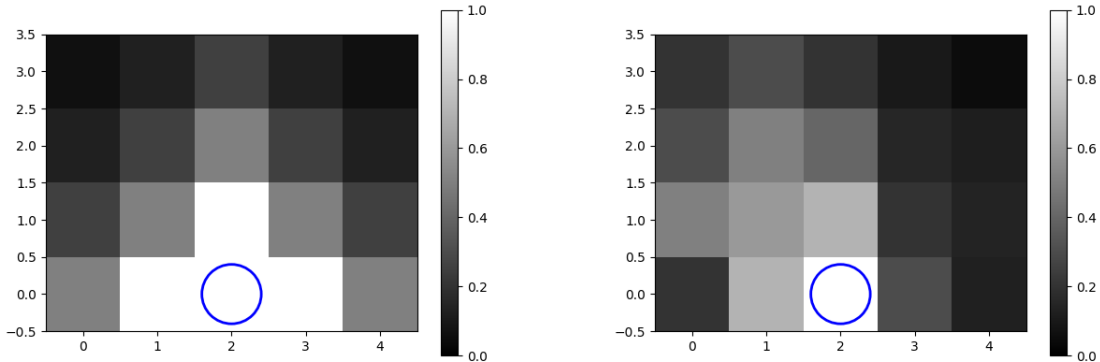


Figure 4: The overall expectation of absolute normalized importance for each cell in a vision grid using SHAP values. The blue colored circle represents the location of the agent. The colors range from dark to light. We assumed that the importance would decay the further the cell is away relative to the agent, as seen on the left. Another possibility is that the agent has a preferred direction, which implicitly states a combination of actions towards a certain direction, which can be seen on the right.

4 Methods

4.1 The environment

The environment is an endless world, just like in the original game. A world \mathcal{W} itself is a grid world and has a width $\mathcal{W}_w = 15$ and a height $\mathcal{W}_h = |\mathcal{L}| = \infty$. This differs from the original version: this environment is 5 cells wider. Each horizontal slice of the grid world is a separate layer. You could thus say that the height of a world \mathcal{W} is made up of an ordered list of infinite layers $\mathcal{L} = \{L_1, L_2, \dots, L_\infty\}$. In the environment itself, only a maximum of 14 layers are displayed. However, due to the mechanics of the game, this could potentially be an endless array.

The cell space \mathcal{C} is the set of all cells in the representation \mathcal{R} of each layer, such that

$$\mathcal{C} = \bigcup_{L \in \mathcal{L}} \mathcal{R}(L)$$

which has $|\mathcal{C}| \geq \mathcal{W}_w \times \mathcal{W}_h$. There are three types of cells present in the environment: valid cells, invalid cells, and terminal cells. The set of valid cells $\mathcal{C}_V \subset \mathcal{C}$ are cells valid for the agent to enter.

The set of invalid cells $\mathcal{C}_I \subset \mathcal{C}$ are states that cannot be entered by the agent. Lastly, the set of terminal cells $\mathcal{C}_T \subset \mathcal{C}$ end the environment once an agent enters. From a cell $c \in \mathcal{C}_T$, no more actions can be taken. The union of these sets gives

$$\bigcup \{\mathcal{C}_V, \mathcal{C}_I, \mathcal{C}_T\} = \mathcal{C}$$

The agent can move from some cell c to a cell outside of the environment. These are called truncated cells. They are the cells outside of the environment, which in theory, is possible for the agent to explore, but are not part of \mathcal{C} . Like $c \in \mathcal{C}_T$, truncated cells end the environment. For clarification, a cell c can only be part of one aforementioned subset, such that

$$\bigsqcup_{x \in \{V, I, T\}} \mathcal{C}_x = \mathcal{C}_V \sqcup \mathcal{C}_I \sqcup \mathcal{C}_T$$

As stated, the world \mathcal{W} contains an ordered list \mathcal{L} of layers. A layer is a one-dimensional self-contained world. This means that each layer has a representation $\mathcal{R}(L) \subset \mathcal{C}$, and may or may not have an updating method. The representation of a layer L is the whole collection of all cells that make up the layer. There is also the observation of a layer. The observation for a layer L is defined as $\mathcal{O}(L(t)) \subset \mathcal{S}$, where $|\mathcal{O}(L(t))| = \mathcal{W}_w$. The observation of a layer is the cells visible in the current world at some time t . Note that

$$\forall L \in \mathcal{L}, \mathcal{O}(L(t)) \subseteq \mathcal{R}(L)$$

The way they differ is that a representation of a layer could be a larger set of cells than the observation. This has to do with the different types of layers that there are. The types of layers can be subdivided into two categories: static layers and non-static layers. Static layers do not contain agents, and thus have a static observation. Non-static layers, on the other hand, do contain agents and thus have a dynamic observation. Non-static layers have an update function that adjusts $\mathcal{O}(L(t))$ according to some time t . Therefore, the observation of some non-static layers is the partition of the representation of that layer respective to t .

There are several layers types of layers. A type of layer L is defined as $type(L) \in types$, where $types = \{\text{Empty}, \text{Bush}, \text{Logs}, \text{Road}, \text{Lilypad}, \text{Rail}, \text{Custom}\}$. We will refer to the collection of layers $\mathcal{L}_x \subset \mathcal{L}$ with some $x \in types$ as

$$\mathcal{L}_x = \{L | L \in \mathcal{L} \wedge x = type(L) \in types\}$$

and the collection of cells of that set as

$$\mathcal{C}_x = \bigcup_{L \in \mathcal{L}_x} \mathcal{R}(L)$$

4.1.1 Static layers

There are three types of static layers: **Empty**, **Bush**, and **Lilypad**. Static layers, as mentioned, are layers that do not contain any agents. The term static refers to the observation of that layer: for every time step t , the observation of a static layer L is the same as the representation of that layer.

$$\forall t, \mathcal{R}(L) \doteq \mathcal{O}(L(t)) \tag{1}$$

Empty

The **Empty** layer is the simplest. The layer describes itself: it is empty. Every cell in this layer is a valid cell: $\mathcal{C}_{\text{Empty}} \subset \mathcal{C}_V$. Visually, the layer looks like grass tiles.



Figure 5: The visual representation of the **Empty** layer. Source: 8

Bush

The **Bush** layer is similar to the **Empty** layer. However, this layer contains bushes. Bushes are considered invalid cells, and cannot be entered by the agent. The layer thus contains a mix of valid and invalid cells: $\mathcal{C}_{\text{Bush}} \subset \mathcal{C}_V \cup \mathcal{C}_I$. The ratio of valid and invalid cells in this layer is determined by a parameter d : density. The density is defined as

$$d_{\text{Bush}} = \frac{|\{c \in \mathcal{C}_{\text{Bush}} | c \in \mathcal{C}_I\}|}{|\mathcal{C}_{\text{Bush}}|} \in [0, 1]$$

For example, $d_{\text{Bush}} = 1$ means that the layer is entirely filled with bushes, whilst $d_{\text{Bush}} = 0.3$ the layer is filled for 30% with bushes.



Figure 6: The visual representation of the **Bush** layer. The cells containing a bush cannot be entered. Source: 8

Lilypad

The **Lilypad** layer looks like the opposite of the **Bush** layer. This layer contains terminal cells $c \in \mathcal{C}_T$. Visually, the layer looks like water with lilypads. The cells without lilypads are terminal cells, whilst the cells with lilypads are valid cells: $\mathcal{C}_{\text{Lilypad}} \subset \mathcal{C}_V \cup \mathcal{C}_T$. The ratio is again determined by a parameter d_{Lilypad} , in a similar way as the density of the **Bush** layer:

$$d_{\text{Lilypad}} = \frac{|\{c \in \mathcal{C}_{\text{Lilypad}} | c \in \mathcal{C}_V\}|}{|\mathcal{C}_{\text{Lilypad}}|} \in [0, 1]$$

4.1.2 Non-static layers

There are three types of non-static layers: **Road**, **Logs**, and **Rail**. These layers contain agents: vehicles, logs, and trains respectively. These agents move horizontally through the environment and have an effect on the observed cells for that layer. For static layers, we have discussed that these have the same representation as observation for every t . The opposite applies to non-static layers.



Figure 7: The visual representation of the Lilypads layer. The lilypads are the only $c \in \mathcal{C}_V$ on this layer. Source: 8

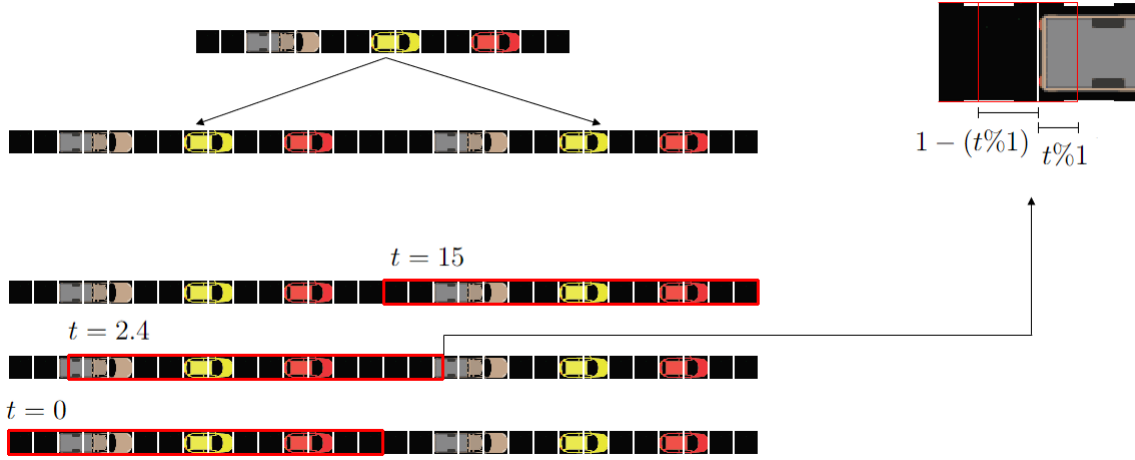


Figure 8: A sliding window approach to establish cell values based on agents. The left figure shows how the sliding window works. For instance, for this Road layer, the representation of width \mathcal{W}_w is copied to create a representation of size $\mathcal{W}_w \times 2$. A window of size \mathcal{W}_w , displayed in red, slides of the whole representation \mathcal{R} to make an observation with respect to t : $\mathcal{O}(L(t))$. Notice that the window moves in the opposite direction of the way the vehicles are facing. This creates the illusion the vehicles are moving. On the right you can see how the average of 2 cells is established. The red box shows a single cell of the sliding window. The sliding window covers 2 cells, each with a proportion of $1 - (t\%1)$ and $t\%1$. The average is thus the sum of the weighted values of the cells.

$\mathcal{O}(L(t))$ is established through a sliding window manner. Figure 8 shows how this is done. The window slides over the representation to create an observation concerning t . Note that, since the original representation consisted of \mathcal{W}_w cells and the representation is literally copied, that $\mathcal{O}(L(0)) = \mathcal{O}(L(\mathcal{W}_w))$. The time t for a layer L lies in the range $[0, \mathcal{W}_w]$. To make sure t lies within this range and will not exceed any side, the following formula is applied after updating t : $t = (t + \mathcal{W}_w)\% \mathcal{W}_w$, where $\%$ denotes the modulo operator. This can be seen as a cycle, and with the fact that the representation is copied, it creates the illusion that once an agent moves out of the screen, it re-enters on the other side. Note that this procedure also states that the configuration of the agents does not change. This is in line with the game, where the configuration of agents is repeated after some amount of time steps.

The range where t lies also shows that t can take any decimal value within that range. This means that one cell is shown as a combination of two parts of two cells. An example can be seen on the right in figure 8. The proportion of the weight of each cell in the calculation of said average is respective to $t\%1$: the two cells each have a contribution of $1 - (t\%1)$ and $t\%1$. The cell that greater

value is considered the discrete cell value. You could also state that a cell has changed value when

$$\lfloor t\%1 \rfloor \neq \lfloor (t + \Delta t)\%1 \rfloor \quad (2)$$

The three types of non-static layers are **Road**, **Logs**, and **Train**.

Road

The **Road** layer looks like a highway with cars present. The cells of the road where no car is present are $c \in \mathcal{C}_V$, whilst the cells where there are parts of a car present are $c \in \mathcal{C}_T$, thus $\mathcal{C}_{\text{Road}} \subset \mathcal{C}_V \cup \mathcal{C}_T$. There are three parameters in this layer that are relevant to its behavior. These are the cycle speed cs_{Road} , the configuration $conf_{\text{Road}}$ and the direction dir_{Road} . The cycle speed cs_{Road} is defined as the difference in the time t in the layer once the layer is updated. This can also be denoted as Δt for the layer. The cycle speed in itself is the number of time steps it takes for the sliding window to make a cycle. The configuration $conf_{\text{Road}}$ of the cars is a preset of where the cells that make up the car are located in the layer. The reason for doing this instead of a density approach is because of access-ability: randomly placed car cells created hard-to-impossible configurations for an agent to pass. With preset car cell configurations, it is made sure that there is always a gap between cars and that randomness cannot interfere with the difficulty. Lastly, there is a parameter for the direction dir_{Road} the sliding window is moving. This can either be left or right.



Figure 9: The visual representation of the Roads layer. Source: 8

Logs

The **Logs** layer is quite an interesting layer. Like the **Road** layer, $\mathcal{C}_{\text{Logs}} \subset \mathcal{C}_V \cup \mathcal{C}_T$. This layer looks like a river with logs floating in it. The cells containing water are $c \in \mathcal{C}_T$ whilst the cells containing (a part of) a log are $c \in \mathcal{C}_V$. Similarly, there is a sliding window concept going on in this layer. However, the agent moves along with the sliding window on this layer. Visually, this makes sense: the agent stands on a log that floats in a direction, thus not moving would mean that the agent moves along with the log. There are some interesting properties to denote here. First up, what would happen when the agent stays put on the logs? It would mean that the agent, along with the log, will float off the screen. But instead of the agent continuing the cycle as the log does, the environment will be truncated. Secondly, how are moves made on the logs, whilst there is an offset? This is solved by using the property explained in formula 2: the agent will still be part of a single cell and move along when the aforementioned property is met. Lastly, moving from a **Logs** layer to another layer, the offset will be calculated and set according to the layer and the current cell the agent is in. There are again three parameters in this layer that are relevant to its behavior. These are the cycle speed cs_{Logs} , the configuration $conf_{\text{Logs}}$ and the direction dir_{Logs} . The cycle speed cs_{Logs} works similarly as in the **Road** layer: it is the number of time steps it takes for the sliding window to make a cycle. A similar problem occurred with the random generation of the logs: hard-to-impossible situations were randomly created. To get more control of the generation, the configuration $conf_{\text{Logs}}$ is introduced. The configurations of logs are several configurations of

where the logs are located. Lastly, there is a parameter for the direction dir_{Logs} the sliding window is moving. This can either be left or right.



Figure 10: The visual representation of the **Logs** layer. The agent moves along with the log once stood upon. Source: 8

Rail

The **Rail** layer is unlike any other layer. This layer has as the base all cells $c \in \mathcal{C}_V$. However, at a certain interval, a train will speed down the track. The agent will be warned via a light signal and a cell representation that a train is coming. The train cells are $c \in \mathcal{C}_T$ and are 7 cells 'wide'. There are three parameters in this layer: the interval i_{Rail} of the train, the speed sp_{Rail} of the train, and the direction dir_{Rail} of the train. The interval i_{Rail} is the number of time steps before the train will come. In the meantime, the layer is safe to enter and walk on. The speed sp_{Rail} is the number of cells it moves each time step. Note that it does not move in partitions: it moves whole states, and no cell can be partly occupied by a train and partly not. The direction dir_{Rail} the train moves in is the third parameter, and this can be in either the left or right direction.



Figure 11: The **Rail** layer. The train moves in whole cells. Source: 8

4.1.3 Custom

There is an option to add custom layers to the environment. Not only can you pass different values for the predefined layers, but you can also define a custom representation of a layer. For instance, the starting area of \mathcal{W} consists of the same three layers: a layer $L \in \mathcal{L}_{\text{Bush}}$, where $d_{\text{Bush}} = 1.0$ such that the whole layer is filled with bushes, and thus $\forall c \in L, c \in \mathcal{C}_I$ making walking backward from the start impossible. The following two layers are **Custom** layers, with a custom static representation $\mathcal{O}(L(t))$ such that the states on the edges of the layer are bushes, and the rest of the layer is an **Empty** layer.

4.2 World generation

The world has a specific way of generating layers: by adding sections. This is comparable to the actual game, where you could see that there are multiple similar layers generated after each other, followed by a static layer or another section. A section can be one or more layers. When the agent moves forward, more world is generated once there is little world left. There is also a factor of difficulty to be taken into account. The actual game gets harder by adding harder and more

non-static sections, along with changing cycle speeds and densities.

The world has a specific way of generating layers: by adding sections. This is comparable to the actual game, where you could see that there are multiple similar layers generated after each other, followed by a static layer or another section. A section can be one or more layers. When the agent moves forward, more world is generated once there is little world left. This addition happens seamlessly outside of the visual range of the game. There is also a factor of difficulty to be taken into account. The actual game gets harder by adding harder and more non-static sections, along with changing cycle speeds and densities.

Algorithm 1: Section generation

```

1 INITIALIZE: some decay value  $\theta < 1$ 
2  $base \leftarrow base + 0.01$ 
3  $currentBase \leftarrow base$ 
4 SELECT SECTION TYPE UNIFORMLY
5
6 WHILE  $r \sim U(0,1) \leq currentBase$ :
7     ADD NON-STATIC LAYER OF SAME TYPE
8      $currentBase \leftarrow currentBase \times \theta$ 
9 ADD STATIC LAYER

```

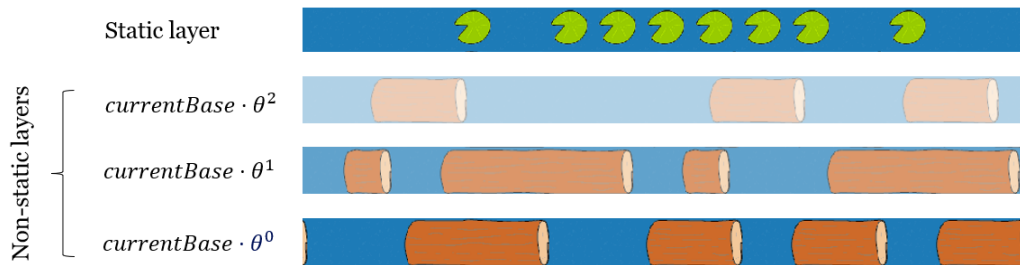


Figure 12: This figure shows a visualisation of the idea of algorithm 1. Any next layer generated has a decreased chance of being part of the section, displayed with transparency denoting lesser chance. The $currentBase$ decreases by a factor θ . The section ends with a static layer.

Now, for some types of layers, the decay factor θ is higher than others. A higher value for the decay factor θ means that there is more chance for more layers to be generated. Similarly, increasing the $base$ at each step will make sure longer sections are added. Longer sections of non-static layers are in general harder. A difference with the game is that the cycle speed nor configurations change at a higher difficulty.

To make sure a section can be crossed by the agent, there is an additional check. This check uses a breadth-first search (BFS) [Kal12] pathfinder to check if it is possible to cross. The BFS pathfinder simulates the environment for several steps, and will then check with the actions available by the agent if it is possible. A heuristic depth of 50 timesteps is used: the agent must be able to make it to the other side within 50 timesteps. The reason for using a heuristic is the following: suppose a scenario where 4 non-static layers are added, each with a different cycle speed between 50 and 250,

then the number of time steps it takes for the 4 non-static layers to get back to the initial state at the same time, it could take up to $LCM(250, 249, 248, 247) \approx 1.9 \cdot 10^9$ time steps, where LCM is the lowest common multiple. This also assumes that the cycle speeds are natural numbers, but in this setting, they could be real numbers. A heuristic is therefore applied to evaluate whether it is possible.

4.3 State space

All this time, we have been talking about cells. The reason for this is that the state space consists of a selection of these cells. For the observation space, we will make use of a vision grid. A vision grid is, quite literally, a grid of vision in each direction relative to the agent. What differs from the way Knecht *et al.* [KDW18] has utilized them, is that the sizes of the vision grid are various and that the agent does not necessarily have to be the center of the vision grid here. Figure 13 shows how the vision grid looks within the game. Each visible cell in the environment will get represented by a single or multiple values, based on the weighted sum of the cells the sliding window covers. Looking at the color coding: the greener-looking cells lean more towards a cell $c \in \mathcal{C}_V$, whilst more red-looking cells tend to lean towards $c \in \mathcal{C}_T$. Dark-colored cells are $c \in \mathcal{C}_I$. The vision grid then states how many of these cells are visible in each direction. In this case, the vision grid relative to the agent, is 2 cells to the left, 2 cells to the right, 2 cells upwards, and 0 downwards. This gets flattened, from bottom to top, into a one-dimensional array to yield a state. As you can see, the larger the vision grid, the more values, the larger the representation of a state.

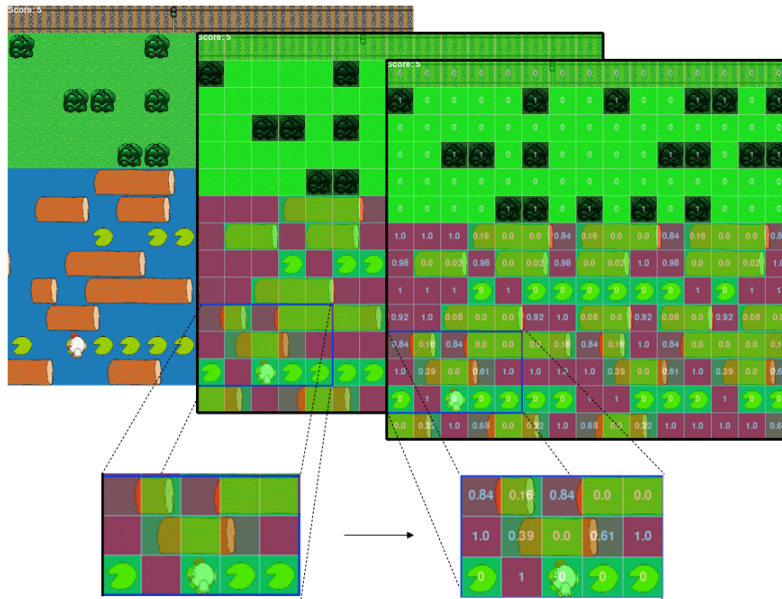


Figure 13: Here you can see how the values are assigned to each cell. In this case, $|c| = 1$ was used. You can see how the vision grid is a 5x3 box of cells, marked in blue. The cells will get assigned a value based on the proportion of the cell that is terminal or invalid, done via the process explained in 8. The color gradient also tells how safe a cell is, with red leaning towards terminal and green leaning towards valid. A dark-colored cell is an invalid cell, represented with a value of 1.





				
	$c \in \mathcal{C}_V$	$c \in \mathcal{C}_I$	$c \in \mathcal{C}_T$	$1 - (t\%1)$ $t\%1$
$ c = 1$	0	1	1	$A \cdot (1 - (t\%1)) + B \cdot (t\%1)$
$ c = 2$	[0,0]	[0,1]	[1,0]	$[A \cdot (1 - (t\%1)) + B \cdot (t\%1), 0]$

Figure 14: A summary of cell representations for every $c \in \mathcal{C}$ and $|c| \in \{1, 2\}$

A single cell can also be represented with two values. For a two-value cell representation, there is an individual value for invalid cells such that one cell is represented by the proportions [terminal, invalid]. This makes the representation for terminal cells and invalid cells distinct. A single cell representation in theory merges these two vision grids into one, which differs from the research of Knecht *et al.* [KDW18]. Note that multi-value cell representations make a single cell be represented by more values, thus adding complexity to the input. Using multiple values to represent a single cell is in line with the approach of [KDW18]. To refer to multi-value cell representations, we will use $\forall c \in \mathcal{C}, |c| \in \{1, 2\}$ to denote what representation is used. Figure 14 shows a summary of what a representation of a cell looks like using different $|c|$ and different $c \in \mathcal{C}$. The amount of values of a state is calculated by the size of the vision grid as well as the number of values a single cell is represented.

$$|s| = |c| \times (vg_{right} + vg_{left} + 1) \times (vg_{down} + vg_{up} + 1)$$

where $|c|$ represents the amount of values needed to represent one state, and vg_x is the amount of cells the agent can look in direction x . In figure 15 you can see how a state is built up for different $|c|$: how different vision grids for different types of cells are used and combined to form a state.

An important thing to denote here is that the train tracks, even though the sliding window moves in full cells, can take a fractional value too. This is the case when a train is approaching: five timesteps before the train enters the environment, all the values of the cells have a value of 0.5, which is meant to be some kind of warning. This remains until the train leaves the environment. It is comparable to the way it works in the game, where there is a light signal notifying the player that a train will be entering the environment momentarily.

4.4 Action space

The action space \mathcal{A} is the set of all possible actions, which is defined as $\mathcal{A} = \{\text{up, left, down, right, still}\}$, which are zero-indexed respectively. As observed, $|\mathcal{A}| = 5$. The world \mathcal{W} could be seen as an (observable) gridworld of \mathcal{W}_w by \mathcal{W}_h . An action is defined as the move from a cell c to a cell c' which is adjacent according to the grid world.

4.5 Transition function

The transition function is defined as $T(c'|c, a)$, which denotes the transition from cell c to cell c' using action a . Here, it requires that $c \in \mathcal{C}_V$. This makes sense because an agent cannot make an

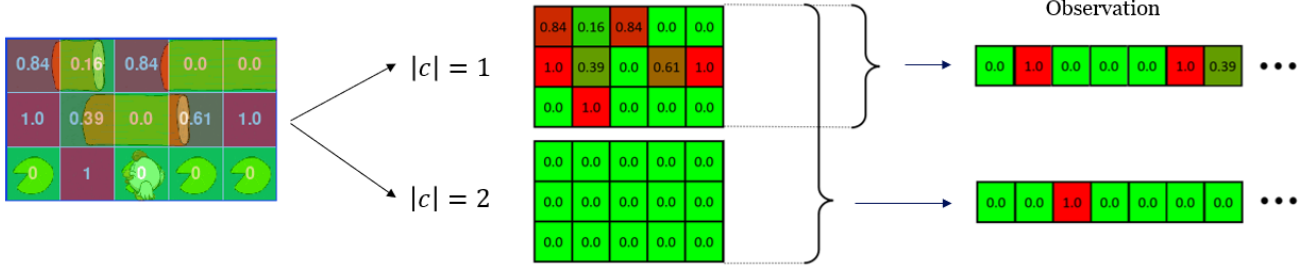


Figure 15: The way a state is built up. Here you can see the two different vision grids, each representing terminal states, and invalid states respectively. This vision grid box is the same vision grid box seen in figure 13. You can see it depends on what $|c|$ is used, for how a state is built up. The general way is for each grid (top to bottom), each row (bottom to top), and each cell (left to right) to take the value and place it after the other. For example, for $|c| = 2$, the first 2 states are the bottom left corners of each grid (top to bottom). The next one would be the cells located right of that corner cell. The cells are color-coded based on the proportions of termination or invalidity.

action from a terminal cell and cannot be in an invalid cell. An agent making a transition to an invalid cell will follow the rule:

$$\forall c \in \mathcal{C}_V; a \in \mathcal{A}; c' \in \mathcal{C}_I, T(c'|c, a) \doteq T(c|c, a)$$

A transition to an invalid state is the same as standing still. This, however, **does not apply to the reward function**. Note that walking outside of the environment is not considered invalid, but is rather considered as truncation. Truncation is also considered when the environment unexpectedly ends, due to a time- or step limit.

Furthermore, the transition function $T(c'|c, a)$ is deterministic: there is no random chance involved in the transition from cell c to c' . Note that there is no invalid cell present in any of the non-static layers: $\{c|c \in \mathcal{C}_I \wedge c \in \mathcal{C}_{\text{Road}} \cup \mathcal{C}_{\text{Logs}} \cup \mathcal{C}_{\text{Rail}}\} = \emptyset$. There is however one special case: the Logs layer. While the agent is on top of the logs agent on this layer, the agent moves along. This means that standing still moves the agent along with the log. Therefore, you could argue that

$$\forall c \in \{c|c \in \mathcal{C}_{\text{Logs}} \wedge c \in \mathcal{C}_V\}, \Pr(c, a = \text{still}, c') \neq \Pr(c, a = \text{still}, c)$$

where $\Pr(c, a, c')$ denotes the transition probability. It thus is possible to 'move' to another cell whilst standing still. This primarily affects the surrounding cell space, since standing still on a log can only end the environment if and only if the log moves off-screen, for which truncation applies.

4.6 Reward function

The reward function is defined as $r(c, a, c')$, which denotes the reward obtained by using action a to transition from cell c to cell c' . The reward function is defined in chronological order:

$$r(c, a, c') = \begin{cases} -100, & \text{if } c' \text{ is terminal} \\ -100, & \text{if } c' \text{ is truncated} \\ -100, & \text{if } c' \text{ is invalid} \\ 10, & \text{if } a = \text{up and to an unvisited (new) layer} \\ \text{otherwise } r(a) & \begin{cases} 2, & \text{if } a = \text{up} \\ -1, & \text{if } a = \text{left} \\ -4, & \text{if } a = \text{down} \\ -1, & \text{if } a = \text{right} \\ 0, & \text{if } a = \text{still} \end{cases} \end{cases}$$

As you can see, a reward of -100 is applied when the agent enters a terminal cell, the environment is truncated, or when the agent tries to move to an invalid cell. The environment can truncate in two ways: the agent walking outside of the environment or the agent exceeding the maximum amount of steps allowed to make in the environment. A reward of 10 is applied when the agent moves to a new unvisited layer. This means that the agent sets a new high score. If none of the previously mentioned cases apply, a step reward is yielded. Upwards yields 2 points, downwards yields -4 points, left and right both yield -1, and standing still yields 0.

Something interesting to denote here is that the rewards are not cumulative. Standing in a cell and moving upwards, to a new unvisited layer, will yield a reward of 10, and not $10 + 2$ (step reward). Another case that could occur is that the same cell transition could yield different rewards. Expanding on this theory, for a combination of static layers, the same state transition could yield different rewards: moving upwards to a new unvisited layer will yield a reward of 10. If the agent then moves downwards, yielding -4, and upwards again, it will only yield a reward of 2.

The thought behind this reward shaping is to promote moving upwards tremendously, whilst moving downwards is punished. It is also punished to move to a bush state: the agent has to learn to stand still for bush states instead of moving towards one. Furthermore, patience is more promoted than moving around: the step rewards for left and right are -1 on purpose, such that when the agent does not 'see a clear path', it will stand still instead of moving left and right. We assume this would help with the logs section.

4.7 Initial state distribution

The initial state distribution must be linked to the generation of \mathcal{W} . The agent always starts in a $c \in \mathcal{C}_V$, and the collection $\{L_1, L_2, L_3\}$ remains the same for each episode: a **Bush** layer with $d_{\text{Bush}} = 1$, followed by two **Custom** layers which are **Empty** layers with bush states $c \in \mathcal{C}_I$ on both sides. Sections are generated until the sum of layers exceeds \mathcal{W}_h .

Together, the tuple $(\mathcal{S}, \mathcal{A}, T(c'|c, a), r(c, a, c'))$ make up a sequential decision environment which can be interacted with by a reinforcement learning agent.

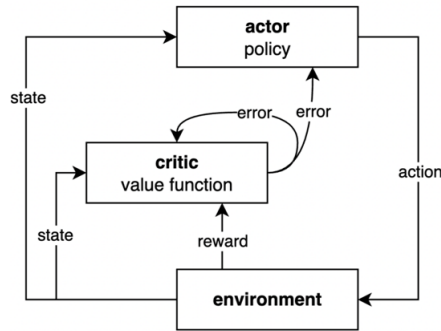


Figure 16: Agent-environment loop for an Actor-Critic model. Source: [OG22]

4.8 Reinforcement learning model

For this environment, we will be using a Proximal Policy Optimization (PPO) model. This is an Actor-Critic model, meaning it makes use of two networks: an actor (policy) network and a critic (value function) network.

How this agent-environment loop functions is as follows: the agent observes a state from the environment. In this case, it is a vector of values concerning the vision grid and the number of values that represent a single cell. This observation is then fed into the policy network. This is a network consisting of an input layer the size of the observation vector, followed by two hidden layers, consisting of 32 and 16 nodes each and using the activation function \tanh , and an output layer of size $|\mathcal{A}|$. This outputs scores for each action. The action with the highest score gets picked and simulated in the environment. The agent then observes the new observation and the reward received from performing that action. This is fed into the value function network. This network, similarly to the policy network, also consists of an input layer the size of the observation vector, with two hidden layers each 32 nodes and 16 nodes with an activation function \tanh . However, the output layer consists of a single node, namely the node that predicts the advantage \hat{A}_t . Based on \hat{A}_t , both the policy network and the value network gets updated. As explained in section 3.2.3, $\hat{A}_t > 0$ will update both the policy network and the value network positively, such that the policy network scores the action next time better, and the value function network makes a better prediction for the advantage. Similarly, $\hat{A}_t < 0$ will update the policy network and value network negatively, such that the policy network scores the action next time worse and makes it less likely that the policy network chooses this action next time. The value function network will also update itself accordingly. These updates are not too significant themselves, since it uses a policy gradient with a clip range $\epsilon = 0.2$. For full details of what hyperparameters and what differs from default hyperparameters, see appendix B.

The world gets updated simultaneously with the agent. This way, the agent has to anticipate in the world what will happen before it happens. In the majority of the cases, no anticipation is not punished, but there are edge cases: a cell could still be valid (i.e. the value for invalidity is ≤ 0.5) but could not apply anymore when the agent acted whilst the world updated.

5 Results

In order to create the interaction between the agent and environment, we use the OpenAI library Gym [BCP⁺16]. Gym is based on making environments operable for reinforcement learning agents. They implement easy access to train reinforcement learning agents using the traditional agent-environment loop. It includes many features, such as a base template, multiple observation- and action spaces, and easy creation and sharing options. It also provides the use of other libraries in combination with Gym, since Gym is highly compatible with for instance Stable Baselines 3 [RHG⁺21], which we will be using to create the PPO model.

Along the many parameters, the parameters that remain the same are the learning rate $\alpha = 3 \cdot 10^{-4}$, the clip range $\epsilon = 0.2$, and the discount factor $\gamma = 0.99$. An overview of all parameters used during the experiments for the world and the reinforcement learning agent can be found in appendix B. Furthermore, the rollout is set to $n_steps = 2048$ and the batch size is set to $batch_size = 64$. Additionally, the policy used for picking actions is deterministic, since an optimal policy π^* of a fully learned agent is assumed to be the greedy policy.

We will be conducting several experiments, with every experiment consisting of testing four different agents, which differ in vision grid size. These vision grid sizes are $vg_{down} = 0 \wedge vg_{right} = vg_{left} = vg_{up} = v$ with $1 \leq v \leq 4$. We will refer to an agent using a vision grid size of $vg_{down} = 0 \wedge vg_{right} = vg_{left} = vg_{up} = v$ with $1 \leq v \leq 4$ as $vg = v$. Furthermore, every experiment is run on the world generation described in algorithm 1 (a more elaborate version, with more insight into probabilities, can be found in 2). When the environment terminates or truncates, \mathcal{W} is created from scratch. Lastly, a world could have a maximum age. This means in essence that the agent is allowed a maximum number of actions in the world. After this amount, the environment truncates. This can be compared with a time limit within an environment. This is added due to the environment being endless, and an agent getting stuck will never terminate the environment. Implementing a maximum number of moves forces the agent to maximize the reward within a certain amount of time steps. The $max_age = 500$ for each experiment. For the evaluation of the model after training, a maximum age of $max_age = 5000$ is used such that the agent is not limited to any time limits in their performance, though still stopping agents that might get stuck.

Due to the crowdedness of many figures, some of the figures related to the training have been moved to appendix D. The results are discussed in the sections, however.

5.1 No step reward and single-value cell representation

The first experiment we did was not rewarding steps. Given the reward function from section 4.6, the alternation made here was that $\forall a \in \mathcal{A}, r(a) = 0$. The only way the agent can get a positive reward is by setting a new highscore, thus the objective function of the reinforcement learning agent is directly correlated to the reward obtained for each action. Furthermore, it allowed the agent more patience as to how to navigate the environment, since taking more steps is now not disadvantaging.

Training

Starting, we are going to discuss the training results of these agents. Figure 17 shows the moving average length of an episode during training (a), the moving average cumulative reward of an episode during training (b), and the highscore per episode obtained during training (c) respectively. The models are color-coded, and consistent color coding is used for the remaining plots where possible. From figure 17a, it can be observed that the average episode length lies around the range of $[250,300]$, with the agents with $vg = 2$ and $vg = 4$ lying around the lower part of that range whilst the agents with $vg = 1$ and $vg = 3$ lie around the higher part. Out of a maximum of 500 steps, these agents seem to not use all steps before getting terminated or truncated, but due to these plots being moving averages and a large portion of episodes also ending quickly, this average is brought down significantly. The peak observed for $vg = 4$ in the first $0.2 \cdot 10^6$ shows that the agent’s policy was changed from standing still and playing safe to moving more forward and taking more risks. Furthermore, from figure 17b, it can be observed that the moving average cumulative reward differs per type of vision grid significantly. For smaller vision grids, this average lies lower and increases in vision grid size. It peaks at a $vg = 3$, after which the cumulative average reward for $vg = 4$ reduces. Additionally, in figure 17c can also be observed what the moving average highscore is for various vision grids during training. As you can observe, the highscore for agents with a smaller vision grid lies a lot lower than the highscore for agents with a larger vision grid. The agents with $vg = 3$ and $vg = 4$ can be argued to perform quite similarly.

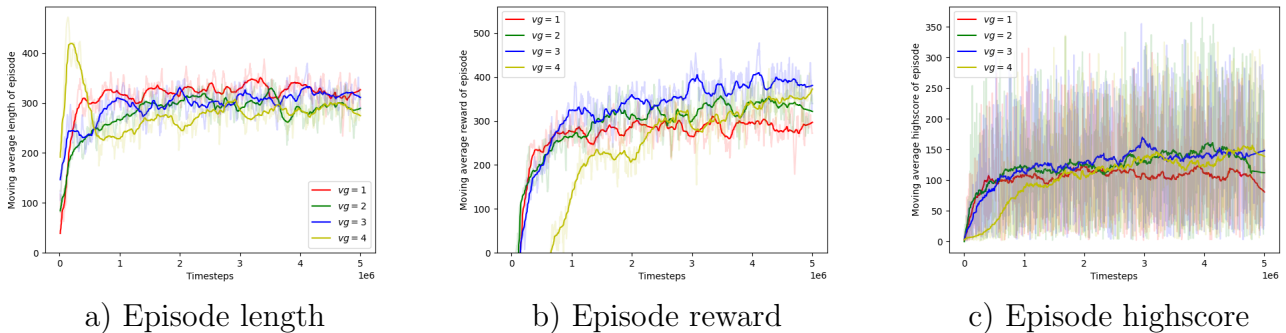


Figure 17: The moving average episode length (a), moving average episode cumulative reward (b), and highscore per episode (c) for agents with $1 \leq vg \leq 4$ during training without step reward. The moving average episode length lies in the range of $[250,300]$. Furthermore, it can be observed that smaller vision grids on average yield lower cumulative rewards whilst larger vision grids yield higher rewards. Lastly, the agent with $vg = 1$ on average yields lower highscores than other models.

In figure 34, it can be observed what the proportions of terminations and truncations are during training. Logically, these proportions add up to 1, thus the plots mirror each other on the x-axis at 0.5. This plot tells us whether the agent learns from the environment or gets stuck, with the implication that the environment more often ends unexpectedly (i.e. truncation), or that the agent has not learned enough, and therefore ends the environment due to making the wrong actions (i.e. termination). From this plot, it can be observed how the proportion of truncation gradually increases during training, with smaller vision grids having a higher truncation proportion shift than larger vision grids. Also, here can be observed that the policy of the agent with $vg = 4$ for the first $0.2 \cdot 10^6$ was playing safe since the majority of deaths are due to truncation.

Additionally, to make clear why these proportions between truncation and termination shift and how they shift, we looked into the proportions of deaths on types of layers during training. This can be seen in figure 35. You can see how the proportions are all quite similar, with the exception that the proportion for deaths on the **Bush** layer is significantly higher for the agent with $vg = 1$. Similarly, the proportion of the non-static layer **Rail** seems to be lying around 0.2 for every agent. Lastly, the dip seen in the graph of the agent with $vg = 4$ for the proportion of deaths on the **Logs** layer does not have to do with better performance on the **Logs** layer, but rather a worse performance on other layers, due to standing still and playing safe.

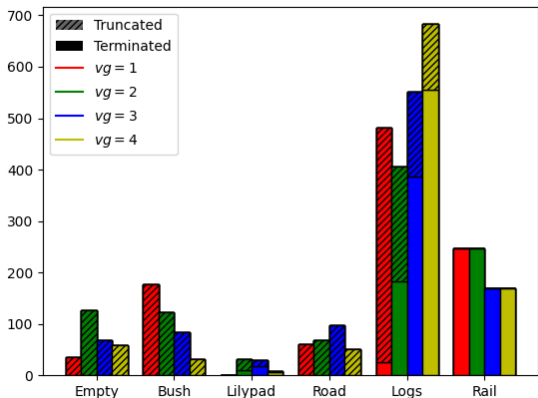


Figure 18: The type of layer the agent ended the episode on for agents without step reward. This can be due to termination (solid bar) or truncation (textured bar). The colors represent the various vision grid sizes. It is clear that the **Logs** layer had the highest death count

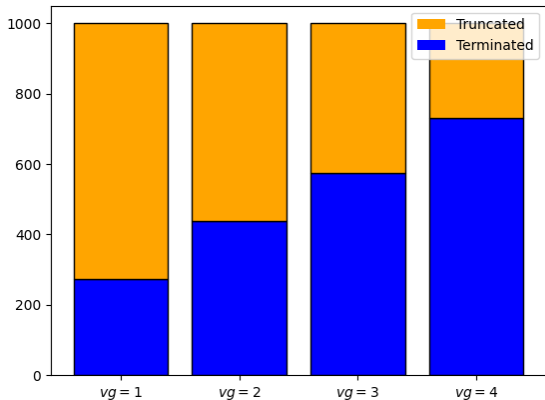


Figure 19: The type of death that ended the episode for agents without step reward. Here you can see how the termination (blue) and truncation (orange) ratios over 1000 episodes are distributed, with more truncation on smaller vision grids and more termination on larger vision grids

Evaluation

This section goes over the evaluation of the plots. As explained, the evaluation is done by running an agent on 1000 episodes, with $max_age = 5000$. Beginning with the first evaluation plot: we observed what layer the agent was on when the episode ended. This can be seen in figure 18. The episode ends can be both due to termination or truncation, marked as a solid bar or textured bar respectively. Furthermore, you can see that each category, i.e. layers, has four bars, representing the different vision grid sizes used. As you can see from this figure, most episodes ended whilst the agent was on the **Logs** layer. For smaller vision grids, this often is due to truncation. This can both mean a time limit, but it is more likely the agent drove along with the log off the screen, resulting in truncation. Additionally, of all non-static layers, the agent died the least on the **Road** layer. It is noticeable how many deaths there are on the static layers, especially for smaller vision grids. For the **Empty** and **Bush** layers, the only way the episode can end is due to exceeding the maximum age. This meant that the agent got stuck on both the **Empty** and **Bush** layer quite often, with more

truncations by agents with a smaller vision grid than larger ones. Lastly, the overwhelming majority of episodes that end on the **Rail** layer are terminations, meaning that the agent on average has the most difficulty with anticipating when to cross this layer.

Continuing, to make it convenient, we have added the types of death as totals per different vision grid sizes from figure 18. This can be seen in 19. As you can see, an agent with a smaller vision grid often results in truncating the environment, whilst agents with a larger vision grid shift to more terminations. This can be explained using several reasons. First of all, for agents with a smaller vision grid, the agent is limited by its vision. In situations where the agent gets stuck behind a row of bushes and has to maneuver around it, the agent is simply incapable of doing so. Furthermore, by not including a heuristic action reward for patience, the agent is too impatient on the **Logs** layer. Agents with a larger vision grid have to deal with the complexity of state representations. This disallowed good generalization of a state by the policy to pick a good action. In other words, too much information causes the agent to be unable to decide what action is considered good. Though observed from figures 17a and 17b, you can see that both curves for episode length and episode reward have converged respectively.

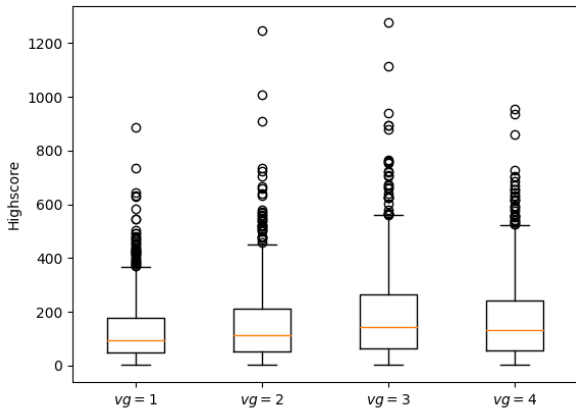


Figure 20: Boxplots of highscores of various vision grids for the agents without step reward. The boxplots show a relatively low IQR with many outliers.

Vision grid size	Mean \pm SD	Median
$vg = 1$	128.88 ± 110.76	95
$vg = 2$	151.90 ± 138.11	112
$vg = 3$	186.52 ± 162.50	145
$vg = 4$	172.64 ± 149.04	133.5

Table 1: Summary of the results seen in figure 20

Lastly, from figure 20, four boxplots can be observed, each showing the highscores respective to the four vision grids over 1000 runs. Next to this plot, table 1 can be observed showing the average reward as well as the standard deviation for each vision grid. Overall, it can be observed that agents, no matter the size of the vision grid, have a significant number of outliers. Furthermore, the interquartile range (IQR) lies quite low, compared to the many high-valued outliers. This is due to the proportion of runs ending in low highscores being far larger. The upper tail of each vision grid is therefore also longer than the lower tail. Nonetheless, the agent with a vision grid of $vg = 3$ showed the best results, with a median episode highscore of 145. The average also lies higher than the other agents, though due to the number of outliers, this is not a reliable basis to

make assumptions.

5.2 With step reward and single-value cell representation

The second experiment we did, was testing the effects of step rewards in contrast to no step rewards. The single-value cell representation is still used in this experiment. This reward function, as described, promotes moving forward and punishing moving backward, as well as reward patience by standing still, instead of moving around. With this experiment, it can also be observed whether this change of feedback makes a significant change in the way vision grids are utilized.

Training

First, we will address the moving average length, the moving average cumulative reward, and the highscore per episode shown during training with step reward. Figure 21 shows exactly that respectively. Figure 21a shows that the average length of an episode lies slightly higher than the agents without a reward, with the converged values lying in the range [260,330]. The length of the episodes of agents with a larger vision grid is shorter than that of smaller vision grids. The moving average cumulative reward plot, displayed in figure 21b, shows that the values converge for each agent within the range of [320,460]. Now, since the reward functions differ, this range lies higher than the range without step reward. However, the trend of the smallest vision grid converging to the lowest cumulative average reward is equivalent. A clear second lowest agent can be observed, namely the largest vision grid. Additionally, we discuss the moving average highscore for various vision grids during training, seen in figure 21c. It shows that the agents with the largest vision grid and the agent with the smallest vision grid have the second lowest and lowest convergence of average highscore respectively. The average values of the agents lie slightly higher than the agents without step reward, but there is no significant difference. The agent with the vision grid size $vg = 3$ outperforms the other agents at the end of training. However, it can be argued that, even with smoothing applied, the graphs have deep peaks and valleys, and thus the agent might have ended lucky.

Second, we will discuss the proportions of the types of deaths that occurred during training. From figure 36 it can be observed that again, the smaller the vision grid, the higher the proportion for truncation as a reason for ending the environment. However, only the agent with the smallest vision grid had a higher proportion of truncations than terminations at the end of the training, whilst here also the agent with the vision grid size $vg = 2$ shares this result. This means that the majority of their deaths happen unexpectedly. The dip seen with the agent with vision grid size $vg = 4$ has also decreased significantly but still shows.

The last training results plot is shown in figure 37. This plot shows similar results as figure 35: the **Logs** layer remains the most deadly layer, the **Train** remains a steady death proportion of 0.2 over all agents, and the **Bush** layer is still troublesome for agents with a smaller vision grid. It thus seems as though the agent with $vg = 2$ gets stuck more often due to the added step reward.

Evaluation

Starting with the evaluation, we will begin to address figure 22. This figure shows the count of

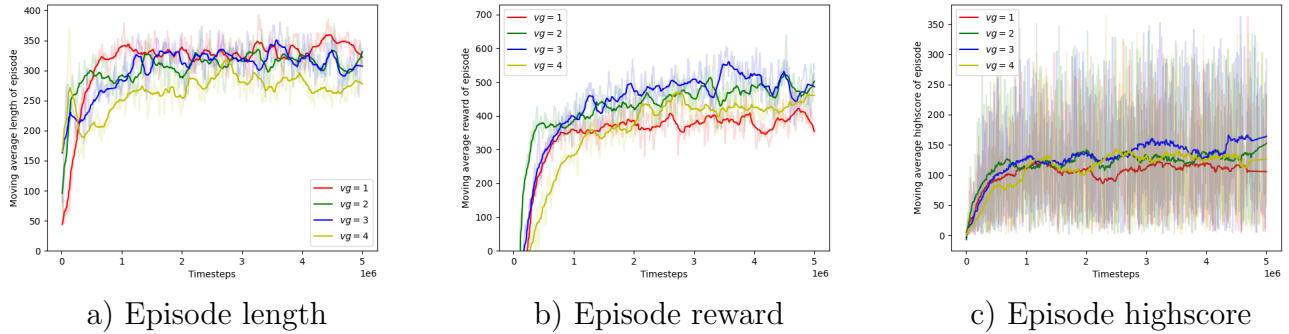


Figure 21: The moving average episode length (a), moving average episode cumulative reward (b), and highscore per episode (c) for agents with $1 \leq vg \leq 4$ during training with step reward. The moving average episode length lies in the range of $[260, 330]$. Furthermore, it can be observed that the agents with the smallest vision grid as well as the largest vision grid on average yield lower cumulative rewards whilst the remaining sizes yield higher rewards. Lastly, the agent with vision grid size $vg = 3$ outperforms the other agents. The trend of the largest and smallest vision grids performing the worst continues.

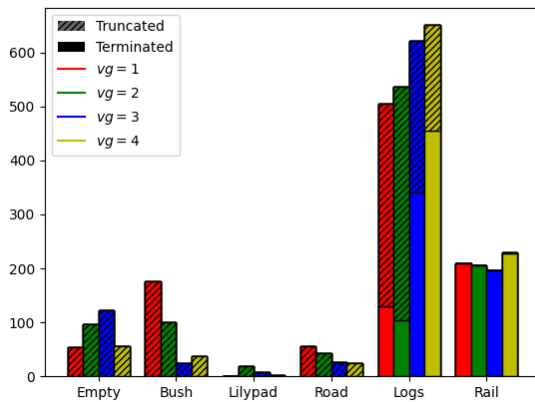


Figure 22: The type of layer the agent ended the episode on with step reward. This can be due to termination (solid bar) or truncation (textured bar). The amount of **Road** truncations decreased in comparison to figure 18.

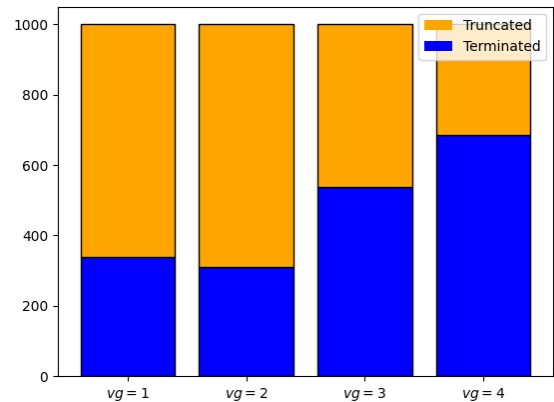


Figure 23: The type of death that ended the episode. Here you can see how the termination (blue) and truncation (orange) ratios over 1000 episodes are distributed. The agent with vision grid size $vg = 2$ shows an interesting division and does not follow the seen trend in figure 19.

the types of layers the agent ended the environment on, separated by vision grid size and split up by truncation and termination. As you can see, the majority of the episodes ended on the **Logs** layer. Along with that, the most truncation happens by agents with a smaller vision grid size. The **Bush** layer is still prone to get stuck behind, with the **Empty** layer contributing here as well. The **Rail** layer remains on a steady approximately 20% environment end rate for each agent. The performance with reward shaping seems to decrease the amount of episode ends on the **Road** layer,

but due to the environment endings merely consisting of truncations, it cannot be said that the step reward has contributed to performance improvement.

Moving on to the total amount of environment truncations and terminations done by each agent, which can be seen in figure 23. It is remarkable how the amount of terminations lies higher for $vg = 1$ than for the same agent without step reward. Additionally, the agent with vision grid size $vg = 2$ has a higher truncation rate than the agent with vision grid size $vg = 1$. The trend of more terminations the larger the vision grid continues here, which is equivalent to the agents without step reward.

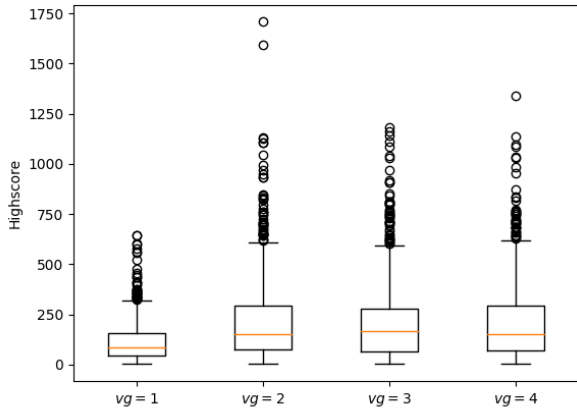


Figure 24: Boxplots of highscores of various vision grids for the agents with step reward. The boxplots show significant outliers.

Vision grid size	Mean \pm SD	Median
$vg = 1$	113.77 ± 94.82	85
$vg = 2$	214.52 ± 198.65	153
$vg = 3$	209.82 ± 190.54	165.5
$vg = 4$	208.44 ± 190.32	149.5

Table 2: Summary of the results seen in figure 24

Lastly, the highscores of the evaluation are visualized in a boxplot, and summarized in a table, which can be seen in figure 24 and table 2 respectively. As you can see, the boxplots again show that the IQR is relatively low compared with some extreme outliers. The highest measured evaluation episode came to a highscore of 1707, set by the agent with vision grid size $vg = 2$. Compared to figure 20, here you can see that the agent with vision grid size $vg = 1$ is lacking performance, compared with the other agents. Due to the extreme outliers of the agent with vision grid size $vg = 2$, it looks like this agent on average performed the best. Though, from the median of the highscores we would argue that the agent with vision grid size $vg = 3$ performed the best. From table 2, it can also be observed that the standard deviation of each agent is extremely high. This variance in performance is also seen back in table 1, thus it could be argued that this is due to the environment.

5.3 With step reward and double-value cell representation

The third experiment we did, was testing whether using multiple vision grids to denote different elements in the environment would help to increase performance. Hereby we refer to 4.3: we will be using $|c| = 2$, thus having two separate grids to denote proportions of termination and proportions

of invalid cells respectively. we want to test this due to invalid cells and terminal cells having the same representation. Even though $r(\cdot, \cdot, c \in \mathcal{C}_I) = r(\cdot, \cdot, c \in \mathcal{C}_T)$, the environment terminates for entering $c \in \mathcal{C}_T$ and not attempting to enter $c \in \mathcal{C}_I$. This may interfere with learning and could lead to inconsistencies while assessing states in the reinforcement learning model. From this experiment, the trade-off between the extra complexity of multiple vision grids and the performance can be observed. Lastly, it may also solve the problem of agents getting stuck behind invalid cells, since a different classification for similar representations can be done, and may grant anticipation.

Training

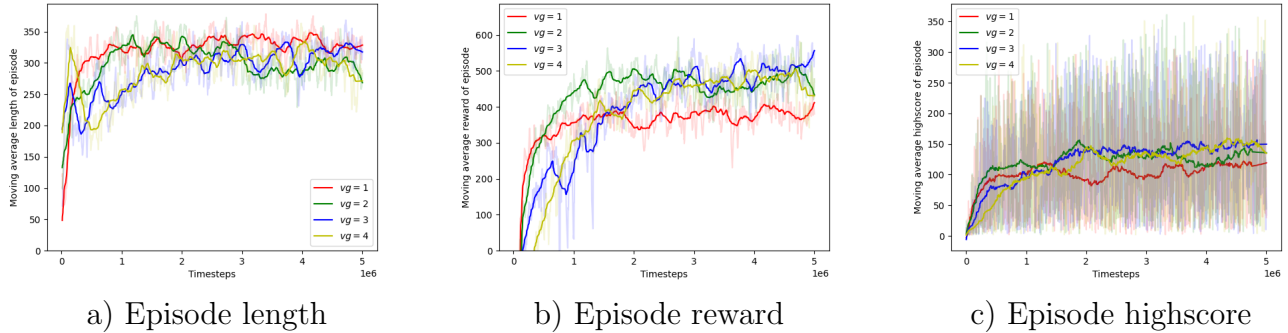


Figure 25: The moving average episode length (a), moving average episode cumulative reward (b), and highscore per episode (c) for agents with $1 \leq vg \leq 4$ during training with step reward and $|c| = 2$. The moving average episode length lies in the range of $[260,320]$. Furthermore, it can be observed that agents with the smallest vision grid yield lower cumulative rewards whilst the remaining sizes yield higher rewards. Lastly, the converged values lie around 130, except for the agent with the smallest vision grid.

Starting, the moving average length of an episode during training, the moving average cumulative reward of an episode during training, and episode highscores during training of agents with step reward and $|c| = 2$ can be seen in figure 25 respectively. Figure 25 shows that the range of the converged average episode lengths lies in the range $[260,320]$, with the agents with vision grid sizes $vg = 1$ and $vg = 3$ converging to larger average episode values than the remaining agents. Where it seems as though the agent with $vg = 4$ with step reward was struggling, this seems like it is not the case here. The small peak seen for this agent is similar to the agent with $vg = 4$ in figure 17c. Furthermore, figure 25b shows that the converged value for the cumulative reward lie in the range $[350,550]$, with the agent with vision grid size $vg = 1$ converging to the lowest value and the remaining agents converging to around the same value. To compare this to the experiment with single-value cell representations with step reward (since they use the same reward function), seen in figure 21b, the agents perform around the same, with the agent with $vg = 1$ lacking behind other agents. Continuing with the highscore convergence during training is seen in figure 25c, a clear distinguishment can be made, where agents with smaller vision grids converge to a lower average highscore. This hierarchy is also seen in figure 21c from the agents with a single-value cell representation. Similarly, here, the graphs contain lots of peaks and valleys, which tend to suggest that there is a lot of variance in the runs.

Moving on with the moving proportions of the types of deaths of the agents during training. These can be seen in 38. It can be observed that the proportion of truncations for the agent with vision grid size $vg = 4$ is significantly higher than for the same agent with a single-value cell representation. Similarly, the agent with vision grid size $vg = 2$ seems to be more gradually decreasing in the proportion of terminations and increasing in the proportion of truncations. That both agents with vision grid size $vg = 1$ and $vg = 2$ have a higher proportion of truncations than terminations, does not foretell much good.

That will be made clear with the last training plot: the proportions of layers the agent has ended the environment on. This can be seen in figure 39. The assumption that multiple vision grids would help reduce the number of truncations on `Bush` layers, cannot be seen back for the agents with vision grid size $vg = 1$ and $vg = 2$. The results shown in this figure are relatively similar to previous figures 37 and 35.

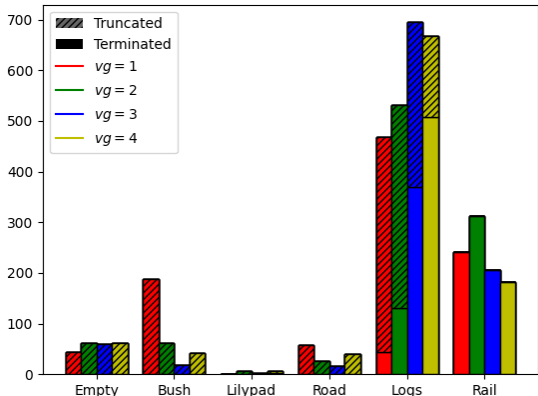


Figure 26: The type of layer the agent ended the episode on with step reward. This can be due to termination (solid bar) or truncation (textured bar). The amount of `Road` truncations decreased in comparison to figure 18.

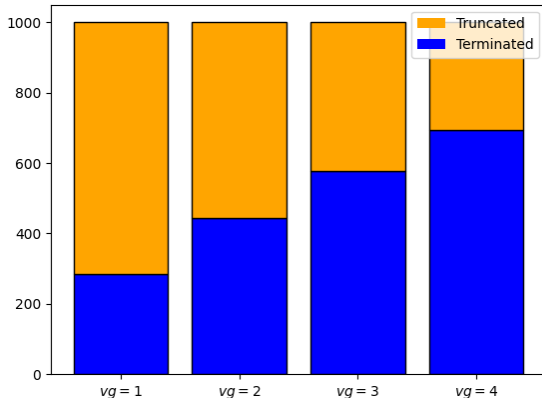


Figure 27: The type of death that ended the episode. Here you can see how the termination (blue) and truncation (orange) ratios are distributed. The agent with vision grid size $vg = 2$ shows an interesting division and does not follow the seen trend in figure 19.

Evaluation

Starting with the evaluation, figure 26 shows the count of each layer for which the episode ended per agent, along with details on whether it was due to truncation or termination. As you can see, the use of multiple vision grids did help to reduce the number of truncations on the `Bush` layer, except for the agent with $vg = 1$. Where we first saw a steady decrease, the agents now show an almost uniform amount of truncations on static layers (again, except for the agent with $vg = 1$ on the `Bush` layer.) For non-static layers, the values are more chaotically spread out. The `Rail` layer shows more variable results: where we first saw a value of around 200 episode ends for each

agent, we now see more variable values, with the agent with vision grid size $vg = 2$ having an exceptionally high value. The Logs layer remains difficult for all agents.

Looking at the total amount of terminations and truncations for each agent in figure 27, the values are highly comparable to figure 19, with a steady increase of the number of terminations for agents with a larger vision grid. The number of terminations for the agent with $vg = 1$ from this experiment lies lower than the number of terminations for the agent from the experiment with step reward and single-value cell representations. This implies that double-value cell representations did not help this agent. Similarly, but for $vg = 2$, this does seem to have an effect, but then again, the contrary could be argued due to the same agent with $vg = 2$ differing in result when comparing with step reward and no step reward.

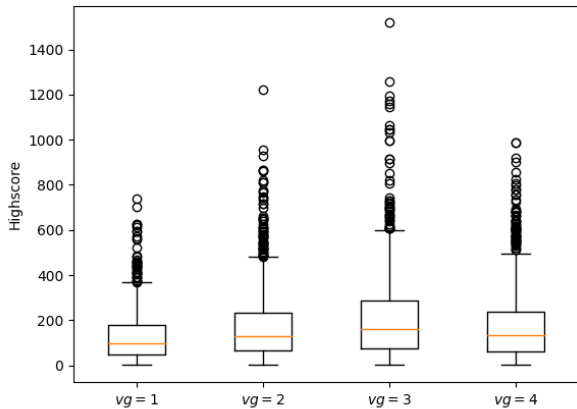


Figure 28: Boxplots of highscores of various vision grids for the agents with step reward. The boxplots show significant outliers.

Vision grid size	Mean \pm SD	Median
$vg = 1$	129.66 ± 110.69	99
$vg = 2$	177.02 ± 160.34	129
$vg = 3$	213.93 ± 193.98	161
$vg = 4$	181.07 ± 166.13	133.5

Table 3: Summary of the results seen in figure 28

Lastly, the evaluation of the highscores gained for each episode is shown in figure 28, along with a similar table 3. The hierarchy of the boxplots is in line with previous results: the agent with $vg = 3$ performs the best and the agent with $vg = 1$ performs the worst. The means of the agents with $vg = 1$ and $vg = 3$ increased. Furthermore, all medians are lower concerning table 2, except for the agent with $vg = 1$. From the training plot, we can already see that the double-value cell representation did not decrease the proportion of truncations on the **Bush** layer, thus this must be coincidental. However, it is noteworthy to mention that the median, although decreased for remaining agents, the agent with $vg = 1$ increased. This can be linked to more consistent good runs.

5.4 Comparison

In this section, an overview is given of the best-performing models for each experiment. A best-performing model is defined as the model with the highest median highscore during evaluation. Determining performance by median evaluation highscore, **the agents with $vg = 3$ performs**

the best in each experiment. Figure 29 show the training plots of each best-performing model.

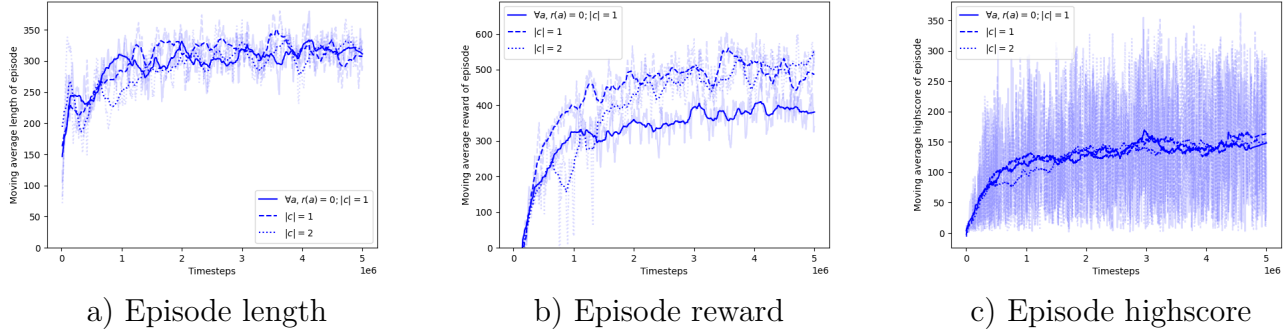


Figure 29: The moving average episode length (a), moving average episode cumulative reward (b), and highscore per episode (c) for the best-performing agents (all $vg = 3$) in each experiment. Each label corresponds to the attributes unique to that experiment. Average episode lengths remain similar for each agent. Intuitively, the average reward shows a clear distinction between the agent without step reward and the agents with step reward. Here, the slower convergence of the agent with $|c| = 2$ can also be observed. Figure c shows that the average highscore during training remains similar over all experiments for these agents.

The moving average episode length seen in figure 29a shows barely any difference among the agents. Figure 29b however, does show a significant difference. Due to not rewarding actions, it is intuitive to say that the agent without step reward converges to a lower average reward than the agents with step reward. You can also see a slower convergence of the agent with $|c| = 2$ over the agent with $|c| = 1$. Lastly, the average highscore during training observed in figure 29c remains similar over all experiments. Here you can also see that the agent with $|c| = 2$ converges slower than the agents with $|c| = 1$.

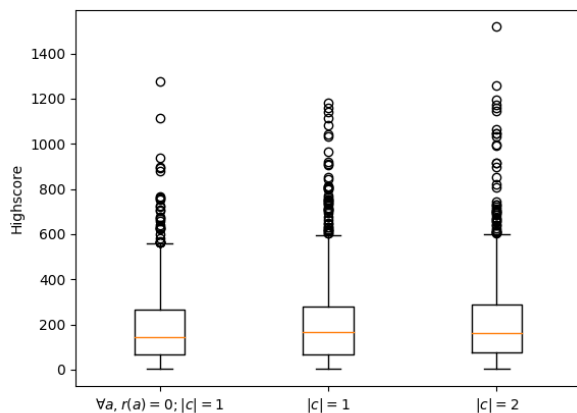


Figure 30: Comparison of best-performing agents (all $vg = 3$) in terms of evaluation of 1000 episode highscores

Vision grid size	Mean \pm SD	Median
$\forall a, r(a) = 0; c = 1$	186.52 ± 162.50	145
$ c = 1$	209.82 ± 190.54	165.5
$ c = 2$	213.93 ± 193.98	161

Table 4: Summary of the results seen in figure 30

The evaluation of the highscores of 1000 episodes, seen in figure 30 and table 4, show that agents with step reward lie closer together in terms of average and median highscores than the agent without step reward. The mean of the agent with $|c| = 2$ lies higher than the agent with $|c| = 1$, however so does the standard deviation. The mean lies just a little higher due to an incredibly high outlier.

5.5 SHAP analysis

With this empirical experiment, we wanted to see whether a connection between action prediction and feature impact can be observed in terms of the closeness of cells relative to the agent. We have done this by comparing visual agent playouts and executing the SHAP values parallel to the current observation. Per observation, the agent generates $|\mathcal{A}|$ heatmaps, one for each action $a \in \mathcal{A}$. These heatmaps show the contributed impact of each cell value on the predicted outcome a . It should be mentioned that the sum of impact values does not account for which action is taken, but is merely a measure to see why a certain action is important.

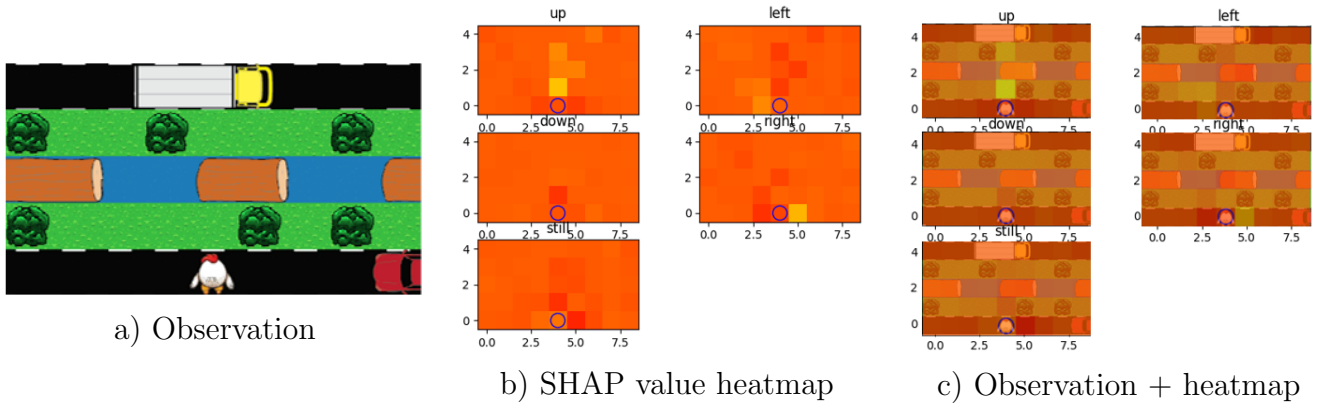


Figure 31: An example of a scenario analyzed with SHAP. The scenario is the observation of the agent (a). For each action, impact values are constructed which show how much influence each of the cell’s values was on the predicted output $a \in \mathcal{A}$ (b). The colors of the heatmap show the impact values, with darker colors representing negative impact, orange representing close to zero impact, and lighter colors representing positive impact. Impact values decay the further the cell lies from the agent. It can be argued that $a = \text{up}$ has a directional decay and set out a trajectory to follow, which can be seen when applying the SHAP values as an overlay to the scenario (c).

We assumed that the impact of cell values gradually decrease and that a directional decay could be possible. What can be observed is that the values do decay the further the cell is relative to the agent. For a single scenario with an agent with step reward and $vg = 4$ is observed in figure 31. From this scenario, you can see that the agent has a slight gradual decrease of impact the further the cells lie from the agent. This effect is especially seen in $a = \text{up}$. The effect of decay is visible for cells lying directly in front of the agent, and generally, cells that are neighboring the cell the agent is currently in. There is some form of directional decrease happening: you can see that for $a = \text{up}$, the agent tends to move right since the log (consisting of 2 parts) is slightly to the right of the agent. One could argue that, based on the trajectory of cells that make up the most impactful cells, the agent has a plan and thus a course of action. This only happens when there is no clear course of

action: the observation is complex with the number of cars, logs, and bushes. Once there is a clear path, especially for moving forward, the impact of the single neighboring cell is incredibly high. For example, as seen in figure 32, the agent cannot move forward. It is observed that neighboring cells for left and right also contribute much to the respective action. The action $a = \text{right}$ shows a light decaying pathway as well. When $a = \text{right}$ is chosen by the agent, the next scenario shows that a clear non-obstructive path is revealed, and the impact of the neighboring cell for $a = \text{up}$ is incredibly high.

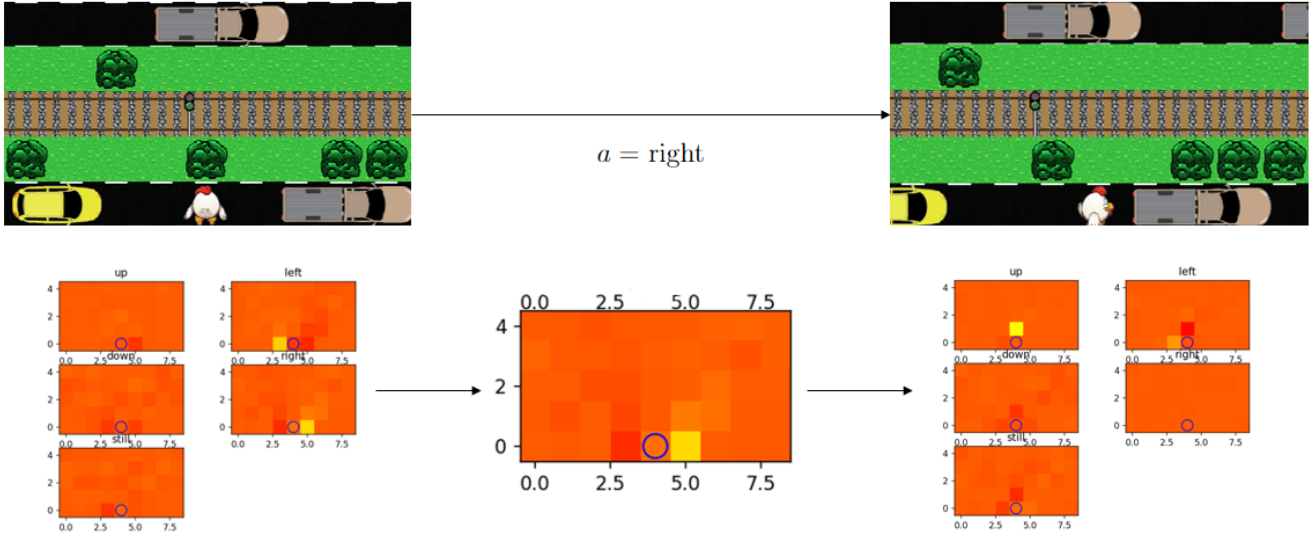


Figure 32: Two consecutive scenarios, where the agent is obstructed by a $c \in \mathcal{C}_I$ for $a = \text{up}$, but has two ways around this: $a = \text{left}$ and $a = \text{right}$. The agent chooses $a = \text{right}$, and its SHAP impact values show a decaying pathway leaning towards the right. Once the action is taken, the action $a = \text{up}$ is no longer obstructed and the impact of the neighboring cell in said respective direction is incredibly high.

6 Discussion

In this project, we have built our own environment. This environment is based on the game *Crossy Road*, and we have tried to replicate it as similarly as possible. This is due to us being able to receive much more information about the environment, without the uncertainty added of convolutional feature extraction via the real game. Furthermore, we wanted to relate results from this experiment to the game *Crossy Road*, and therefore we had to make assumptions about certain world generation values. This lead to bias in the environment in the attempt of recreation, since all parameters cannot be estimated correctly. For instance, the bias in the world generation can be seen back at how sections are generated. Road sections are usually longer, due to a higher θ_{Road} value, whilst Logs sections are a bit shorter. Furthermore, Lilypad layers are rarely generated, and thus terminating or truncating on these layers rarely happens. Although we believe we got to similar world generations, it did not help the agent to learn. The bias in the way the environment is built has, what we believe, a direct effect on the performance of the agents. The amount of variance for each agent

during evaluation is significant. This has had us believe that environments can be 'lucky' generations. Even though we assumed 1000 runs would resolve this variance, it did not have much effect.

Results were variable, and even though correlations could be made between how the agent terminated or truncated on a specific layer, it is not clear why it kept having difficulty with that layer. For instance, for the **Logs** layer, the amount of terminations and truncations merely vary between experiments. A reason for this could be because the value of the cell the agent currently stands on decreases, whilst its neighbour increases. Once condition 2 is met, the agent is automatically transferred, but this is not something the agent specifically learns since it does not have a perception of what type of layer it is. The agent does not know that on the **Logs** layer, they get moved along, and therefore it should not be learned similarly. For further research, we would let the agent train on variable worlds, such as a world where dense **Logs** sections are possible. This could allow for better training and generalization. As a side note, we have used the old implementation of Gym. At the time of research, the new distribution Gymnasium by Farama [TTK+23] existed, but was not compatible with Stable Baselines 3 [RHG+21]. Currently, the distribution Gymnasium is compatible with Stable Baselines 3. This allows for the parameter *info* to be returned, and future distributions of Stable Baselines 3 could utilize this. With this option, the agent could know what layer it currently is on, and this would open new possibilities for research.

Due to the advantage being worse predicted for a valid move upon truncation, the agent might learn inconsistencies. The reward for truncation is -100, even when the time limit is reached. This may work in environments where the environment is not endless: this forces the agent to find the optimal policy for which it also has to take into account not to take too much time and thus too many actions. In this environment, however, a valid move, or at least an expected valid move is being punished because of a time limit being reached. This could interfere with current knowledge, and the advantage function would be negatively influenced. Instead, we should have yielded the reward that the agent should have received and then ended the environment. This would not negatively influence the advantage function and might have improved consistency.

The agents all have similar settings, with the same network architecture but different input layers. These input layers vary from 15 nodes for the agents with a single-value cell representation and $vg = 1$, to 90 nodes for the agents with a double-value cell representation and $vg = 2$. This major difference is not being captured in a more complex network architecture. Even though every agent has been trained on 5×10^6 time steps, which is more than the agents with the smallest vision grids needed, it seems to not always be enough for agents with larger vision grids. A next time, we would use larger architectures that match the complexity of the input layer as a base, such that the requirements are at least met for the agents with a larger vision grid.

Additionally, the perspective for the agent on the **Logs** layer could be adjusted such that the agent has the same offset from the **Logs** layer it stands on in its vision grid values. This matches the intuitive thought that their vision grid exactly matches seen from its perspective, but could open new problems since it still is not aware that it stands on the **Logs** layer.

Something we wanted to try but was out of the scope of this research was to apply Monte Carlo Tree Search (MCTS). The assumption here is that MCTS might indirectly find that the agent

should avoid difficult situations beforehand. These include last-second actions moving forward on dangerous, edge-environment-related situations. The environment is reversible, in the sense that you can 'save' and 'load' an environment state along with its respective values.

We did an analysis of the model's predictions by utilizing the Shapley explainable AI method. We have done this for agents with $|c| = 1$, but due to time limits, we did not implement the tools to analyze agents with $|c| = 2$. This is an interesting research topic to continue in, and possibly do a more in-depth analysis of various complex cases.

Lastly, future work can consist of adjusting the environment to either make it complete or tune it. We left out some key elements of the game *Crossy Road*. These include a time limit for moving forwards and randomly appearing coins. These can be added to manipulate the agent moving forward and guide the agent in certain ways for a rewarding coin respectively. Furthermore, the decay values θ are now set such that more per section more Road layers are generated than Logs layers. One could argue that because more Road layers were generated, the number of encounters with states primarily made up of Road layers and thus combinations of such encountering values are increased, and thus learns how to navigate the Road layer better than for example the Logs layer. However, all this cannot be confirmed from the plots provided in this thesis: the plots only show the number of terminations and truncations on certain layers (for example figure 18), but not the number of crossings of these layers. If these values are included, proportions of difficulty per layer can be established by dividing the number of unsuccessful crossings by the total amount of crossings. With that, the effect of adjusting θ values can be observed.

7 Conclusion

To conclude, we have looked at both *Frogger* and *Crossy Road* in previous research. From this, we gained knowledge of difficulties with agents learning *Frogger* using multi-layer perceptrons as well as use cases of vision grids. We have implemented a self-made simplified version of *Crossy Road*, along with a clear explanation of world generation and layer generation. We have utilized vision grids along with a PPO reinforcement learning model to capture the environment and learn the environment respectively. With this being said, we discovered that the agents with a larger vision on average perform better. Agents with a small vision grid struggle with getting stuck behind obstacles. The larger the vision grid, the less this problem occurs. However, larger vision grids have the problem of generalization due to their large state representation. Furthermore, reward-shaping actions help the agent progress better. Using multiple vision grids as state representation only helps to get less stuck, but due to increased complexity, is a burden for agents with a bigger vision grid. The Logs section remains the most difficult layer to cross, which might be able to get solved using the aforementioned techniques. Lastly, this thesis shows how the observation space size and shape, made up of one or more vision grids of various sizes, influences learnability, in an evaluation on a self-made simplified *Crossy Road*. This information can be used to assess information importance for agents when determining what a state is in a state space: i.e. what does my agent necessarily need to know about this state?

References

- [BCP⁺16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- [Bel10] Richard Bellman. *Dynamic Programming*. Princeton University Press, 7 2010.
- [BHV⁺14] Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew D. Taylor, Daniel Kudenko, and Ann Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. 7 2014.
- [BM03] Andrew G. Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(1/2):41–77, 1 2003.
- [BS3] D Beechey, T.M.S. Smith, and Ö Şimşek. Explaining Reinforcement Learning with Shapley Values. *arXiv:2306.05810v1*, 6 2023.
- [con23] Wikipedia contributors. Frogger. *Wikipedia*, 5 2023.
- [DKG11] Sam Devlin, Daniel Kudenko, and Marek Grześ. AN EMPIRICAL STUDY OF POTENTIAL-BASED REWARD SHAPING AND ADVICE IN COMPLEX, MULTI-AGENT SYSTEMS. *Advances in Complex Systems*, 14(02):251–278, 4 2011.
- [EKB⁺14] Matthew Emigh, Evan Kriminger, Austin Brockmeier, Jose Principe, and Panos Pardalos. Reinforcement learning in video games using nearest neighbor interpolation and metric learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 8:1–1, 01 2014.
- [HTS⁺17] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments, 2017.
- [Kal12] Sreekanth Reddy Kallem. Artificial intelligence algorithms. *IOSR Journal of Computer Engineering*, 6(3):01–08, 2012.
- [KDW18] Stefan J. L. Knegt, Madalina M. Drugan, and Marco A. Wiering. Opponent Modelling in the Game of Tron using Reinforcement Learning. 1 2018.
- [Kon81] Sega/Gremlin Konami. Frogger - Videogame by Sega/Gremlin, 1981.
- [LL17] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. volume 30, pages 4768–4777, 12 2017.
- [Mai19] J. Maijers. Playing Frogger using Multi-step Q-learning - Student Theses Faculty of Science and Engineering, 2019.
- [MKS⁺13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv (Cornell University)*, 12 2013.

- [NHR99] Andrew Y. Ng, Daishi Harada, and Stuart D. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. pages 278–287, 6 1999.
- [O’G22] J. O’Grady. RL Agent - Softmax Actor-Critic, 2022.
- [PW96] Jing Peng and Ronald B Williams. Incremental multi-step Q-learning. *Machine Learning*, 22(1-3):283–290, 1 1996.
- [RHG⁺21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition*. MIT Press, 11 2018.
- [Sch17] John Schulman. Deep RL Bootcamp Lecture 5: Natural Policy Gradients, TRPO, PPO, 10 2017.
- [SLM⁺15] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, I, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv (Cornell University)*, 2 2015.
- [SWD⁺17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
- [TTK⁺23] Mark Towers, Jordan K Terry, Ariel Kwiatkowski, John U. Balis, Gianluca Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, and et al. Gymnasium. Jul 2023.
- [vdV18] S.T.M. van der Velde. Learning to play Frogger using Q-Learning - Student Theses Faculty of Science and Engineering, 2018.
- [Wha14] Hipster Whale. Crossy Road - Endless Arcade Hopper Game, 2014.

8 References images

For the creation of the environment, we have used several images. We do not claim ownership over these images and all credits should be considered to the creators. Hereby a list of sources of the images:

- [Bush tile by pixelartmaker.com](#)
- [Lilypad tile by pngitem.com](#)
- [Gravel base of the Rail layer by theappguruz.com](#)
- [Train and tracks by stock.adobe.com](#)
- [Chicken sprite by nl.depositphotos.com](#)
- [Remaining tiles and sprites by cs.cornell.edu](#)

A Symbol definitions

\mathcal{W}	The world of observable states
\mathcal{W}_w	The width of world \mathcal{W} in terms of states
\mathcal{W}_h	The height of world \mathcal{W} in terms of states
\mathcal{L}	An ordered list of infinite layers
L	A single layer
\mathcal{C}	The set of all cells
$\mathcal{R}(L)$	The representation of a layer L
c	A single cell
\mathcal{C}_V	The set of valid states
\mathcal{C}_I	The set of invalid states
\mathcal{C}_T	The set of terminal states
t	A certain time step
$\mathcal{O}(L(t))$	The observation of a layer L at time t
$type(L)$	Type of a layer L . This is within $types$
$types$	Set of all types a layer could take
\mathcal{L}_x	The set of layers with type $x \in types$
\mathcal{S}_x	The set of cells for each layer with type $x \in types$
d_x	The density on a layer, with $x \in \{\text{Bush, Lilypad}\}$
cs_x	The cycle speed for a layer, with $x \in \{\text{Road, Logs}\}$
$conf_x$	The configuration for a layer, with $x \in \{\text{Road, Logs}\}$
dir_x	The direction for a layer, with $x \in \{\text{Road, Logs, Rail}\}$
i_{Rail}	The interval for the train to appear
sp_{Rail}	The amount of cells the train moves with
θ_x	Decay value for world generation, with $x \in \{\text{Road, Logs, Empty}\}$
s	A single state
$ c $	Amount of vision grids used
$ s $	The amount of values representing state s
vg_x	The amount of cells the agent can look in direction x , excluding the cell the agent is currently in, with $x \in \{\text{left, right, up, down}\}$
\mathcal{A}	The action space
a	A single action
$T(c' c, a)$	The transition from cell c to state c' using action a
$\mathbf{Pr}(c, a, c')$	The transition probability from cell c to cell c' using action a
$r(c, a, c')$	The reward function from cell c to cell c' using action a
\hat{A}	The advantage value predicted
π	The policy
π^*	The optimal policy
α	The learning rate

ϵ	The clip range of the PPO agent
γ	The discount factor
n_steps	The amount of steps to do the rollout phase of training
$batch_size$	The batch size used whilst training
$vg = v$	Notation to state $vg_{down} = 0 \wedge vg_{up} = vg_{right} = vg_{left} = v$
max_age	The maximum amount of steps that can be made by the agent in the environment

B Hyperparameter values

d_{Bush}	$\sim U(0.1, 0.5)$
d_{Lilypad}	$\sim U(0.5, 0.8)$
cS_{Road}	$\sim U(50, 250)$
$conf_{\text{Road}}$	$\sim \{$ "0011111000000000" $,$ "000111000111000" $,$ "001100000011000" $,$ "011100110011000" $,$ "110000111000000" $,$ "001100000000000" $\}$
dir_{Road}	$\sim \{$ "left" $,$ "right" $\}$
cS_{Logs}	$\sim U(100, 500)$
$conf_{\text{Logs}}$	$\sim \{$ "100010001000111" $,$ "110011001100111" $,$ "100110001100100" $,$ "110000101000010" $,$ "111000111000111" $,$ "101110011001100" $\}$
dir_{Logs}	$\sim \{$ "left" $,$ "right" $\}$
i_{Rail}	$\sim U(20, 50)$
sp_{Rail}	$\sim N(3, 6]$
dir_{Rail}	$\sim \{$ "left" $,$ "right" $\}$

Table 5: Layer generation hyperparameters

θ_{Logs}	$= 0.5$
θ_{Road}	$= 0.7$
θ_{Empty}	$= 0.2$

Table 6: World generation hyperparameters

Learning rate α	$= 3 \cdot 10^{-4}$
Clip range ϵ	$= 0.2$
Discount factor γ	$= 0.99$
Number of steps n_steps	$= 2048$ to run for update
Minibatch size $batch_size$	$= 64$
Epochs when optimizing surrogate loss	$= 10$
Normalize advantage	$= \text{True}$
The maximum value for gradient clipping	$= 0.5$
Entropy coefficient for loss calculation	$= 0.0$
Value function coefficient for loss calculation	$= 0.5$
Hidden network architecture	$[64,64]$
Activation function	\tanh

Table 7: Default PPO hyperparameters

The reinforcement learning model used is a PPO model. The *policy* this model uses is a multilayer perceptron (MLP), with Actor-Critic implementation. Standardly, both the policy network as well as the value network have an input layer consisting of $|s|$ nodes, then 2 layers of each 64 nodes follow with activation function \tanh , and lastly the output layer consisting of $|\mathcal{A}|$ nodes for the policy network and 1 node for the value network. Table 7 shows the default parameters used for a PPO model. What differs from the PPO models we have used, however, is that we have edited the network architecture to 2 layers of 32 and 16 nodes respectively, for both the policy network and the value network. A visualization can be seen in figure 33. The activation function remains the same.

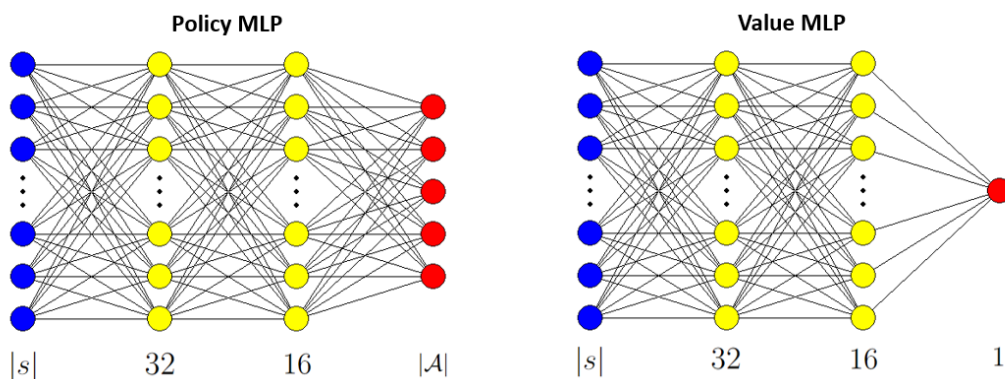


Figure 33: The different MLPs used in the PPO network. At the bottom of each layer, the amount of nodes are denoted. The blue coloured nodes form the input layer, the yellow nodes form the combination of hidden layers and the red coloured node(s) form the output layer. The individual perceptrons use a \tanh activation function.

C World generation algorithm

Algorithm 2: Section generation algorithm

```
1  base ← base + 0.01
2
3  SELECT SECTION TYPE UNIFORMLY FROM {Road, Logs, Train, Empty}
4
5  # create an ordered set.
6  layers = ∅
7
8  IF SECTION TYPE == Logs:
9      currentBase ← base
10     ADD LLogs to layers
11
12     # Generate non-static layers
13     WHILE  $r \sim U(0,1) \leq \text{currentBase}$ :
14         currentBase ← currentBase ×  $\theta_{\text{Logs}}$ 
15         ADD LLogs to layers
16
17     # Generate static layer
18     IF  $r \sim U(0,1) \leq 0.3$ :
19         ADD LBush to layers
20     ELSE:
21         ADD LLilypad to layers
22
23 ELSE IF SECTION TYPE == Road:
24     currentBase ← base
25     ADD LRoad to layers
26
27     # Generate non-static layers
28     WHILE  $r \sim U(0,1) \leq \text{currentBase}$ :
29         currentBase ← currentBase ×  $\theta_{\text{Road}}$ 
30         ADD LRoad to layers
31
32     # Generate static layer
33     ADD LBush to layers
34
35 ELSE IF SECTION TYPE == Rail:
36     ADD LRail to layers
37
38     IF  $r \sim U(0,1) \leq 0.5$ :
39         ADD LRail to layers
40
41     ADD LBush to layers
42
43 ELSE IF SECTION TYPE == Empty:
44     ADD LEmpty to layers
45
46     WHILE  $r \sim U(0,1) \leq \text{currentBase}$ :
47         currentBase ← currentBase ×  $\theta_{\text{Empty}}$ 
48         IF  $r \sim U(0,1) \leq 0.1$ :
49             ADD LEmpty to layers
50         ELSE:
51             ADD LBush to layers
52
53 # Check whether the generated section is possible to get across
54 IF section is possible to cross:
55     # Recall that  $\mathcal{L}$  is an ordered set!
56      $\mathcal{L} \leftarrow \mathcal{L} \cup \text{layers}$ 
57 ELSE:
58     Discard current generation and repeat this function
```

D Training plots

D.1 No step reward and single-value cell representation

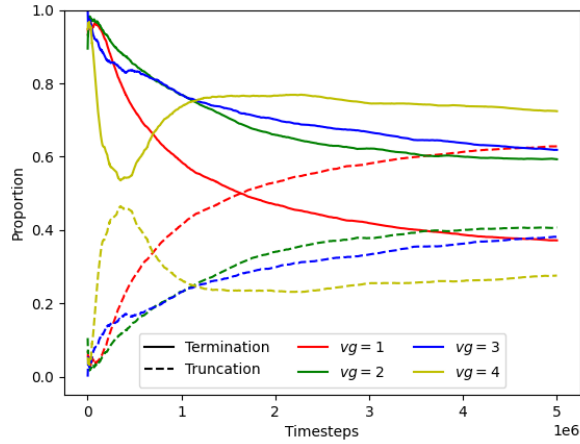


Figure 34: The type of death that ended the episode during training. Here it can be observed how the proportions of the type of death shift, increasing the proportion of environment truncations during training, especially the smaller the vision grid.

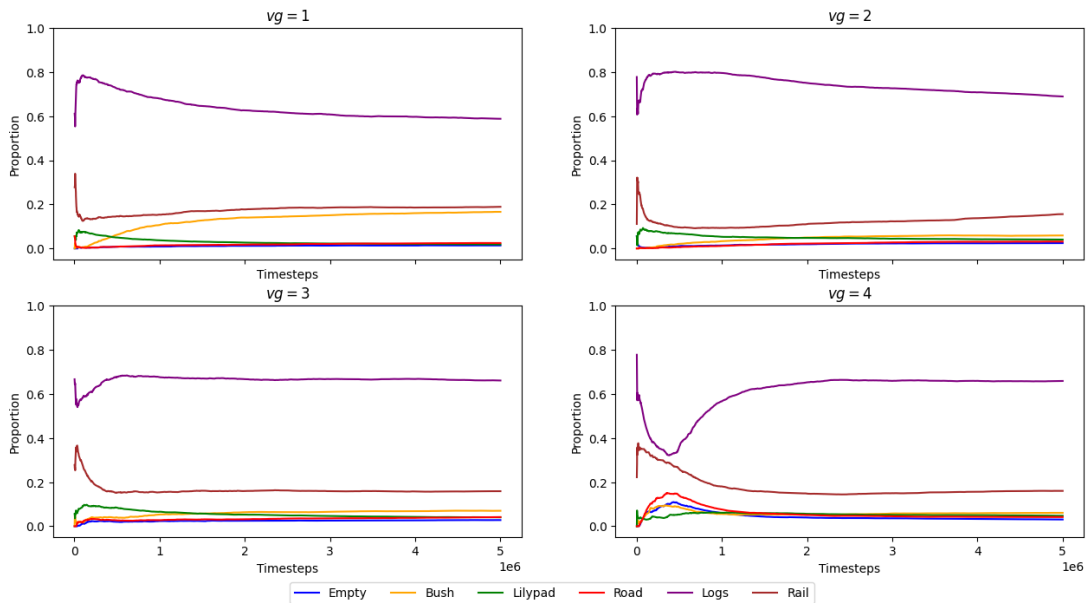


Figure 35: The proportion of types of layers the agent ended the episode on for various vision grid sizes during training. All vision grid sizes show similar proportions, with some exceptional behavior.

D.2 With step reward and single-value cell representation

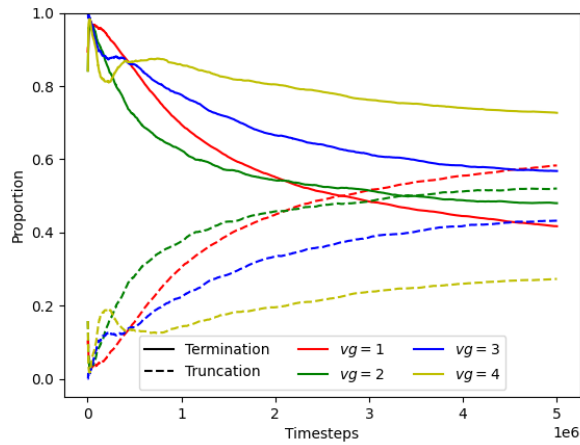


Figure 36: The type of death that ended the episode during training with step reward. Here it can be observed how the proportions of the type of death shift, increasing the proportion of environment truncations during training, with the two smallest vision grid sizes having the proportion of truncation exceeding the proportion of termination after training.

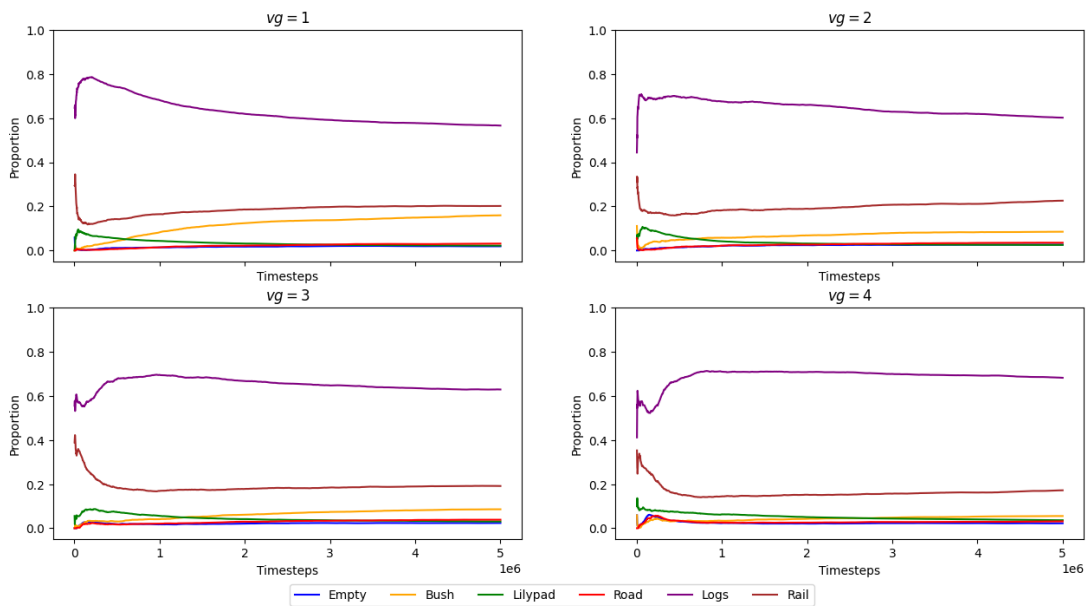


Figure 37: The proportion of types of layers the agent ended the episode on for various vision grid sizes during training. All vision grid sizes show similar proportions, and results are comparable with figure 35

D.3 With step reward and double-value cell representation

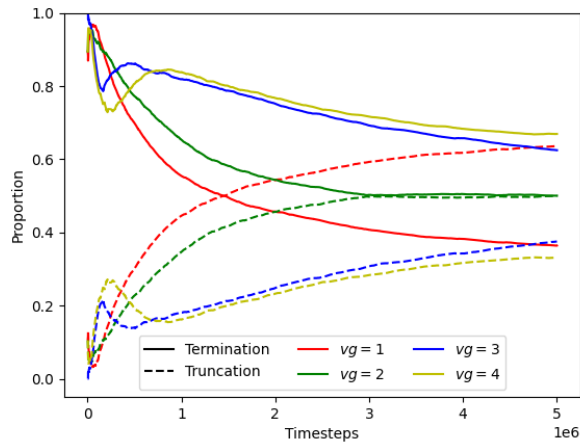


Figure 38: The type of death that ended the episode during training with step reward and double-value cell representations. The agents with vision grid sizes with vision grid size $vg = 2$ and $vg = 4$ seem to be lying closely together.

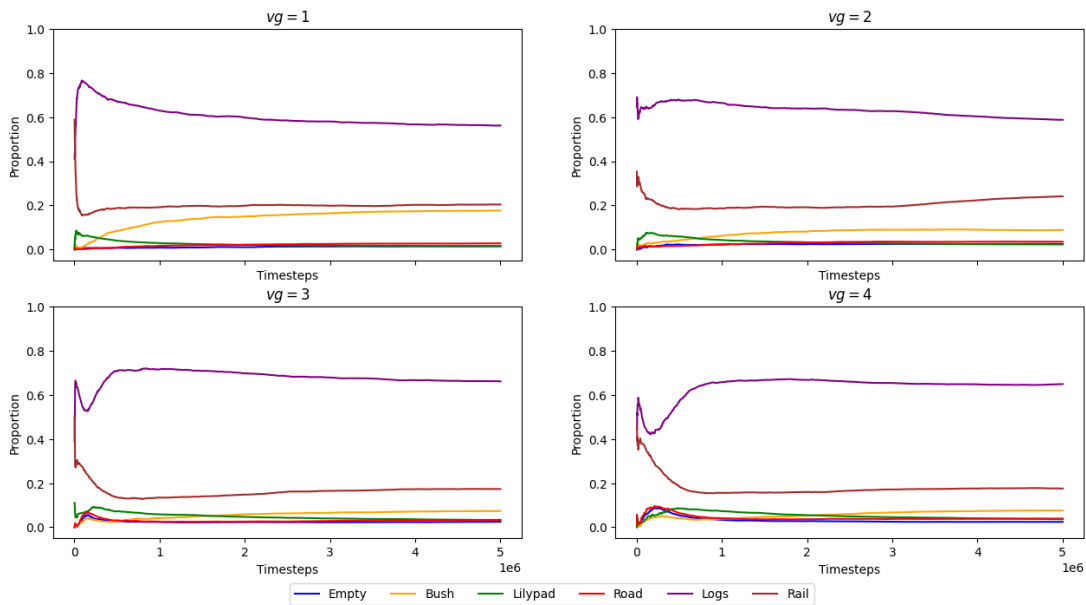


Figure 39: The proportion of types of layers the agent ended the episode on for various vision grid sizes during training. All vision grid sizes show similar proportions, and results are comparable with figures 37 and 35