GAMES AND ARTIFICIAL INTELLIGENCE TECHNIQUES ASSIGNMENT 2 | GROUP 1

Team members:

[redacted]

Kenrick Krause

[redacted]

CONTENTS

TABLE OF FIGURES

1.0 INTRODUCTION & GAME CONTEXT

This section will provide a brief overview of the existing game that our group implemented as part of assignment 1, in order to provide context to the work done in assignment 2.

1.1 GAME THEME

In assignment 1, our group implemented a 'haunted house' game, where the player needed to navigate three distinct stages (see Figure 1) to reach some diamonds, before returning through the same levels. Each team member of our group implemented a distinct enemy character, including finite state machine and other characteristics.

FIGURE 1: THE THREE GAME STAGES

1.2 GAMEPLAY DESCRIPTION

As mentioned in the previous section, in assignment 1 we implemented four different enemy types:

2. Ghosts
3. Rats
4. Zombies
5. Vampires

The **ghost** operated using a wandering mechanism, until a player was detected, at which point it would pursue. Instead of wander, the **rat** enemy type roamed the maps using pathfinding through predefined points, but used the seek mechanism to chase the player. The **zombie** character class utilised a flocking implementation for the general roaming behaviour as initially laid out by Craig Reynolds (Reynolds, 2001). While also using seek, the zombie also utilised a broadcasting system in which it would tell nearby zombies of the player's location. Lastly, the **vampire** would default to the wander steering behaviour, but would path find to the last known player location or to the sound of a player's footsteps and would use pursue if in line of sight to the player.

2.0 MACHINE LEARNING STRATEGY

2.1 METHODOLOGY CHOICE

After some initial discussion, our team narrowed our desired advanced methodologies to either reinforcement learning, or genetic algorithms. The advantages of the reinforcement learning approach are many, such as:

- It is a rich area of research, with many available resources to refer to.
- It simplifies implementation, with Unity ML Agents already available to use.

Some concerns about using this approach were that we would need to learn a new API and rely on the implementation. There are also concerns about adding the extra dependency of python packages.

The evaluated benefits of utilising genetic algorithm approach were:

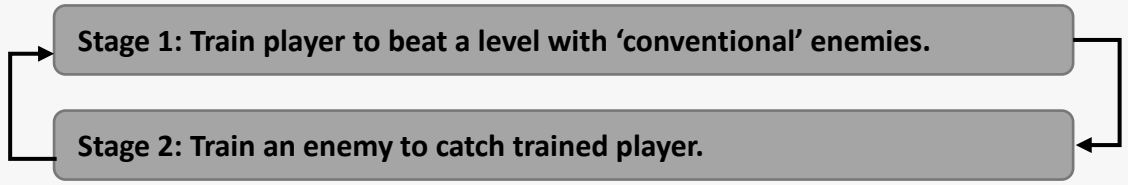
- It is a more unique area of AI, with more possibilities of novel findings
- It seemed to lend itself nicely to the zombie enemy class, with interesting possible dynamics.

With no known existing frameworks, we would likely need to 'hand roll' our own genetic algorithm solution if we were to go down that path.

Given the already significant complexity associated with training any artificial intelligence agent, we opted to utilise reinforcement learning and Unity ML Agents. If we had early successes with reinforcement learning, we would reconsider adopting genetic algorithmic approaches as a bonus addon.

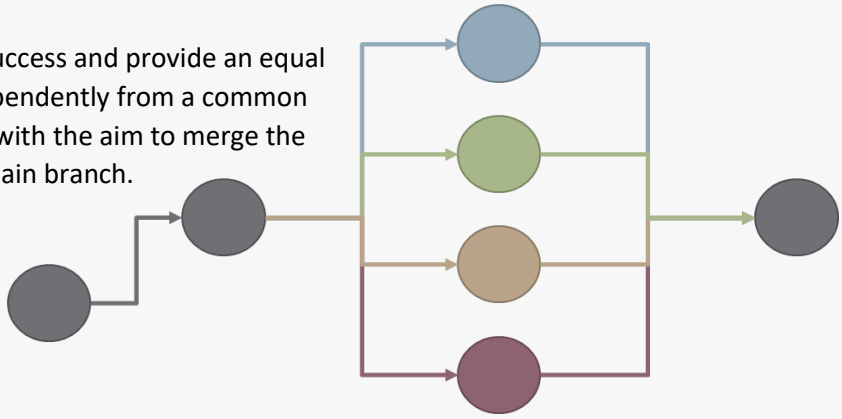
2.2 HIGH-LEVEL APPROACH

Once we had decided on the methodology, we needed to set an overall goal and strategy to reach it. Our idealised goal was to re-implement our existing game as a simulation, whereby the player and all enemy characters would be trained using ML Agents to adopt similar behaviours to their hand-crafted counterparts. To do this, we looked to use an iterative approach:



The specific mechanics of this were not agreed upon, such as whether we would try to train one enemy at a time or all at once.

Finally, to maximise the likelihood of success and provide an equal workload, we decided to all work independently from a common base branch with alternate strategies, with the aim to merge the most successful attempts into a final main branch.



3.0 IMPLEMENTATION

This section will provide an overview of the specific implementation details used across the group, including the setup of the agents, and various learning strategies.

3.1 BASE BRANCH

Before the team got to work, we first set up a common 'base' branch from which we would all branch off to try different learning techniques. In order to do this we made a copy of the first graveyard level, and made the following alterations:

- Removed the end stage trigger.
- Added a 'goal' prefab object for the player to reach.
- Created a new player movement script (*PlayerMovementML.cs*) and replaced the existing player movement script component.
- Added behaviour parameter, decision requester, and 2D ray perception sensor components to the player.
    - This included both discrete actions and continuous actions.

In the new player movement script, we started with four main reward mechanisms, detailed in the code snippet in Figure 2:

1. A discrete reward of '1' for reaching the goal.
2. A discrete punishment of '-1' for being attacked by an enemy.
3. A smaller reward and punishment conditional on the player moving towards or away from the goal.
4. A small punishment for hitting obstacles.

```
void HandleObstacleHit() {
    AddReward(-0.01f);
}

void HandleEnemyAttack() {
    AddReward(-1f);
}

void HandleFindGoal() {
    AddReward(1f);
}

void checkGoal() {
    if (Vector2.Distance(rb.position, goal.transform.position) <
        Vector2.Distance(last_position, goal.transform.position)) {
        // if we are getting closer, give small reward
        AddReward(0.01f);
    } else {
        AddReward(-0.005f);
    }
}
```

FIGURE 2: INITIAL REWARD SCHEME

4.0 RESULTS & LEARNING STRATEGIES

While the base branch we created was working in a functional sense, early tests showed that a lot would need to change to train the player AI towards our original intended behavioural goals. As previously mentioned, we took a heterogenous approach to developing strategies to improve the player performance. This subsection will detail some of the more notable strategies, including some (of the many) that did not achieve the hoped-for improvements.

4.1 REMOVE ENEMIES

In various tests, it became clear early on that the enemies as implemented were too aggressive to allow for any substantial learning. The passive goal (avoid enemies) was going to overwhelm the primary player goal (reach the end of the level). Figure 3 shows the small distance the player was covering after several iterations of learning:

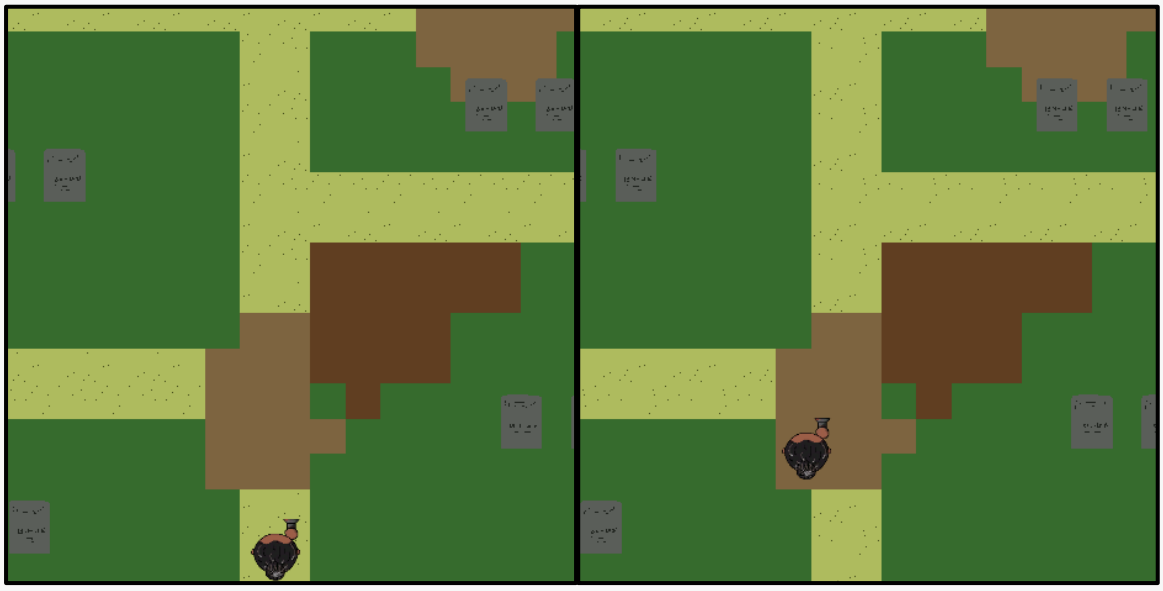


FIGURE 3: OVER-CAUTIOUS PLAYER

Figure 4 shows the cumulative reward in this situation drop slowly, and then faster as the player starts to move less and is more easily attacked by enemies.

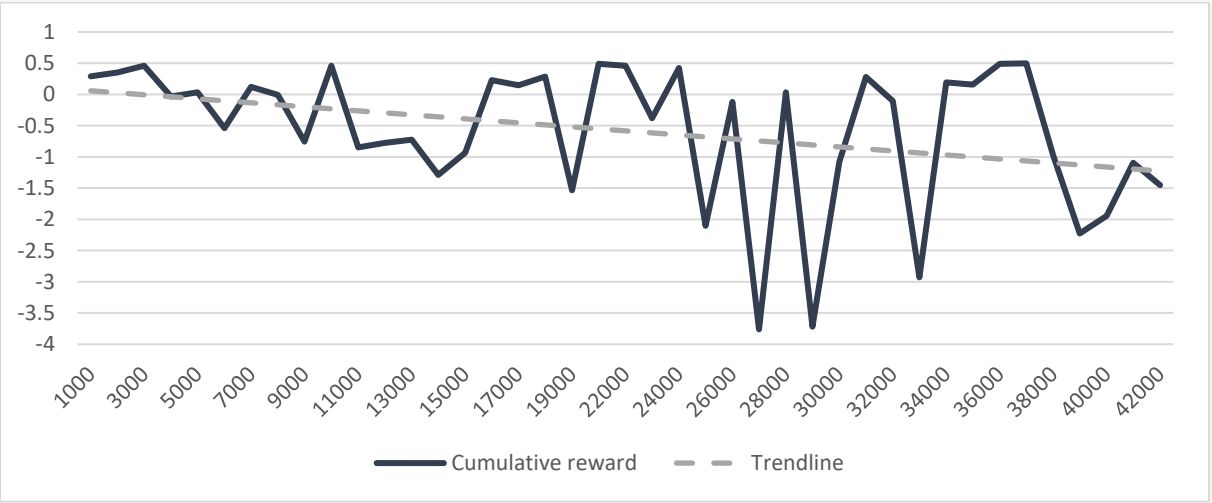


FIGURE 4: INITIAL CONFIGURATION CUMULATIVE REWARD

To get a deeper understanding, we quickly threw together a results logger to see the full distribution of rewards received by the player at the point of the `CollectObservations` call. The histogram in Figure 5 below shows largely uniform distribution, but with a clear negative average: virtually all of the rewards logged being negative.

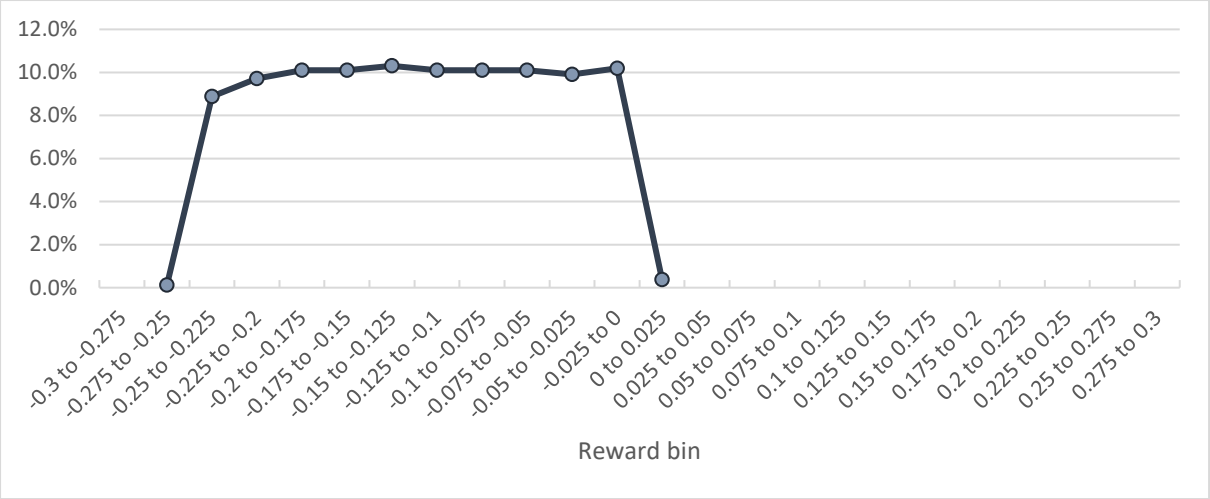


FIGURE 5: **INITIAL CONFIGURATION REWARD DISTRIBUTION**

Clearly this needed to change. To this end, all team members quickly adopted the same strategy of stripping back the scene further to remove the enemies. While this stopped the agent from amassing a large amount of negative rewards, it wasn't a cure-all.

4.2 SCALE DISTANCE REWARD

Some team members found that even though the enemies had been removed, the player was not approaching the goal. One idea to address this was to scale the distance-based reward based on the distance travelled. The idea behind this was that the player would be rewarded for traversing longer distances towards the goal, and similarly punished. Figure 6 below shows the rough implementation of this strategy:

```
float distance_delta = Vector2.Distance(last_position,
                                        goal.transform.position)
                     - Vector2.Distance(rb.position,
                                        goal.transform.position);

if (distance_delta > 0) {
    // if we are getting closer, give small reward
    AddReward(0.035f * distance_delta);
} else {
    AddReward(0.035f * distance_delta);
}
```

FIGURE 6: **SCALING DISTANCE REWARD**

This seemed to promote larger movements, but in most cases the movements were erratic. Depending relative difference between the base reward/punishment factor, the player might be 'game' the system by moving forward and back, pocketing a net reward, or even careening past the goal and similarly being no worse off.

If the reward/punishment factors were balanced, then the result was largely noise. For example, Figure 7 shows the cumulative reward against the training entropy. The entropy of this run is at a minimum at around step 45,000, even though no positive reward trend has emerged. In fact, the cumulative reward trendline has an effective gradient of 0.

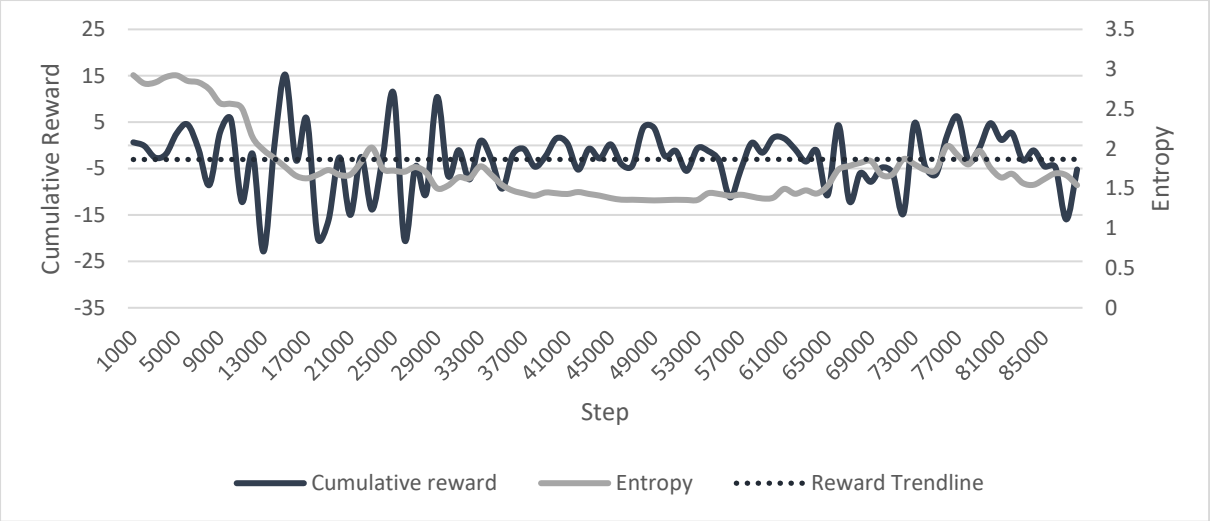


FIGURE 7: SCALED REWARD RESULTS

On further reflection, it is possible that the above results could have been improved by increasing the beta parameter for the run. However, it is still likely that the environment and reward scheme needed to be refined.

4.3 ADJUST BATCH SIZE

One team member found some success in increasing the batch size of the configuration file. Their initial test found that the player character struggled with obstacle avoidance. After doubling both the batch size to 64 and buffer size to 4096, they found the agent was better at navigating obstacles, but struggled with the final step of reaching the goal. This dynamic was a common thread for most of our team members: we could lead the agent to water, but couldn't make them drink.

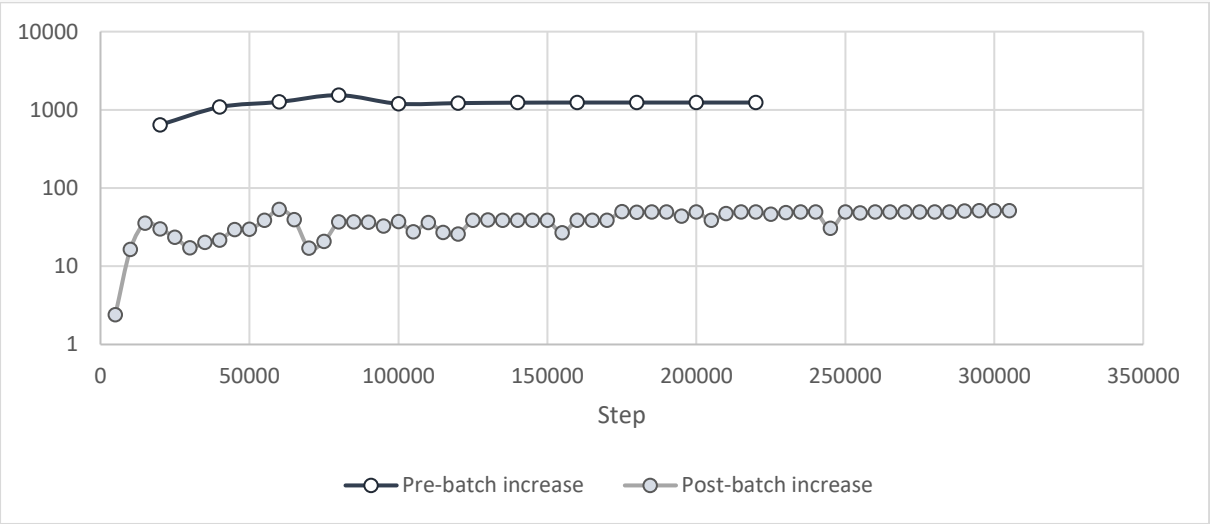


FIGURE 8: INCREASING BATCH SIZE – CUMULATIVE REWARD

Along with the changed configuration, the reward structure was scaled back. Figure 8 shows the comparative cumulative reward between the two runs. From this viewpoint, it would not appear the much has changed, emphasising the importance of observing the more experimental runs.

4.4 CONTINUOUS ACTIONS

The base ML Agents implementation we started with utilised the continuous action buffer for scaling the player speed. The idea in doing so was that with enemies present, the player would have differing speed priorities depending on whether an enemy was nearby or not.

It is difficult to say whether this strategy was working for a number of reasons, such as:

- We didn't set up a proper test to see the player actions with and without the continuous actions
- Because we included the continuous action from the start, it was operating in an already unfavourable environment.

One issue picked up with the continuous action was that the range of values seems to be (at least initially) drawn from a uniform distribution between -1 and +1. This was likely responsible for a lot of erratic behaviour initially displayed by the agent. A quick fix for this was to simply translate the continuous action by 1, changing the default range to 0-2. This was added upon with an additional scaling factor.

The ultimate effect of this was a much more ambitious agent, quickly traversing through the level. However, the agent was biased to moving towards the top-right of the level. This was thought to be caused by the continuous action values but was later discovered to be caused by a bug in the distance-based reward scheme.

The use of continuous actions was quickly abandoned by the team due to the added complexity with no clear immediate benefits. If we managed to train the player to reach the goal, we would consider adding back in continuous actions for more complex behaviour.

4.5 RANDOMISATION

One of the more successful approaches taken by team members was randomising the environment the agent was acting in. This was done in various dimensions, but the most impactful were:

- Randomising the player position.
- Randomising the goal position.
- Randomising the number of goals

There were quite a few perceived benefits to adding more random attributes to the learning environment. *For one*, it would significantly reduce the likelihood of 'overfitting' our agent to the level layout and ideally learn more generalisable behaviours. *Second*, we were having difficulty getting the player to receive enough positive rewards without continually providing distance-based rewards. By randomising the player position and number and position of goals it made reaching the goal in the earlier stage of learning more likely.

Different team members had different levels of success with this strategy. In one case, the agent still did not learn a suitable strategy for reaching the goal(s) and struggled with obstacle avoidance. In another player's experience, the player successfully avoided obstacles and learnt to search for the goal. The difference between these two experiences is likely summarised by the fact that the former team member was using the original, large graveyard level to train the agent, including many obstacles and a delayed introduction of enemy characters. By contrast, the latter player had moved to a simpler, smaller scene with fewer obstacles and their agent quickly picked up the desired behaviour.

Figure 9 shows the results from one of these more successful runs. The player is negatively rewarded frequently at the beginning of the run, but eventually learns a successful strategy and is consistently receiving positive rewards. The episode length is suitably inverse to the cumulative reward, indicating the agent reaching the goal quicker over time.

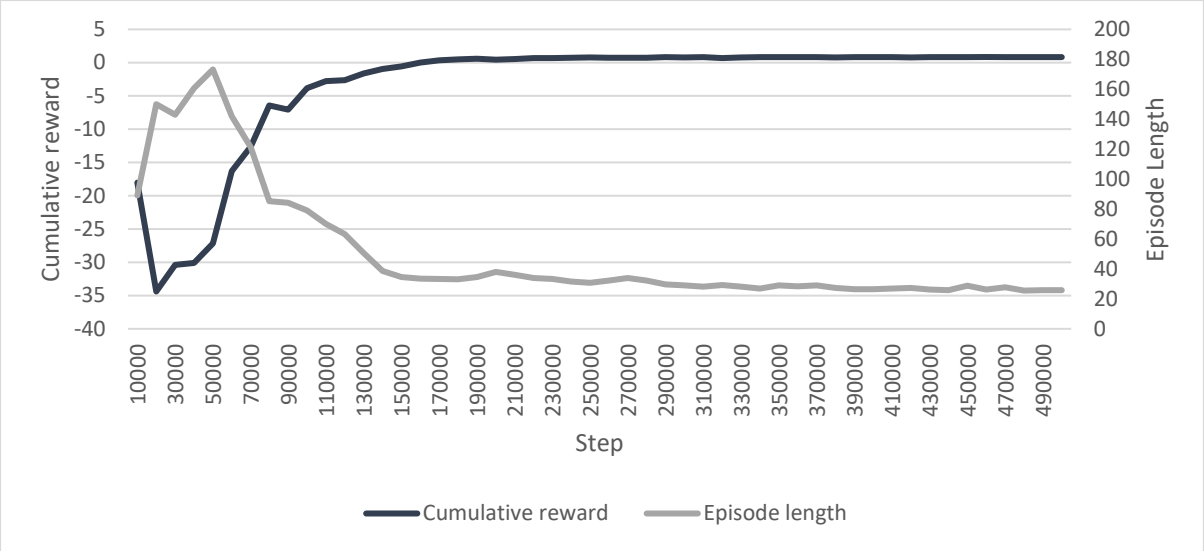


FIGURE 9: RANDOMISATION IN SIMPLE ENVIRONMENT

4.6 CHANGE BETA VALUE

The beta parameter in the configuration YAML file allows us to change the level of randomness in the agents actions. Most team members were focussed on constructing a more amenable environment and adjusting reward schemes, and not as much work was done as ideal in the area of coefficients. One small test was done in this area to see if increasing the beta value would encourage a largely stagnant agent from discovering more novel paths to reach the goal. Figure 10 shows the cumulative reward in this context of both increasing and decreasing the beta value. Generally, no strong pattern emerged in both cases, with the cumulative reward cycling in a semi-sinusoidal pattern.

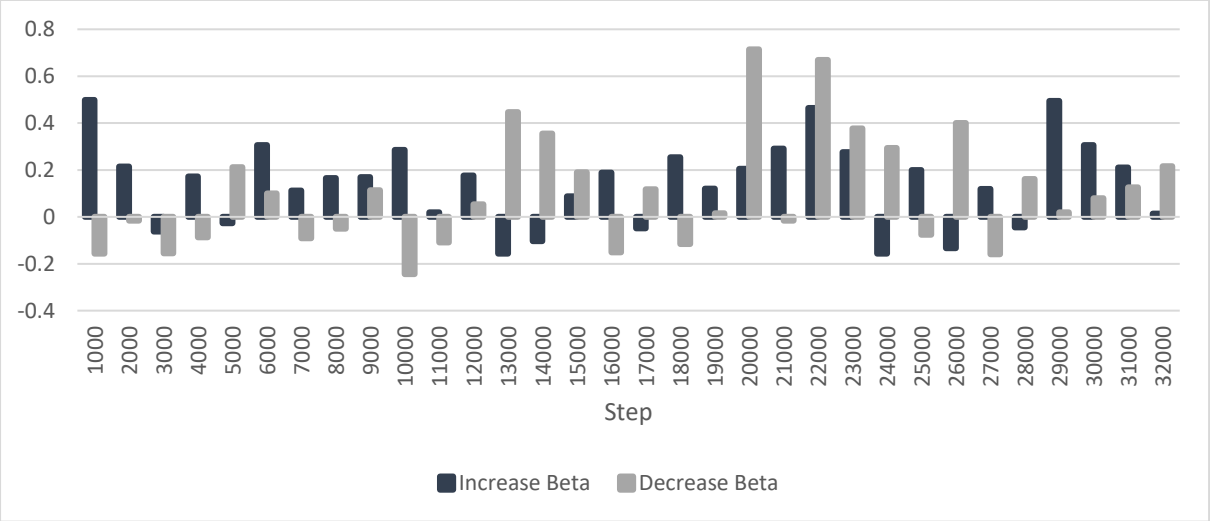


FIGURE 10: EFFECTS OF CHANGING BETA COEFFICIENT – CUMULATIVE REWARD

Figure 11 shows the entropy for the same test cases. The lack of any decrease indicates that This aligns with the lack of any pattern with Figure 10 above.

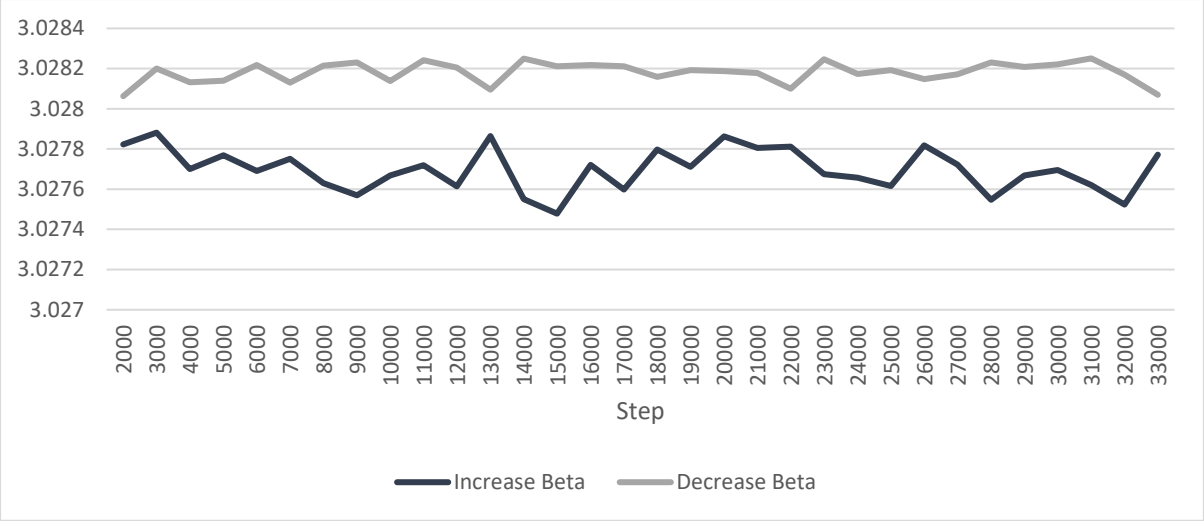


FIGURE 11: EFFECTS OF CHANGING BETA COEFFICIENT - ENTROPY

4.7 CHANGE LEARNING RATE

Another configuration file parameter that we shortly toyed with was changing the learning rate. This controls the gradient descent update step value. Updating this value is largely guesswork without a very well structured or globally concave/convex optimisation problem.

Similar to section 4.6, we only adjusted the learning rate as part of a smaller once off test. Figure 12 shows the resulting cumulative reward from the tests. Generally, if we were operating within a successful learning environment, we might expect the values from the larger learning rate to quickly increase and plateau, while the smaller rate would take longer to get to a similar optimum. Alternatively, a higher learning rate may lead to noisier results, while a lower rate could have a greater chance at finding an optimum. Unfortunately, the test we conducted was still in the unsuccessful testing stages and led to a 'garbage in, garbage out' result:

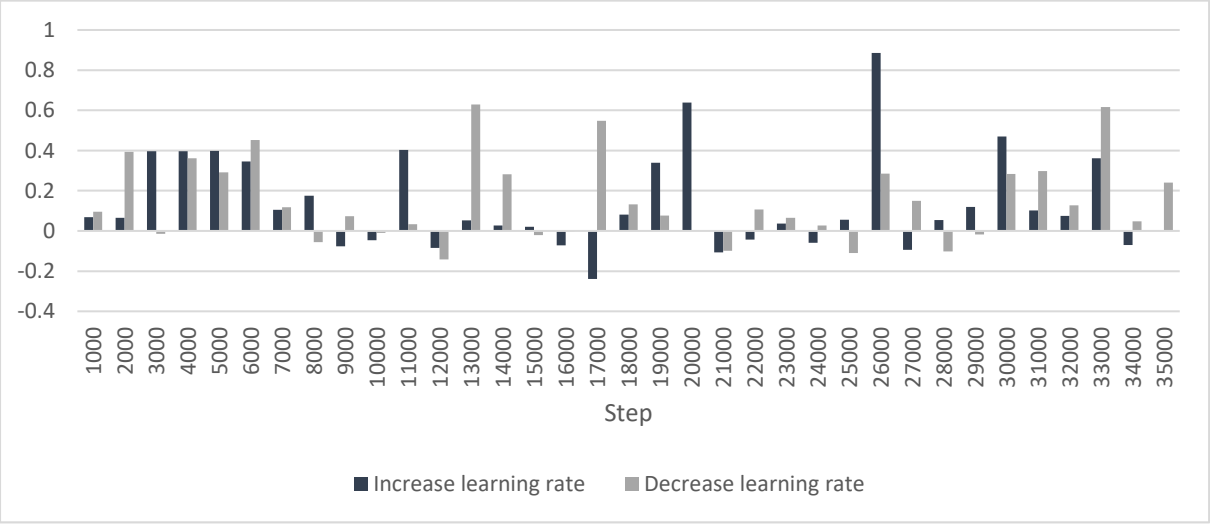


FIGURE 12: EFFECTS OF CHANGING LEARNING RATE – CUMULATIVE REWARD

4.8 IMITATION LEARNING

Imitation learning is the act of feeding in a player's inputs and making the agent learn from the player. This is used for more complex actions, and one of us wanted to use this for the agent to learn how to avoid obstacles and reach the goal.

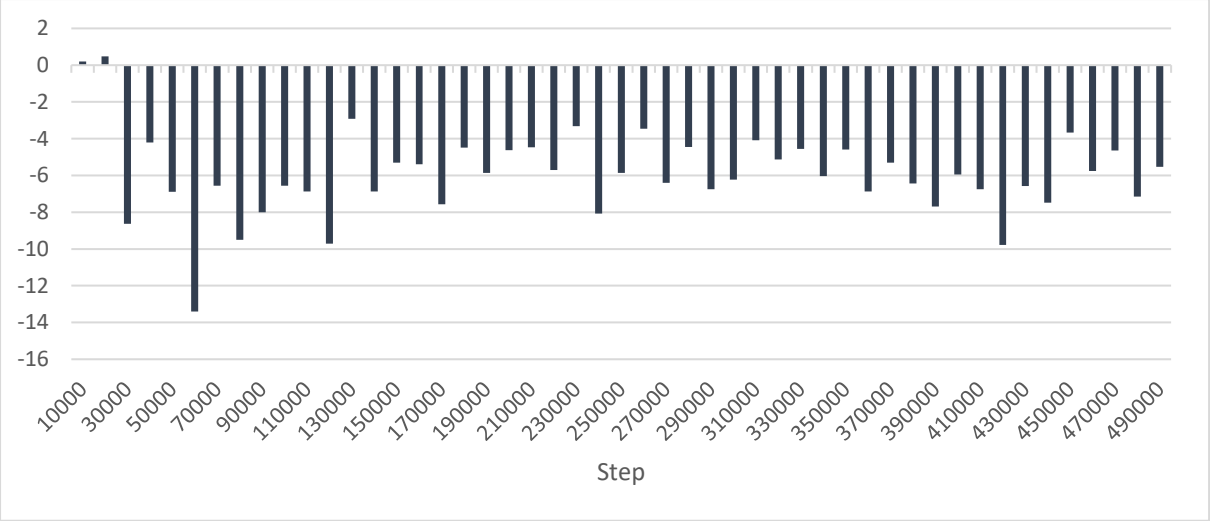


FIGURE 13: IMITATION LEARNING CUMULATIVE REWARD

After running the training with the player data, the results weren't favourable, as shown in Figure 13. There were several problems the agent displayed, such as:

- Learnt to stand still and not move at all.
- Wiggling back and forth in the same position, and only going toward the goal if the agent was close to it.

Figuring out and solving the agent's problems wasn't a favourable option as it did take a considerably long time to train the agent using imitation learning. The unexpected amount of time this have taken wasn't a reliable way to train the agent given we had limited time to work on this assignment.

If there was time, figuring out how to speed up the training would be helpful and adjusting the parameters in the config file could possibly help the agent learn better. Possibly even step back a bit and try train the agent again with no obstacles to see if the imitation learning is working correctly.

4.9 ADJUSTING OBSERVATION INFORMATION

A more successful approach was to update the method of collecting observations for the agent. Instead of relying on 2D ray perception sensors to provide observations to the agent, one team member wrote their own observations, as shown in Figure 14.

```
public override void CollectObservations(VectorSensor sensor)
{
    // Goal (Diamond)
    Vector3 offset = targetTransform.position - player.transform.position;
    sensor.AddObservation(Sign(offset.x) * Max(1 - Abs(offset.x) / 20, 0.0f));
    sensor.AddObservation(Sign(offset.y) * Max(1 - Abs(offset.y) / 20, 0.0f));

    // Rigidbody
    sensor.AddObservation(rigidBody.velocity.x);
    sensor.AddObservation(rigidBody.velocity.y);
}
```

FIGURE 14: UPDATED OBSERVATION CALCULATIONS

4.10 SIMPLIFIED SCENE

By far the most successful strategy taken by team members was to simplify the Unity scene. We have already partially discussed this in section 4.1, through the method of removing the enemy characters. However, other team members went further by departing from the original game level design and adopting a simplified scene with few to no obstacles.

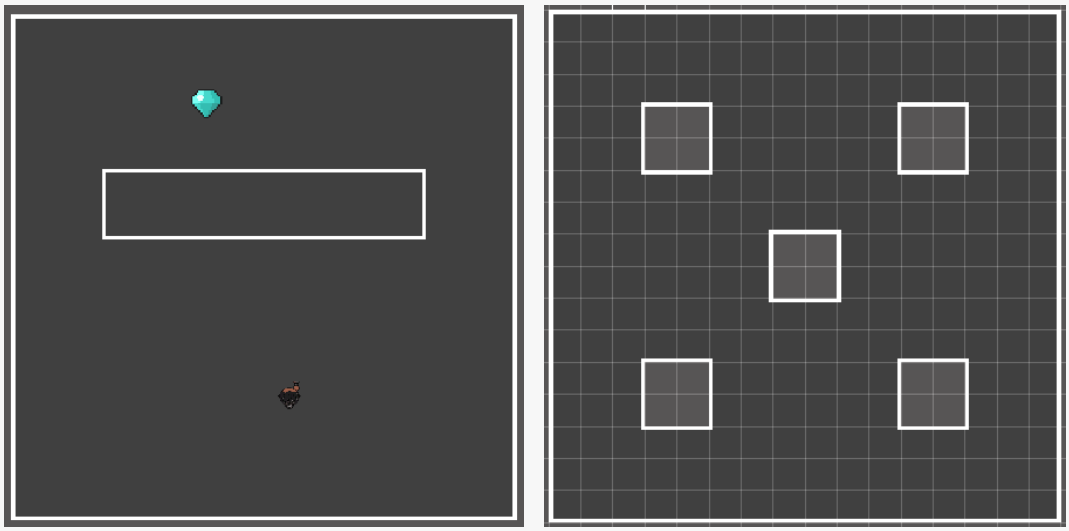


FIGURE 15: SIMPLIFIED SCENES

Coupled with random positioning and modified observations (see section 4.9), this led to the most successful implementation our group saw, with a stable (mostly) positive cumulative reward after enough iterations, as shown in Figure 16.

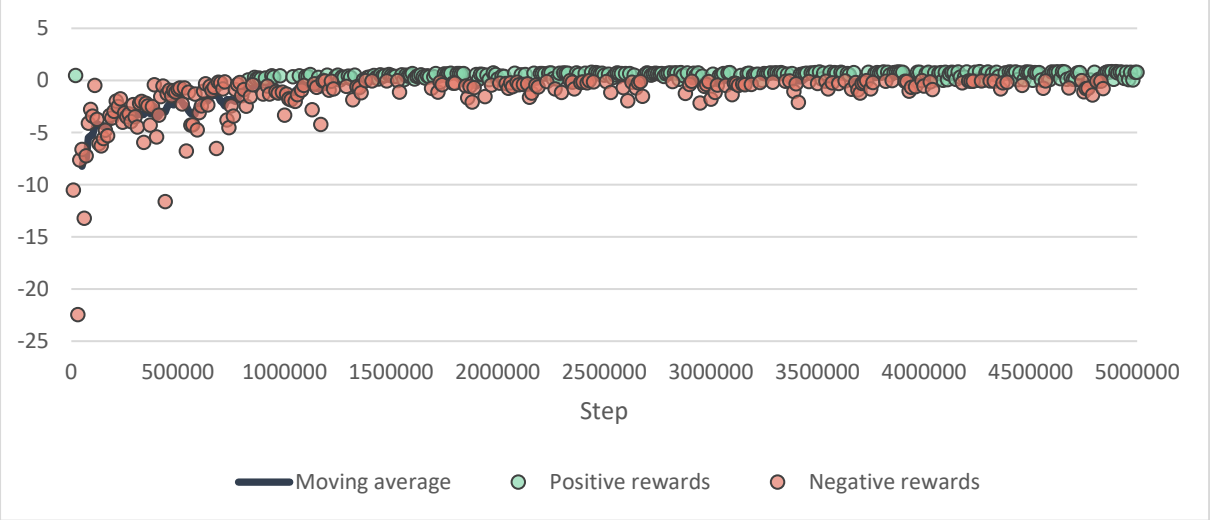


FIGURE 16: SIMPLIFIED SCENE AND ADJUSTED OBSERVATIONS – CUMULATIVE REWARD

5.0 ANALYSIS

This section will try to provide some further context to the various strategies and results shown in the previous section, and reflect on how we could have approached the assignment differently with the knowledge we have gained over the course of the project.

5.1 OUTCOMES

As the previous section shows, we did not achieve the original goal as we laid out in section 2.2. A useful question to ask is whether we could have sufficiently adjusted our strategy to achieve the desired results, or whether our original goal was too ambitious.

After some unsuccessful attempts, the team started to diverge in strategies, with some continuing the course of trying to get the player agent to work in the existing level, and others scaling back the scope of the training environment to achieve more manageable successes. The latter approach was recommended by the teaching staff, and was ignored at our peril.

It is difficult to assess whether if we had started with a smaller scope, whether we would have achieved our overall goal of a simulated semi-replica of our original game. With more exposure to and greater understanding of the mechanics involved in setting up a reinforcement learning agent in Unity, we would likely scale back our ambitions if we were to redo the assignment today.

5.2 LESSONS LEARNT

Throughout the assignment, we have made various mistakes and had varying levels of success using different approaches and philosophies. As a result, we have come to some core lessons learnt, which we have presented below.

Scale back and simplify: one of the most important and easy to ignore pieces of advice. Perhaps because the ML Agents framework took care of a lot of work for us, it was surprisingly easy to underestimate the task of training an agent for what appeared to be a simple set of tasks.

Be absolutely sure system is working: a good portion of time (more than we would care to admit) was spent trying to analyse the behaviour of the agent that resulted from bugs in the core agent implementation. This ties in to the above point of simplification, create something that works and build on top of it.

Create more structured tests: using a more ad-hoc process of changing the state representation has its benefits, largely it allowed the team to get started quickly without too much prior knowledge of the ML Agents system in theory and in practice. However, testing the effect of a parameter change or adjusted reward architecture should have been done more frequently and in relation to a working ‘base’ to compare against.

More formal testing naming: after a few tests have been performed, it is easy to start to lost track of what which test was attempting, what changes were made in the C# script, and what the visual effects were. Further to this, we also found that we should have:

Log agent position for unsupervised testing: when trying a more unstructured approach, or when we feel more confident about a setup, we would let tests run for longer. This meant we would often leave the training run without supervision, and come back to unexplainable behaviour. Logging the agent position, and key events would allow for more formal analysis of the emergent behaviour.

6.0 CONCLUSION

This report has gone through our attempt at integrating machine learning – in particular reinforcement learning – into our pre-existing game. We set a high goal of trying to train not only the player character, but also one or more enemies. We proposed to do so through an iterative process, whereby we would train the player to play our pre-existing game, before training the enemy characters on the simulated player agent.

As this report has demonstrated, we were a bit over-ambitious in this goal, and quickly scaled back the scope of the project in response to a greater understanding of the Unity ML Agents framework and its limitations. We have provided an overview of the various key strategies employed by our team, from individuals to the entire group. Some of these strategies were successful, while others floundered, either due to a poor theoretical foundation or due to an overcomplex learning environment.

As a result, we did not achieve our stated goal, and were limited to the success of training an agent to reach a 'goal' object. We have learned many lessons in this process, and if we were to attempt this assignment again today, would greatly simplify our approach and stated goals.

7.0 REFERENCES

Reynolds, C. (2001, September 6). *Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)*. Retrieved from Reynolds Engineering & Design: https://www.red3d.com/cwr/boids/