Universiteit Leiden
The Netherlands

Opleiding Informatica

Performance of Reinforcement Learning Agents within Gameworlds of differing Coarsnesses

Anthony John Bot

Supervisor:
Dr. D.M. Pelt

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

17/07/2023

Abstract

Games are meant to be played by humans, but even so AI's have been created to play games too. Many games are built as a gridworld where elements are kept within designated tiles. This would be good for implementing Reinforcement Learning(RL) agents that use a state-space to save its exploration data and choose actions according to it. So a tabular gameworld can be easily converted into a state-space. However this becomes more difficult when the player character is separate from the gridworld, where it can exist anywhere between multiple states. With a continuous location for the player it is hard to compute with a RL agent, but it has been done before using approximation algorithms and deep learning. But what happens when the player is bound by this grid, when the player position is contained by tiles or subdivisions of tiles? In this thesis, the effects on RL agents are explored when the RL agents are bound by a grid that can be subdivided. With a Python program gameworlds can be created where a player can interact with the world elements and its physics. Using a variable called Coarseness the movement of the player can be changed through changing the amount of subdivisions of a tile. By implementing RL agents, Q-learning, SARSA and Expected SARSA, they can traverse the world and learn from the environment at multiple coarseness settings. After testing the results are accumulated into multiple graphs as well as the shortest path each agent was able to achieve. In small and simple worlds all agents perform comparatively with only execution time going up with each coarseness setting. As the complexity of the worlds rise so does the difficulty of the agents reaching the goal. Q-Learning and Expected SARSA often still perform similar with Expected SARSA usually performing a little bit better than Q-learning, both of them capable of learning more complex world design. SARSA performs comparatively badly, unable to finish some of the tests and sometimes unable to create efficient shortest paths where Q-Learning and Expected SARSA do both. All in all with a sufficiently well coded agent they can be able to traverse gameworlds with many subdivisions normally given enough time. Q-learning and Expected SARSA show this, but SARSA is only good at low complexity. The biggest impact is execution time, as the tiles get more subdivided the state-space grows with it. Using smaller and less complex worlds reduces this execution time, a high exploration perimeter helps finding the goal. Future research could be sub-rewards for reaching new areas, player inputs during subdivided tile movement and adjusting the state-space implementation.

Contents

1 Introduction

There are many games in the world, naturally they are meant to be played by humans, but with the availability of computers, modern advances and demand for Artificial Intelligence many games have been explored to implement some sort of computerized player. May it be by the game developers themselves to create an antagonist or companion for the player that acts separately, or made for playing the games instead of the player. External programs than can play games as aids to the player or by themselves in sections, from start to finish, supervised or even starting with no knowledge of the game.
Amongst those AI's are Reinforcement Learning(RL) Agents, that use reinforcement to learn of the game from nothing using a state-space. Through exploration of gameworld it fills the state-space with action-reward values to determine what the best action is in each state.
Games are often based on grid systems, tiles with set boundaries and characteristics that define what they are and do. This is really good for RL agents as these tiles can be concisely converted into a state-space. The issue for RL agents arises when the player character is not bound by this same grid system. This means that a player can interact with the tiles and spaces differently than what an RL agent may expect. A game such as `'Geometry Dash'`1, as well as games such as Super Mario Bros., Castlevania and Celeste use similar tabular worlds. Worlds that consists set tile sets than can include floors and platforms, hazards such as spikes and different world items than can be interacted with. With the aforementioned games the players as well as most enemies are loose

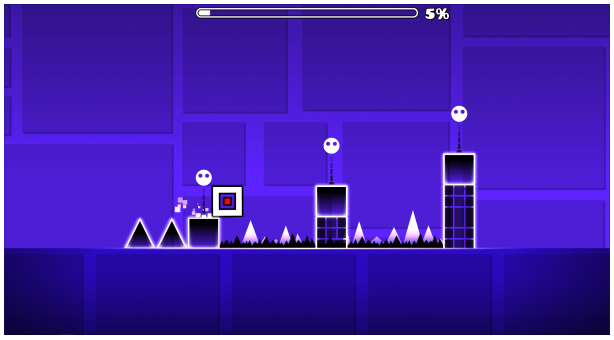

Figure 1: Screenshot of the first level in the Geometry Dash game. The playercharacter moves freely from the gridworld RobTop Games - Steampage.

from this tabular gridworld and are not predefined to be entirely within tiles. Instead they use hitboxes around the position of the entities location that can float freely through, on top of and below these tiles. The hitbox interacts with the game world, if the hitbox collides with a floor, the entity and hitbox will stay on top of the floor. If a hitbox collides with a wall, the entities horizontal movement stops. If a players hitbox collides with an enemy hitbox or with a hazard, the player will take damage. As such there are numerous examples of entities and, in particular, the player interacting with the surrounding environment.
Applying an RL agent to this type of game would mean making compromises somewhere. Using a state-space as continuous as the players position may be outright impossible. But methods do exist that can make use of this continuous position, such as using approximation policies and deep learning. Reducing the state-space will be beneficial to some point, but at what point would one see

diminishing returns? Where the state-space is coarse enough where the agent makes unnecessary mistakes due to the rounding towards one state or lacking a good state transition to go from one state to a further state.
But what happens when the player IS bound by this tabular world? For one, movement may seem very rough as a result, as the player could move one whole tile per timestep. However this is not necessary, the player could be programmed to move to sub-tiles, a division of the tiles, so that the movement of the player does not appear as rough and yet still be bound by the gridworld, but instead smoother with every further division.
As such one could wonder how the performance of RL agents are affected when such a translation is made between continuous player movement and a strictly set state-space. Or how the performance of the agents is effected when traversing such a gridworld where the players location is indeed locked to this grid, but with differing subdivisions of those tiles to indicate this players location.
Naturally the research question follows and will be addressed in this thesis: **Will Reinforcement agents be affected in performance when more states in relation to tiles are used within the agents decision process?**

1.1 Method

With the research question in mind, a program or game is created to find the results. This program stands free from previous made games, without the need to lock into their executables or screen outputs. Without the need to check for framerate pacing and alignment. Then we can research at our own speed with the settings we desire. But we do need to create a gameworld, where we can control the player and their inputs, as well as their outputs with the physics of the game.
This gameworld is modifiable to allow a multitude of different configurations of elements as well as be scalable through the coarseness variable, the variable that determines how many subdivisions of a tile the players position can partake. The player has access to the controls, which is going forward and backwards as well as be able to jump or instead simply moving in cardinal directions.

Contributions to the project are:

- A program of code that can create a gameworld environment where the coarseness can be adjusted, with capabilities for user-defined worlds, visualization of these worlds and taken paths.
- An implementation of three RL agents that can use the gameworld and play it.
- A set of experiments to determine what the effects of coarseness are on the learning behaviour of the agents and the effects on computational requirements.

1.2 Thesis overview

This chapter contains the introduction about gameworlds and Reinforcement Learning agents; Section 2 includes the definitions of used RL agents, their formulas as well as other broad terms; Section 3 discusses related work; Section 4 describes the creation of the game tools, how each module is set up and connected; Section 5 describes the experiments on the different gameworlds and each their outcome; Section 6 concludes and gives suggestions for further work.

2 Definitions

In the world of AI there is a partition is called Reinforcement Learning Agents. As the name suggest these agents learn through reinforcement, when an agent takes an action, decided either randomly or through some formula, in a certain state or location, it records its reward in a table so that it can compare and choose the actions to take when it re-arrives in this same state. This table is the State-Space or Q-table the agents use, where all possible states and actions are recorded with the reward each action gives in each state, usually stored in an array or index, which is a list of items. The agents used were the Q-Learning agent, State-Action-Reward-State-Action agent(SARSA) and the Expected SARSA agent.
All agents use the same selection criteria to select an action a from an actionset $\mathcal{A}$. This is called a greedy action where the agent uses the best action it can find in the current state. However there is a random chance that the agent will do a random action instead, not including the best action. In this thesis this chance for a random action is 10 percent, denoted by ϵ. This random action can help exploring actions and states that have not been recorded into the state-space before, but it could also lead to taking an objectively bad action, such as entering a state where the agent dies. Meaning that there is a $1 - \epsilon = 90$ percent chance of choosing the action a that is the best action b in the action set $\mathcal{A}$ of the state Q. The 10 percent remainder is divided over the rest of the available actions.
This is the formula used for choosing an action with the ϵ-greedy policy:

$$\pi_{\epsilon-greedy}(a) = f(Q, \epsilon) = \begin{cases} 1 - \epsilon, & \text{if } a = \arg\max_{b \in \mathcal{A}} Q(b) \\ \frac{\epsilon}{|\mathcal{A}|-1}, & \text{otherwise} \end{cases}$$

The agents also use what is called a policy update, an update to the their respective state tables with some different implementations for each agent. The Q-table state value $Q(s_t, a_t)$ is updated by modifying its own state value by adding a combination of a learning perimeter α, given Reward R_{t+1} and a static value γ (which is set to 1 in this thesis) that controls the influence of the policy, a value that changes per policy and lastly subtracting the current state value. With t being the current timestep and $t + 1$ being the next timestep.
`Q-Learning`[WD92]
With the part $[\max_a Q(s_{t+1}, a)]$ the Q-learner always updates its policy with the best move it could have taken in the next state. However this does not take into account that the next state might not be actually realized, which makes it off-policy.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big)$$

`SARSA`[Lor22]
SARSA is the on-policy variant of the Q-learner, now using $[Q(s_{t+1}, a_{t+1})]$ instead to update its policy. With this policy it always updates the Q-table with the value that was actually realized from the action that was taken.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big)$$

`Expected SARSA`[JGC+19]
The Expected SARSA is a modification of SARSA, using the expected value $[\Sigma_a \pi(a|s_{t+1})Q(s_{t+1}, a)]$. Instead of sampling the next action like SARSA does, it uses the sum of all possible actions at the next state to reduce the variance caused by random sampling by the selection policy. In this implementation Expected SARSA is an on-policy agent as it uses the same policy to calculate the expected value as is used to select actions.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(R_{t+1} + \gamma\Sigma_a \pi(a|s_{t+1})Q(s_{t+1}, a) - Q(s_t, a_t)\Big)$$

The agents in this example will traverse a gameworld, a 2D array of locations, each resembling either Air, through which the agent can freely move, Walls/Floors which the players cannot enter and Hazards, which the agent dies to. The gameworld also encompasses the Goal location and player location, the latter of which is influenced by coarseness. In this context coarseness will mean subdivisions of an integer or gameworld tile, with each subdivision the player can move half of the previous coarseness level in one step partition or quarter step. At coarseness 0 the player moves 1 tile each step, at 1 it is half a tile, 2 a quarter, 3 one-eighth, etc.
When an agent traverses a world it receives a reward through every step, dying gives a strong low reward to deter doing that action, reaching a goal ends the game. These rewards can be plotted over time creating a learning curve. This learning curve usually starts really low, where the agents are met with an unknown environment and occasionally bump into hazards before ever reaching the goal. Over time this curve increases where the agents learn to avoid hazards and are able to more efficiently reach the end goal for a higher reward. When the agent finds optimal strategies to follow to reach the goal, the curve flattens or converges.

3 Related Works

3.1 Markov Games

To start simple we have Littman's research[Lit94], in which he describes two learning agents, a minimax-Q agent and an implementation of the Q-Learning agent that was also used in this thesis. His research shows that these agents can give good performance in a smaller and clearly defined state-space. These simple experiments can show the ground work of reinforcement learning agents and show how simple problems can be solved by them.

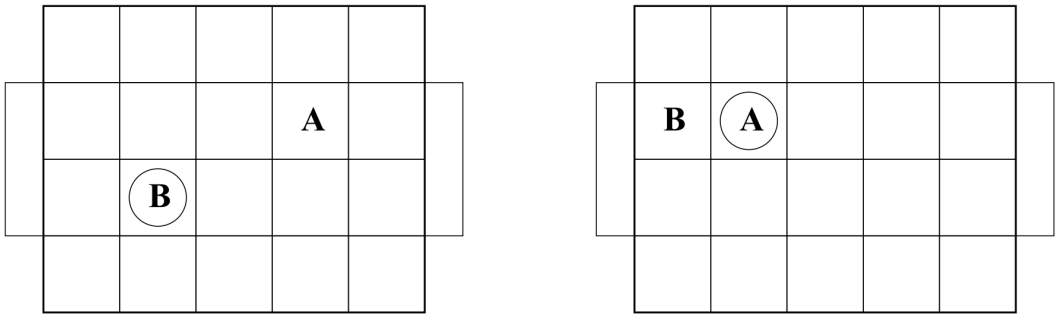

Figure 2: [Lit94]Representation of the soccer field, left is the initial board where with both players needing to score at the opposite goal. Ball possession is initialized at random and indicted by the circle. This is a small state-space with gridbound players.

3.2 Atari Reinforcement Learning

With the previous example the basics of RL agents can be explained, but using them can be quite limited. In this thesis the implementations for Q-Learning, SARSA and Expected SARSA are unmodified, but sometimes you need well calculated modifications to make agents function within a new research space. One such modification can be Deep Learning with image data, so that you can take in image data and convert that into usable data for RL agents. In 2016 this was done by Zhao et al.[ZWSZ16] to create 'Deep SARSA' and compare it against 'Deep Q-Learning' using the Atari 2600 for the input data.
In 2020 another group took the Atari 2600 one step further. Instead applying Model Based Reinforcement Learning to the agents to learn the games. Kaiser et al.[KBM+19] apply an algorithm called Simulated Policy Learning `(SimPLe)` to the Atari console instead. Showing how the performance of RL agents can differ compared to others, in this case comparing against Rainbow and PPO. Which can tell what strategies and policies can lead to greater performance than previous research and implementations.

3.3 Recurrent Neural Networks

As video games often are image data RL agents often are adapted to use this image data. With Recurrent Neural Networks[MJ01] you can apply another modification of an RL agent. Showing this off visually we have a Youtube video made in 2017 by SethBling where he shows how an RL agent sees the input screen and processes this into useful data using Recurrent Neural Networks generating outputs for the agent called Mariflow[Set17]. Here we can see how continuous data can be constructed into a much simpler concept, even when the gameworld is not clearly tabular to the input data, this concept of data can be adapted to work with RL agents. Converting an image into a simple data stream that, in this case, the neural network can interpret alongside the additional info about the environment it receives.

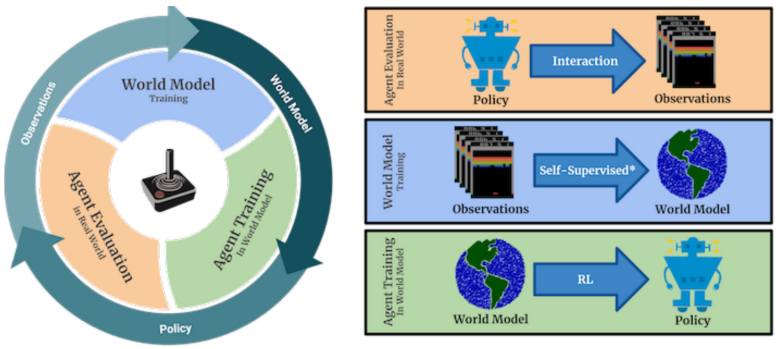

Figure 3: [KBM+19]Showcase of the SimPLe loop, where it 1). Interacts with the world, 2) updates the algorithms Model of the world and 3) the agent acts with the new parameters of the updated model

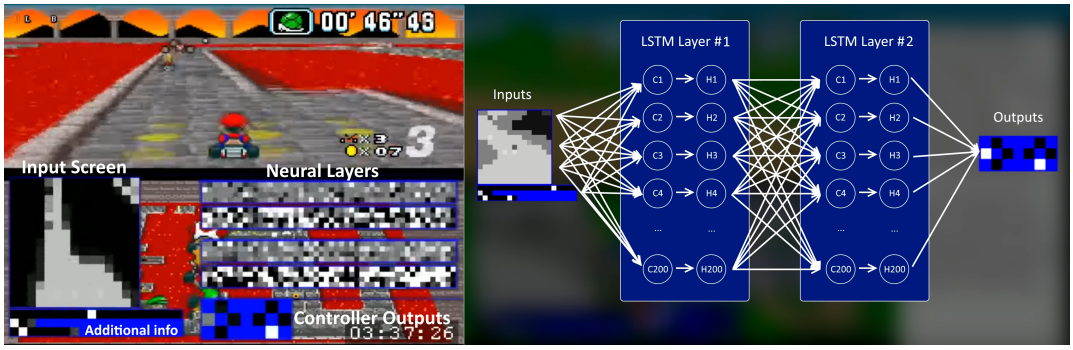

Figure 4: MariFlow showcase. In this figure we can see the screen of the Mario kart game on the left, with the neural network inputs, layers and outputs overlayed on top. On the right side we see a further in depth visualization of what the neural network entails. Source: www.youtube.com/Sethbling.

3.4 Starcraft II

Games such as Starcraft II serve as different example. Instead of a small state-space and limited world elements we have a gameworld with a giant state-space as well as a multitude of players, which creates more difficult learning criteria and conversion for RL agents.
Yet in 2017 Vinyals et al.[VEB+17] tried implementing an RL agent for the Starcraft game. By having all the game elements converted from 3D into 2D and making them grid aligned, the game data will be more machine readable for an RL agent. Converting all game data into several 'feature' layers of a state-space. This makes RL agents capable of learning how to play this game.

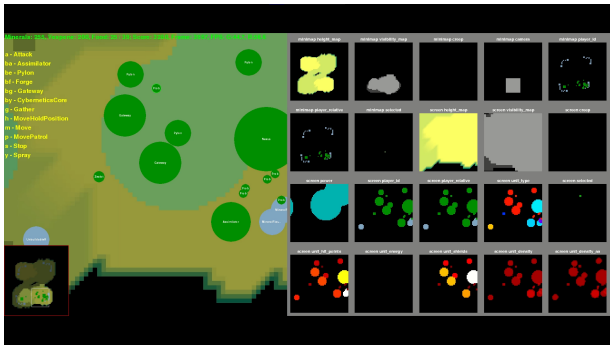

Figure 5: Left is a human interpretable game world representation, right are a multitude of feature layers for an agent. Source: DeepMind open source PySC2 toolset for Starcraft II.

4 Program Building

In order to answer our research question there are a few requirements that need to be met before testing and its results can be achieved. We need a gameworld with all its elements, a player that experiences all physics that correspond to the gameworld, Reinforcement Learning Agents that can control the player and learn to move towards the goal and lastly a method to gather the data from the agents.

4.1 Generating worlds

To start with I first needed to create game worlds that an agent has to traverse in order to reach an end goal. The world size, a player position, walls, floors, air, hazards and the end goal had to be established to create a functional gameworld.
To make this task easier a way was made to create worlds within the command line interface of the program that would automatically fill a world with air and lets the user place all the other elements within the game world. The world is then saved within a .txt file which can be further modified in a text editor.
When the program environment is created it tries to read a default world.txt, when it is found it does some basic checks to see if the file is properly formatted. Properly addressing a value to each variable in the process. When the file is not found the user is asked to type a different file location or create a new game world.

4.2 World typing

To output the visual of the gameworld and its elements each tile must be printed to the console with a character, due to the command line interface we need to print the lines in reverse order, going from array element to array element to output its data of being Air, Wall, Hazard, Goal or Player.
There are two types of worlds that are most important to this project:

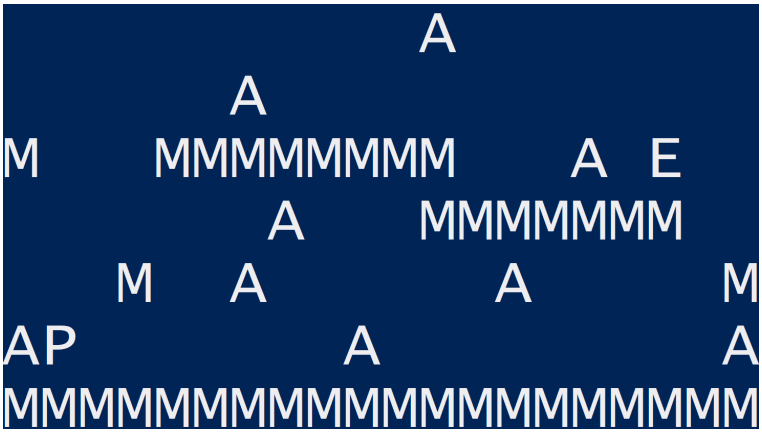

Figure 6: Screenshot of the Command Line Interface. "P" representing the player character, "M" is land/walls, "A" are hazards, and "E" representing the end goal position.

4.2.1 Top-down

In the book `Reinforcement Learning: An Introduction`[SB18], written by Richard S. Sutton and Andrew G. Barto, we learn about gridworlds. Amongst the gridworld types there is one where the player is set within a 2 dimensional world which is viewed from the top down. The player is able to go towards the 4 cardinal direction to reach its goal. This mode was created to see if the agents performance still closely resembles the implementation of the Reinforcement Learning Agents.

4.2.2 Vertical Slice

One popular 2 dimensional world would be vertical slice, used within primarily older titles or indie games, such as Super Mario, Castlevania and Celeste. Here the player is set unto a world with ground to stand on. With the ability to move forwards, backwards and jump. Gravity and speed are important within this world type as the player can reach different velocities within the same location. This creates a bigger state-space where each velocity type adds a dimension of complexity and every value for velocity enlarges this dimension.

4.3 Physics

The player must experience physics in the gameworld, though limited to only moving towards cardinal directions in the top-down world, in the vertical slice variant there are a multitude of additions to the gameplay. One such thing is gravity, if the player is airbound, i.e. has no floor below the player, they will receive an increase in downwards momentum every timestep until the player is grounded again. Only when the player is grounded on the floor are they able to jump upwards.
Another added aspect is the velocity, the player is capable of moving multiple tiles in one timestep, limited to 3 tiles per timestep in each direction. With this velocity aspect a small subset of moves was added to the vertical slice gameworld as well. No longer does it feature all four cardinal directions, instead it is jumping, moving left and right, and the combination of the two. But an

action that moves straight downwards is excluded due to the implementation of gravity.

4.4 Implementation of Coarseness

As coarseness would subdivide each tile 2^c times(where c = the set coarseness), this means each tile is $2^c \times 2^c$ as large as before. This comes with numerous issues to the physics model, player movement and collision detection to apprehend.
Since the top-down gameworld would be unaffected by the coarseness changes only to the vertical slice has an addition been made to the stepping process. Each step now include so called quarter steps that take the player one coordinate further in accordance to the set coarseness. When the coarseness is set to 0, there is only one quarter step made per point of velocity. At a coarseness of 1 this is now two quarter steps, at 2 and 3 it will be 4 and 8 respectively.

Coarseness	nr. Quarter Steps	Total Tile Size
0	1	1
1	2	4
2	4	16
3	8	64
4	16	256
5	32	1024

Table 1: Coarseness against the amount of quarter steps made to move a single tile

As players should not be able to phase through walls and hazards each quarter step now has 4 collision checks at each corner of the player character to see whether the player has entered the hitbox of a wall, floor, hazard, goal or out-of-bounds, such that player will be corrected to the right position or outcome. But as in this case only the players position has been modified by the coarseness, a translation to the players actual position to any gameworld elements had to be made to function correctly without scaling the entire world array up by the same $2^c \times 2^c$ value.
To show the gameworld command line output every element indeed had to be printed $2^c \times 2^c$ times to show the players actual position. Even though this were a simple addition, the command line would be quick to become unreadable at higher coarseness values without zooming out unnecessarily.

4.5 Implementation of RL Agents

With the gameworld asking for player input, a person can manually play the program by themselves. However, for testing how Reinforcement Learning Agents traverse these worlds at different coarseness levels I need to implement the agents themselves.
To initialize the agents a few parameters need to be set in accordance to the gameworld, such as the size of the state-space and the number of actions the agents have, which are 4 for the top-down type and 6 for the vertical slice type. To select an action I implement the ϵ-greedy algorithm that picks the best action within the current state with a 10% chance of picking a different random action. Every agent also needs its own update policy, coded in the same way as the formulas in Section 2 are built up. This means that every agent should have different performance within the gameworlds, for a good comparison on what can be a good or bad RL agent for these type of tests.

To test with the agents they need to be set up in a way that they can play these gameworlds automatically, with multiple exploration settings, from start to finish. Such that the program cycles through each agent, exploration perimeter, as well as cycling through every episode without resetting the state-space and repetition to average the results. All the test data must be interpreted and transferred into graphs to clearly show the performance of each agent at each settings.

4.6 Pygame

To combat the illegibility of the command line interface a different source of visualization should be used. Therefor, a Pygame function was implemented to show a screen of the gameworld instead of the previously used text based command line output. Using a set of coordinates the agents path towards the goal can be plotted. With a dynamically calculated resolution, the gameworld would be readable at any setting. The maximum resolution is calculated by multiplying the gameworlds width and height with the coarseness resulting in a pixel multiplier or divisor. By comparing the world size with the maximum screen size (such as 1920 by 1080 pixels) the pixel multiplier can be adjusted to fit the screen size properly. In the case a large set coarseness the base size of the world may exceed the screen size, in this case the pixels are divided to be able to fit, with close attention that a pixel drawn on screen cannot be less than a pixel in size.

4.6.1 Importing and exporting paths

If a path must be showcased without the use of the programs that manually or automatically play the gameworld, these programs can export an array of path elements into a JSON file. By running the visualization module on its own it will ask for a JSON file to import the path from it. With the import of the gameworld it can display the agents path frame by frame without playing the game.

4.7 Gradient

To show the path of a player in a single image the idea was made to display it in a gradient. The visualization module was therefor copied and modified to show a single run of an agent. This program imports the players path in the same way as the visualization module using JSON files. Now it only displays a single pathway without resetting the background and air tiles. Such that each frame of the player character is overlayed on top of one another. By slowly increasing the amount of red and green of the blue player character the player is drawn from blue at the start to white at the end so that and earlier path can be distinguished from a later path. The finished result is then saved as a PNG file upon which the program closes.

5 Experiments

To test the final performance of the three implemented agents a diverse testing suite must be set up to see the effects of coarseness in a multitude of circumstances. Each circumstance much be easily differentiated from each other to see whether agents' performance becomes dissimilar in these situations.

5.1 Game Worlds

Simple World

This world shows how quick an agent can adapt to finding the goal in a small amount of steps, to converge into the most optimal route, even at larger coarseness's.

Figure 7: Pygame representation of the Simple Gameworld, with Blue as the players starting position, Green as the end goal. White as air in which the player can move freely, Black as walls and floor into which the player cannot move, Red as hazards to which the player dies and gets moved back to the start.

Short World

This world the agents deal with a higher step of complexity while still being able to traverse a similarly small world as the simple world. This should lead to the agents receiving a higher path length as well as more jumps and turns.

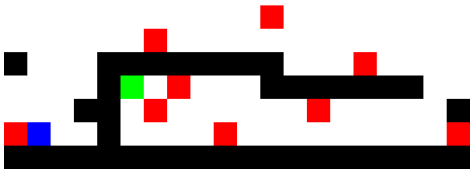

Figure 8: Pygame representation of the Short Gameworld, for color coding see figure 7

Long World

This world is to show how the agents will perform at large speeds, as the agents can move at 3 tiles per step at a maximum. No turning around, only jumps and two paths, an upper and a lower path.

Figure 9: Pygame representation of the Long Gameworld, for color coding see figure 7

Tall World

This world tests the jumping process of the agents and whether they can learn mastering the great amount of simple and complex jumps. Additionally the agents must traverse the same height downwards, with large amounts of hazards and together with gravity finding a way through the tight passageways.

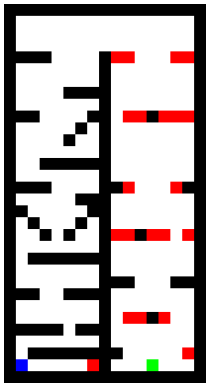

Figure 10: Pygame representation of the Tall Gameworld, for color coding see figure 7

Endurance World

This world tests the endurance of the agents to see how they learn a big world where finding the goal at all is a difficult task. By combining features of the previous worlds, this world features lots of vertical jumping, horizontal traversal, falling, multiple turns and at the end a split path.

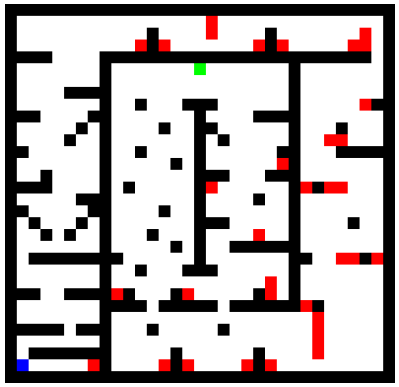

Figure 11: Pygame representation of the Endurance Gameworld, for color coding see figure 7

5.2 Results

5.2.1 Settings

Each world were tested with the three aforementioned agent settings. Each agent ran with 4 different exploration perimeters namely: $\alpha = 0.01, 0.1, 0.5, 0.9$. Each agent ran for 250 episodes with a maximum of 20000 timesteps each. With 8 repeats to average the tests. In the figures the path consisting of quarter steps of the agent goes in a gradient from blue to white, with teal representing a full step, though some full steps are overshadowed by quarter steps. Full blue are the first steps, almost white are the last steps. The path length is determined by counting every quarter step as well as every full step.
For reference to the execution time. The testing programs were ran on a Ryzen 5 3600, 16GB of 3200MHz DDR4 and a 7200RPM harddrive.

5.2.2 Simple World

Figure 12: The path of the Q-Agent, Expected SARSA agent and SARSA agent with no coarseness, coarseness 1 and coarseness 2 respectively in the Simple world. Coarseness 0 has no subdivisions per tile, coarseness 1 has 4 subdivisions and coarseness 2 has 16 subdivisions. Black being walls, Red are hazards, green is the goal in the bottom right corner. The players path is a gradient from dark blue to white. Teal is the player in a position where they take an action. In all three paths there is a single jump towards the right over the obstacles before reaching the goal in the bottom right, with only coarseness 2 jumping up before moving.

Discussion

In the simple world all performance seem similar with only minor differences. The best movement of each agent seems rather similar in each coarseness setting, consisting of a single jump and relying on the height to carry its speed over the hazards and to the goal as can be seen in figures 12 and 13. Therefor the agents have the same path length to each other in each coarseness as well as roughly the same performance curve as can be seen in figure 14. With the shortness and lack of hazards in this test, all agents in all coarseness levels quickly converge into an optimal movement set as seen at the high reward level, leveling off at around 100-150 episodes like in figure 14. With each coarseness level we can see how the path of the agents looks to be more smooth, yet more jagged by moving up, then sideways and only then down while looking more like a natural jumping curve at the low coarseness levels.
The biggest differences in the Simple world are in path length and execution time as seen in figure 15. With the coarseness increasing both path length and execution time increase seemingly exponentially, most likely due to the size of a single tile increasing by $2^c \times 2^c$. This world was quick to test, where execution time goes from just $3\frac{1}{2}$ minutes to 44 minutes.

Figure 13: Continuing from figure 12, these are the paths of the Q-Agent and Expected SARSA agent, coarseness 3 and coarseness 4 respectively in the Simple world. Coarseness 3 has 64 and coarseness 4 has 256 subdivisions. The same action happens as in coarseness 2 in figure 12, where the agents jump almost straight up, then straight right before slowing down and moving down towards the goal. See figure 12 for color coding.

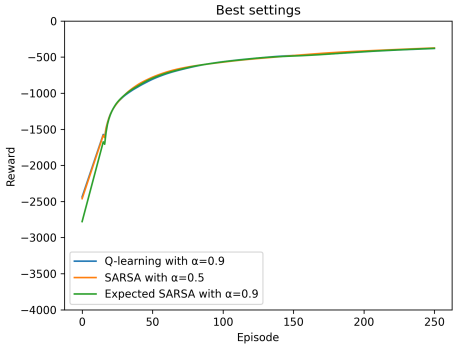

Figure 14: One of the performance curves of the Simple world agents, this one being of coarseness 4. All of the curves of this world look similar, only with the lowest rewards going down due to the coarseness rising. Lower rewards means bad performance as the agents explore the world and run into hazards before reaching the goal. As the agents explore the world they learn how to reach the goal faster and more efficiently each time, thus the rewards goes up the more episodes there are.

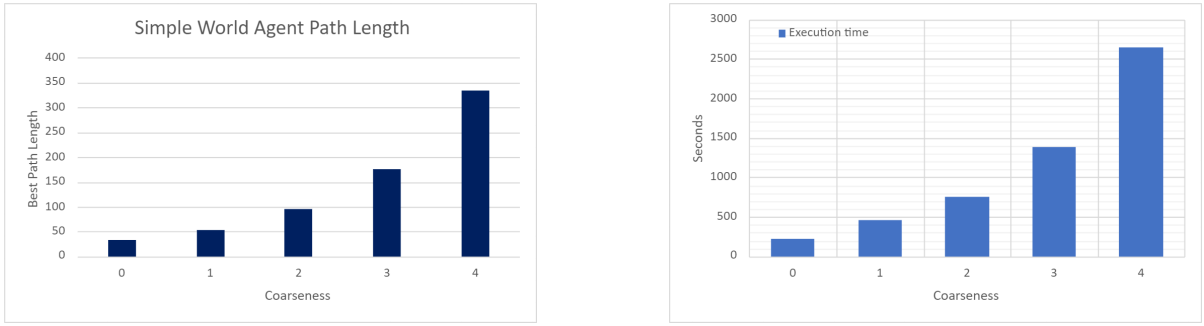

Figure 15: On the left is the path length of the agents throughout the coarsenesses of the Simple world. On the right are the execution times of each program of each coarseness. As the coarseness rises so does the path length and execution time seemingly exponentially. This can be explained as the tile size gets multiplied by $2^c \times 2^c$

5.2.3 Short World

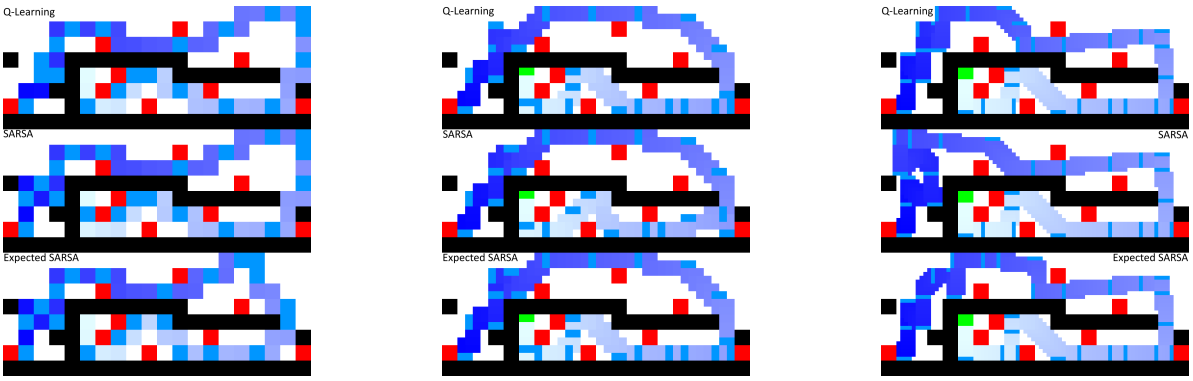

Figure 16: These are the best paths of each of the three agents for coarsenesses 0, 1 and 2 of the short world. Coarseness 0 has no subdivisions per tile, coarseness 1 has 4 subdivisions and coarseness 2 has 16 subdivisions. Black being walls, Red are hazards, green is the goal down and left from the center. The players path is a gradient from dark blue to white. Teal is the player in a position where they take an action. Overall the agents have optimal paths with only the SARSA agent in coarseness 2 having an extra jump to the left at the start. Some paths opt for moving under the top middle hazard while others jump over it with preference depending on coarseness.

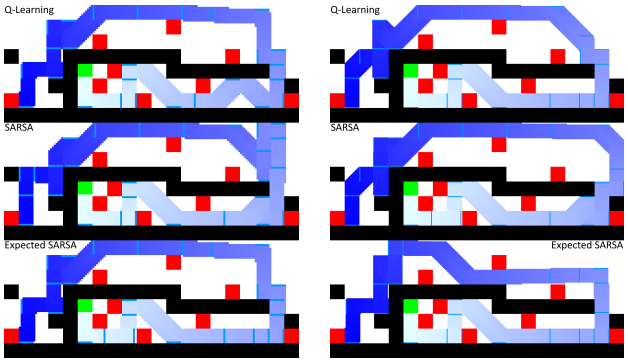

Figure 17: Continuing from figure 16, these are the paths of the three agents of coarsenesses 3 and 4 of the short world. Coarseness 3 has 64 and coarseness 4 has 256 subdivisions. Much like the previous coarsenesses, the paths are mostly optimal except for a few extra jumps mainly for SARSA at both coarsenesses and Q-Learning for coarseness 3, although with no time loss. The agents mostly show preference at jumping over the top middle hazard aside from Expected SARSA at coarseness 4. For color coding see figure 16

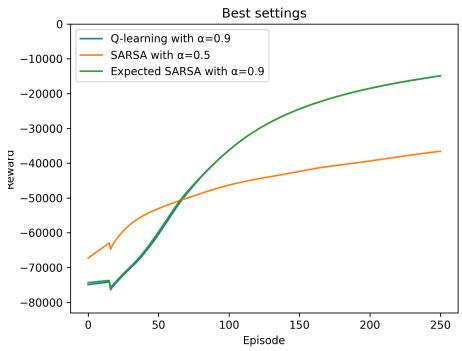

Figure 18: One of the performance curves of the Short world agents, namely of coarseness 4. As the complexity rises with the coarseness the performance curves start at a lower reward. As the coarseness rises it seems that the agents also take more episodes to converge into a flat line. Compared to the Simple world in figure 14, the coarseness 4 short test still has not converged after episode 250. In all coarsenesses but coarseness 0 the SARSA agent shows these lower performance metrics.

Discussion

The short world test is rather similar to the simple world test, where the agents are truly capable of finding a path that works rather optimally. However due to the rise in complexity and length towards the goal the agents seem to have issues with converging as can be seen in figure 18, where it takes more than the alotted 250 episodes to do so. The SARSA agent in coarsenesses 1 through 4 have unexpectedly worse performance compared to the other agents being about half as high. Yet the SARSA agent finds a short path just fine, but cannot seem to exploit this path.
In figure 19 we can see a graph of the path lengths of the agents, this graph tells that the Expected SARSA agent finds the best shortest paths while just SARSA has trouble finding a short path length with Q-Learning performing somewhere in the between the two other agents. In the same figure 19 we see the execution times. Interestingly the coarseness 0 test took longer than coarseness 1, as it seems that coarseness 0 seemingly has a very low reward (far from reward=0) towards the end compared to the other tests of this world.
We also see an incredible hike in execution time in figure 19 compared to the Simple world in figure 15, going from an already staggering 7.7 hours to an incredible 2.8 days. This can be attributed to the increased complexity and path length to the goal versus the simple world.

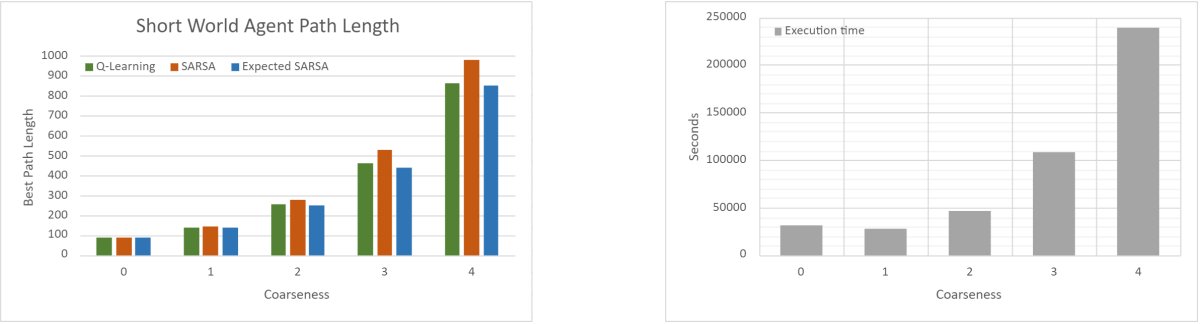

Figure 19: On the left are the best path lengths of each of the three agents at each tested coarseness level of the short world. We can tell that Expected-SARSA creates the shortest paths while SARSA is creates the longest. On the right are the execution times for each coarseness test of the short world. Interestingly coarseness 0 has a higher execution time than coarseness 1. Both graphs follow a upwards curve corresponding to coarseness seemingly due to the tile size increasing by $2^c \times 2^c$

5.2.4 Long World

Discussion

Performance of the SARSA agent proves to be even worse that the previous worlds as can be seen in figure 23 and a lack of a path in figure 24. In the performance curves SARSA converges more quickly than the other agents, often to the detriment of finding the end goal. Q-Learning and Expected SARSA both are very capable of finding a good path towards the goal. We see this reflected the path length graph in figure 26, where both Q-Learning and Expected SARSA have similar performance while SARSA's path length is almost double of the other two in coarseness 3. Sometimes the agents have a dimple in the performance curve. This can be explained by the two diverging paths where an agent tries one path, receives too much negative reward before continuing

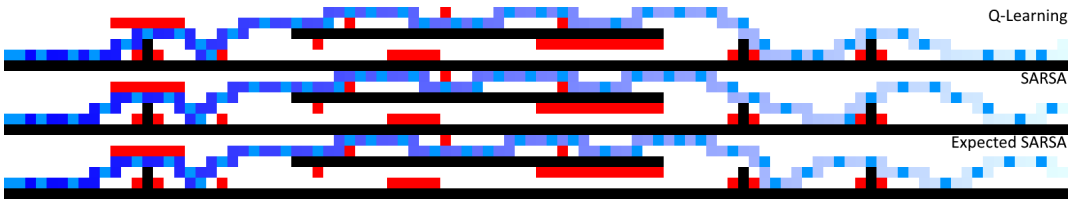

Figure 20: The best paths of the three agents for coarseness 0 of the long world with no subdivisions of the tiles. Black being walls, Red are hazards, green is the goal in the bottom right corner, 1 tile off the ground. The players path is a gradient from dark blue to white. Teal is the player in a position where they take an action. All three agents take the top path and move to the goal fluently

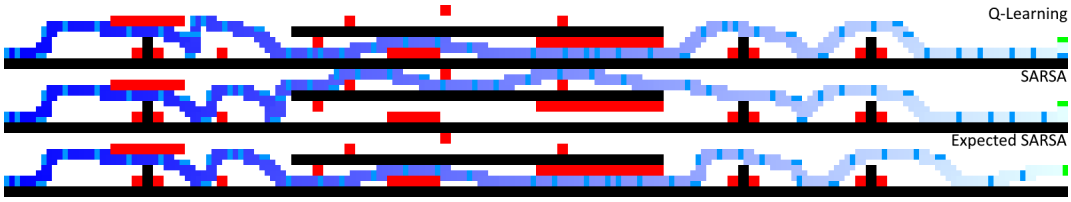

Figure 21: Paths of the three agents with coarseness 1 of the long world where the tiles are subdivided into 4. We see an interesting occurrence, both the Q-Learning and Expected SARSA agents take the bottom path to reach the goal. For color coding see figure 20

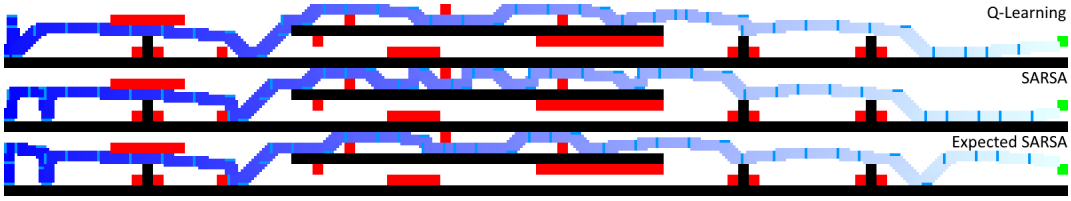

Figure 22: Paths of the three agents with coarseness 2 of the long world where the tiles are subdivided into 16. In contrast to coarseness 1 in figure 21 no agents take the bottom paths, instead doing some extra jumps in the beginning. For color coding see figure 20

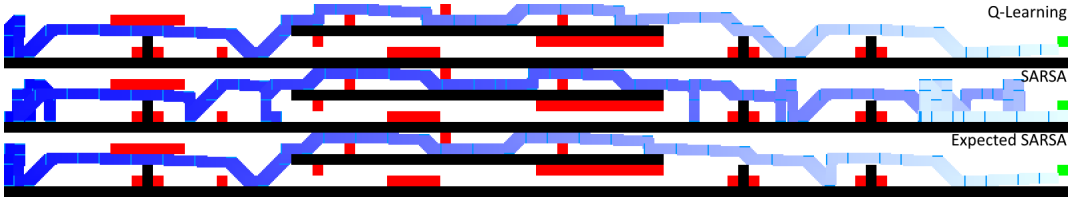

Figure 23: Paths of the three agents with coarseness 3 of the long world where the tiles are subdivided into 64. The same behaviour as the previous coarseness level in figure 22 when it comes to the early jumping. Now there is a difference where SARSA is being incapable of creating a straight line towards the goal, doing several extra jumps and even moving backwards before finally reaching the goal. For color coding see figure 20

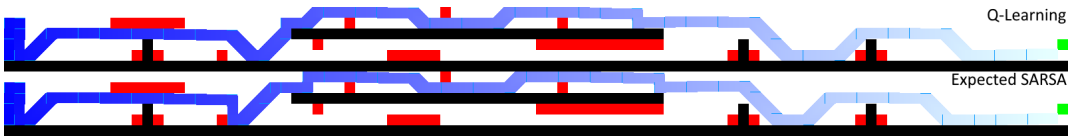

Figure 24: Path of the Q-Agent and Expected SARSA agent with coarseness 4 in the long world where each tile is subdivided into 256. Both the Q-Agent and Expected SARSA's paths look very similar and optimal. SARSA was unable to finish this coarseness, unable to reach the goal, somewhat expectable considering the performance of SARSA in the previous coarseness level in figure 23. For color coding see figure 20

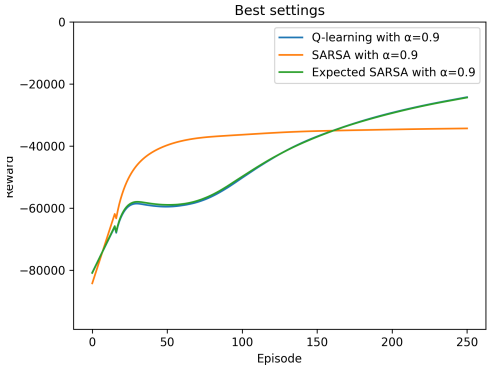

Figure 25: One of the performance curves of the Long world agents, this one being of coarseness 3. Most curves either look like a normal performance curve or with a small dip in the middle. This can be the agents trying the bottom path, receiving too much negative reward and then going to the top path. At higher coarseness levels both Q-Learning and Expected SARSA have trouble converging to a high reward in the 250 episodes of time they have. In this figure SARSA converges normally, avoiding hazards but often also being unable to find the goal effectively as seen in figure 23.

onto the other path. We see in a multitude of paths(22, 23, 24) sometimes multiple jumps in the beginning. Perhaps this is so that the agents can find a good setup to do the first real jump with hazards around.
In this world we can see how the physics can allow some moves in some coarsenesses, but disallow it in others. In figure 21 we can see that deviation from the norm: both Q-Learning and Expected SARSA take the bottom path, just barely making the jump over the hazards, while none of the other examples use this bottom path.
In figure 26 we see the execution times, the curve between the coarseness levels is nothing new, but with another leap in complexity there is another increase in execution time as well, this time ranging from 4.8 hours to 4.2 days. Ignoring the SARSA path length in the path length graph we can once again see the same exponential curve where the tile size increases by $2^c \times 2^c$

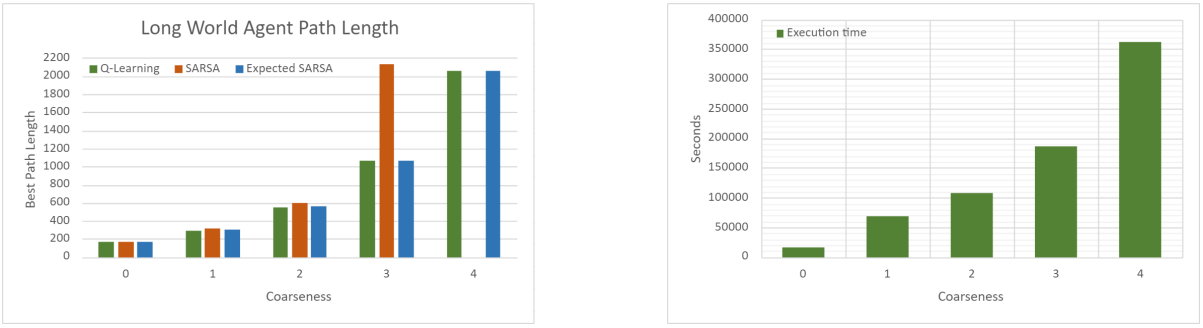

Figure 26: On the left are the path lengths of the Long world at each coarseness level. We can see that SARSA at coarseness 3 has worse performance compared to the other two agents and previous coarseness levels, to the point where in coarseness 4 SARSA was unable to finish the test. On the right we see the execution times of the Long world, where there are no anomalies. Ignoring the SARSA paths the graphs follow the same trend of growing exponentially with the tile size growing by $2^c \times 2^c$

5.2.5 Tall World

Discussion

Even though the Tall world is completable in every coarseness, it is proving very difficult for the agents to finish due to the sparse moves that can complete some jumps or falls. Thus the agents were not able to finish coarseness 4 and surprisingly coarseness 0 neither. SARSA was only able to finish coarseness 1 as seen in the left image in figure 27.
In this world as can be seen in figure 28 a high exploration value is very important to find the end goal of the world. With this high exploration perimeter more states are more quickly explored which leads to finding a way into the next section. In all graphs but coarseness 0, there is a large dip in the reward curve that both Q-Learning and Expected SARSA experience. This can be explained by the amount of hazards in the second half of the world. By exploring this section the agents experience a lot of deaths, which in turn gives a lot of negative reward until they find the end goal, where the reward curve starts to rise again.
Using the graphs in figure 29 we can draw a better picture. Due to the precise nature of this world the execution time start off rather high steadily increasing with coarseness, from 14.3 hours

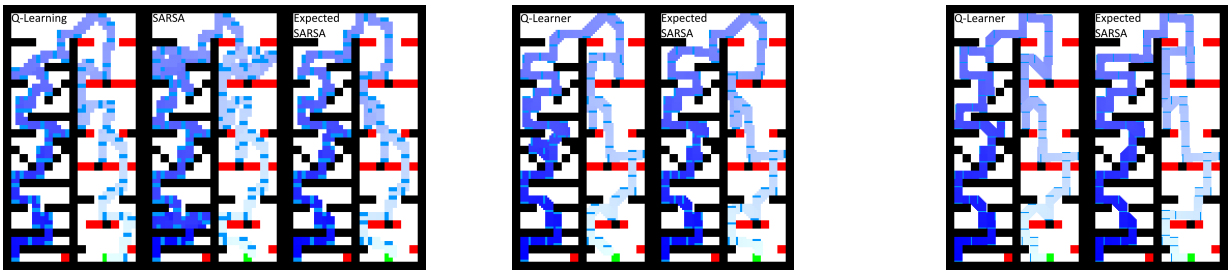

Figure 27: Coarseness 0 and 4 of the Tall world none of the agents made it to the goal. Only with coarseness 1 one the left were all agents able to get to the goal, with SARSA having much trouble at the start. With coarseness 2 in the center and 3 on the right only Q-Learning and Expected SARSA found a path to the goal, both being quite optimal paths. Coarseness 1 has 4 subdivisions per tile, 2 has 16 and 3 has 64. Colors are Black being walls, Red are hazards, green is the goal in the bottom center of the right column. The players path is a gradient from dark blue to white. Teal is the player in a position where they take an action.

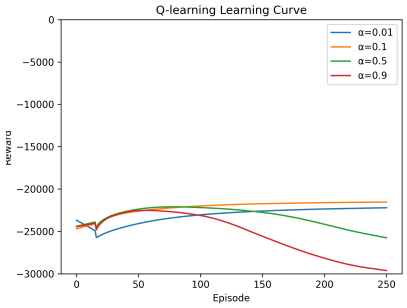

Figure 28: This is a performance curve for the Q-Learning agent of the tall world. In this world we have an upwards and a downwards section as can be seen in figure 10. This is reflected in the performance curves as well, where the reward stays mostly constant and no hazards are met until the agents reach the top of the world and need to move downwards where there are a lot of hazards. Both explorations values $\alpha = 0.5$ and $\alpha = 0.9$ follow this downward trend before rising again by exploiting a found path towards the goal. Neither $\alpha = 0.01$ and $\alpha = 0.1$ have been capable of reaching the downwards section in this example.

to 3 days. The latter of which still was not enough to let the agents finish the last coarseness level, it merely did not have enough steps and episodes to finally find a path to the end goal. The Expected SARSA and Q-Learning agents show the same downwards trend in figure 28 that should consequently lead to an upwards trend again thus telling that the agent have found the goal. Perhaps with enough time even SARSA would follow that same trend.

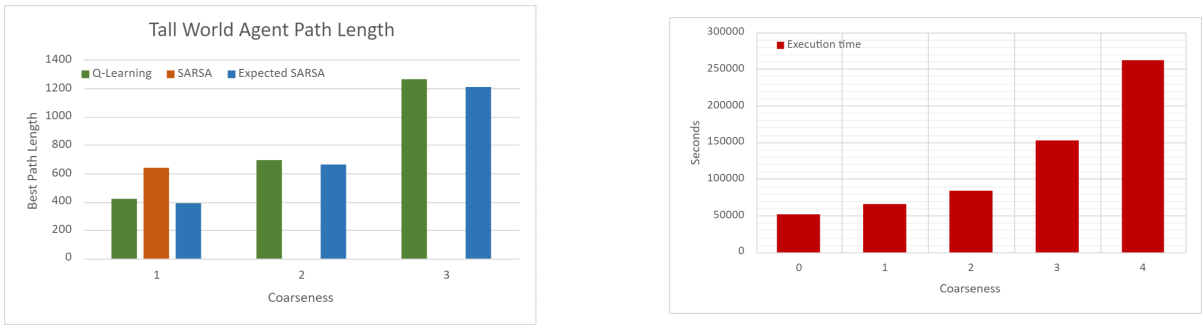

Figure 29: On the left is the path length of the three agents of the tall world with coarsenesses 1, 2 and 3. No agent was able to complete the level with coarseness 0 and 4 and SARSA was only able to complete coarseness 1. Expected SARSA looks to be a bit more efficient on finding the best path compared to Q-Learning. On the right we see the execution times, once again following the same exponential pattern where the tile size increases by $2^c \times 2^c$

5.2.6 Endurance World

Testing of the endurance world was aborted due to the high execution times of the program. Not only can higher coarsenesses take multiple days, with the enormous state-space the programs can already use more than 10GBs of RAM at its peak. Running multiple programs on the same system in a short amount of time would be impossible.
Additionally, looking at the tall world we can tell that the agents have trouble with vertical sections. With the addition of extra length the agents would have trouble reaching the end at all with the current settings and features.

5.3 Enveloping Discussion

With complexity of the levels comes a longer path for the agents to find with a longer execution time which can easily be seen in figure 30. If these paths are too precise the agents may have difficulty reaching the end goal as not enough states are explored in the allotted timeframe.
We can see that the physics slightly change depending on coarseness, where in coaseness 1 in the Long world in figure 21 it allows for a clearly faster path than is normally possible.

6 Conclusions and Further Research

In conclusion we now know for certain that increasing the coarseness for observing a world and increasing the state-space is detrimental to especially execution time. Agents will have a harder time exploring all the enormous state-space and reaching the goal. This however does not keep the agent

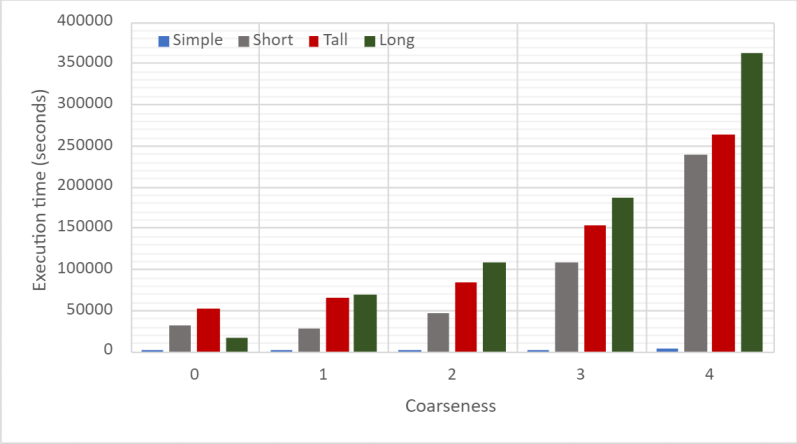

Figure 30: Comparing Execution time of the 4 different worlds, which clearly shows that all worlds follow the exponential trend where the tile size increases by $2^c \times 2^c$. Two anomalies are the Short world coarseness 0, that strangely had a longer execution time than coarseness 1, and the Simple world, that had an overall quick execution time across all coarseness levels compared to the other worlds

from finding a very optimised path if they are given enough time and time-steps. SARSA however seems to be unable to keep exploring newer areas when met with many deaths that cause a lot of negative reward. Q-Learning and Expected SARSA do much better in the regard of coarsenesses higher than 0 in more complex worlds.
As seen with the Simple and Short worlds it instead may be smarter to subdivide worlds with longer paths into subsections. Then an agent requires less time exploring this subsection saving time getting to the subsection of that world.
For further research we can also toy with sub-rewards, where agents receive a one-time reward reaching a point or section of a world, increasing the likelihood of the agent reaching newer areas. As a second point of exploration quarter steps can be expanded further. As the physics engine of the program runs in quarter steps, perhaps the inputs of the agents can be used in the quarter steps as well, where the agent can change directions and time jumps more precisely during the quarter steps. This would allow for a more flexible moveset and it would not increase the size of the state-space, as the agent exists within the world and state-space within quarter steps. Just merely acceleration has to be calculated differently, but even so requires a large rewrite of the codebase.
Another point would be adjusting the agents to better adapt to the environments, such as having an exploration focused α value during the start of the program, but later on, such as upon finding the goal it starts to focus more on exploitation by lowering the α value. To add to this we could adjust the state-space, such as to include distance to the goal or by adjusting the current state-space like shortening the state-space to only include whether a speed value is positive or negative and leave out whether the agent is touching the ground altogether. The distance towards the goal could also adjust the way rewards are calculated, where getting closer to the goal will give positive rewards and moving further from the goal will give negative rewards. Though this distance value will be calculated either through a straight line to the goal or a pathway, may it be an impossible shortest one, that leads to the goal.

References

[JGC+19] Hao Jiang, Renjie Gui, Zhen Chen, Liang Wu, Jian Dang, and Jie Zhou. An improved sarsa (λ) reinforcement learning algorithm for wireless communication systems. *IEEE Access*, 7:115418–115427, 2019.

[KBM+19] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[Lit94] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[Lor22] Uwe Lorenz. Reinforcement learning from scratch : understanding current approaches - with examples in java and greenfoot. *QA76.73.J38; QA76.73.J38 .L674 2022; Java (Computer program language); Reinforcement learning; Greenfoot (Electronic resource)*, 4:58–62, 2022.

[MJ01] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Set17] SethBling. Mariflow - self-driving mariokart using a recurrent neural network, 2017.

[VEB+17] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[ZWSZ16] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE symposium series on computational intelligence (SSCI)*, pages 1–6. IEEE, 2016.