Augmentation of human behaviour in crowd simulation using reinforcement learning.

Caolan McDonagh
School of Computer Science
National University of Ireland Galway

Supervisor(s)
Karl Mason

In partial fulfillment of the requirements for the degree of
MSc in Computer Science (Artificial Intelligence - Online)
August 26, 2023

DECLARATION I, Caolan McDonagh, hereby declare that this thesis, titled "Augmentation of human behaviour in crowd simulation using reinforcement learning", and the work presented in it are entirely my own except where explicitly stated otherwise in the text, and that this work has not been previously submitted, in part or whole, to any university or institution for any degree, diploma, or other qualification.

Signature:

Abstract

Crowd simulation is the recreation of human crowd behaviour and movement in a virtual context. These simulated crowds are made of virtual agents, that can either be individually simulated (Microscopic, the approach this report will lean towards) or simulated as a whole, like a flock/swarm (Macroscopic). These agents should have accurate object/agent avoidance algorithms (Such as optimal reciprocal collision avoidance (ORCA)) to avoid collisions with obstacles such as other agents or barriers like walls [1]. The agents should also have some sort of navigational algorithm for getting from A to B, such as the A* search algorithm for finding the best route.

These object avoidance and navigational algorithms are at their core "robotic" as that was their early use case. In modern use cases, we want the most realistic crowd simulations for use in video games, other forms of entertainment such as films and TV, urban engineering, emergency simulation or behavioural sciences. This project will investigate the potential application of reinforcement learning in the use case of simulating realistic human behaviour in navigation and avoidance in both singular agents and crowds of agents. This report will attempt to outline the usefulness of RL inside of the different areas of crowd simulation, using empirical evidence to note if it is worth using RL for crowd simulation. Comparing this to a "classic" implementation of agent navigation and avoidance, we hope to see potential realistic

simulations and emergent behaviour.

Keywords: Crowd Simulation, Reinforcement Learning, Human Behaviour, Avoidance, Navigation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this report is to investigate the potential effectiveness and use cases of using RL to take existing crowd simulation methods, specifically the navigational and avoidance algorithms and bolster their "humanlike" traits. Crowd simulation is a potentially very computationally intense simulation to run, with potential 100's to 1000's of agents all attempting to both navigate a given world, while also avoiding other agents and obstacles [3].

Due to the computationally intense nature of crowd simulation, the algorithms involved need to be smart, this does not inherently mean they are human-like. Especially when stacking algorithms e.g., having an initial navigational algorithm with an avoidance algorithm overlayed to get from A to B most optimally. Human nature isn't always optimal, especially in crowds. Crowds tend to work as a single organism when they become very dense. RL could potentially combine these two algorithms [4] and allow for a much more unified solution to helping crowd simulation become more human-like. Not only this, but by combining algorithms there could be potential performance gains or losses depending on how well agents react to an RL implementation for navigation and avoidance.

Another potential gain of using RL is the learning system it offers. With the

ability to customize and designate your reward system, you can potentially create branches off one core crowd simulation RL algorithm that involve certain environments or situations. For example, we have a branch in the crowd simulation RL algorithm that is for use in emergency simulation for testing a new shopping centre fire exit [5] [6]. This reward system is altered to allow for more selfish behaviour in agents, such as less forgiving avoidance towards other agents. Reinforcement learning has the potential to expand the abilities and realism involved in crowd simulation.

RQ1: The first question we must start with is can RL be used to recreate human navigation and avoidance behaviour in an agent? This is the main hurdle we need to jump before we can apply RL to multiple agents/crowds. Here we may investigate RL for enhancing the smoothness of navigation, avoiding sharp adjustments to movement to make them less robotic looking. We can also investigate obstacle avoidance in a similar fashion, where agents don't sharply adjust course when too close, but rather from a distance make smooth, short adjustments to ensure they pass by without interacting.

RQ2: Is this RL recreated human navigation and avoidance applicable to multiple agents/crowds? We want to make sure that if RL is successfully implemented for one agent, will it work well when scaled up to dozens of agents.

RQ3: Is it worth it? While it would be amazing to see RL correctly augment human behaviours in crowd simulation, is the work involved worth it? Do the enhancements happen on a noteworthy scale where we can tell a difference that one simulation is better than a non-RL simulation? Is it performant? As stated before, crowd simulation can be incredibly computationally intense, and existing algorithms are performant. Will an RL-enhanced version be as performant or will be much worse? We can combine these two points and check if the performance loss (If any) is worth the visual behaviour change in the agents. The empirical

evidence gathered during the course of this can help us answer these questions. Repeatable simulations can be run with RL essentially toggled on/off and we can monitor the agent interactions and avoidance, as well as performance metrics to get a good view of all of the above-mentioned points.

Chapter 2

Related Work and Background

Crowd simulation is a relatively niche area, but Reinforcement learning and human behaviour is not. The core papers and journals were found via ScienceDirect, ACM Digital Library, Google Scholar, and the James Hardiman Library. There are only so many keywords to look for, such as "Reinforcement learning", "Multi-Agent Reinforcement learning", "Human Behaviour (with/without RL), "Crowd Simulation" and "Obstacle avoidance (with/without RL). Brief reviews of the paper's abstract would let me know if they were related at all to this report, and if they were I would carry on with the read. As previously mentioned, crowd simulation is not a hugely researched topic, and due to this, I was rather relaxed about my paper reduction. There are limited papers that dip into both crowd simulation and machine learning, so these were not so easily removed. Anything of obvious low quality was, but that is a given. When it came to general reinforcement learning and multi-agent reinforcement learning, there were many more papers to review. The main factors here for filtering were quality, relevance to agents (independent and co-dependant), citations and scores. After these initial filters, I concluded with the intro and abstracts to see if they could be helpful to the research. The first papers to review were those that fell under the keywords

for “Crowd Simulation”. This is because to understand an approach to applying RL to crowd simulation, we need to properly understand the crowd simulation itself. A name that consistently came up in this field was Daniel Thalmann. He has written “Crowd Simulation” [7] which would be considered to be one of the core books in the crowd simulation field. He is highly cited in several papers relating to crowd simulation and group collisions. “Crowd simulation” is a good base for a comparative study that needs to be carried out. The core of what makes good crowd simulation can be divulged by this text, as it specifies everything on macroscopic and microscopic crowds down from overall movement behaviours to the human psychology in people within crowds.

Thalmann and Musse also wrote a paper [8] on the interrelationship of groups and collision detection analysis. This is more so around the modelling of a crowd. As stated in the paper there are few studies on crowd modelling, most existing papers were around flocking in fish and birds. In this paper, they want to model groups of individuals (Humans) based on local rules. This can very easily translate to reinforcement learning rules even though this paper has no mention of AI or ML. As per the quality crowd simulation papers, this brushes on sociological aspects of large groups of humans. This is great information to feed into RQ1, as to be able to accurately answer the question of whether is RL appropriate for human behaviour within crowds, we need to accurately determine the human behaviour seen in crowds. Model-free learning for human behaviour is possible and works well [9], this helps show how RL can effectively be applied to crowd simulation on the individual agent level. There is another paper [10] by Thalmann and Musse, where they model the real-time simulation of crowds through a hierarchical model. This hierarchy is virtual crowds, groups, and individuals. They review the behavioural rules required to allow for more realistic animation. Again, these rules can be used to account for the rules in a potential RL setup in

RQ1. This paper also feeds into individuals rather than group behaviour (Microscopic crowd), which in turn can feed into RQ2. Can the RL of a single agent be used to successfully simulate a crowd? This paper outlines the promising results of individual behaviours, which is promising when related to multiple a single RL agent, to multiple agents.

The papers that revolve around RL and multi-agent RL and crowd simulation are again few, due to the nicheness around crowd simulation. Out of the handful that exist, some very well-written papers help towards the goal of the report. [11] This paper on improving existing multi-agent reinforcement learning when it comes to path planned in crowd simulation is almost a direct answer to RQ2. It establishes that RL is effective in path planning for crowd simulation, but there are some hurdles when it comes to the mutual influences these agents have on one another when using RL, and they should be adjusted to account for this. This paper introduces IMARL (Improved multi-agent reinforcement learning method) to combat this. It makes good note of problems in RL, such as the dimensionality disaster and improving convergence. Other papers can hint towards implementing good quality rules for an RL set-up, in the form of parametric reinforcement learning [12], which can directly link to setting up a good base for RL in crowds.

Several of the reviewed papers also hover around RQ3, explicitly showing their findings on whether an RL approach is better [12] than the "standard" model [13] approach offered by standard crowd simulation. The papers available to us have answers in both the non-ML model approach and the results of RL approaches to crowd simulation. From here results can be compared to try to establish the pros and cons of using this RL approach to crowd simulation. A key thing to remember here is that this is all real-time and so it is not just superficial results that are being investigated. Performance is still a large factor [14] and needs to be correctly analysed and tested on an RL approach. The initial approach to

this report is that a microscopic crowd simulation (Single agent level) should be used, but it may be worth comparing the results in independent and cooperative agents [15]. We could see potential performance gains this way, but it may not result in the optimum human simulation we are looking for, as usually we see crowds approach on flocks in the macroscopic form. [14]

Of the three RQs I proposed, the literature prepared is sufficient in answering these RQs for the most part. RQ1 asks can RL be used in modelling agents for crowd simulation, and there are a few high-quality papers that directly tackle this question and produce good results. RQ2 is potentially inclusive of RQ1, where the question is raised, can a successfully RL-trained agent co-operate with other RL agents to create realistic crowds? Again, papers that directly answer RQ1 help here, while some other papers bridge the gap on multi-agent reinforcement learning and crowd simulation. The papers gathered are sufficient in my opinion, to show the possibility of reinforcement learning becoming very applicable to recreating human behaviour [9] and crowds [4]. RQ3 is a bit more open-ended, but a few of the papers lead me to believe an RL approach to crowd simulation could be worth it. With the "old" model-driven approach to crowd simulation, RL opens the doors for new possibilities, scalability, and more dynamic reactiveness in the crowds (alternative behaviours, e.g., panic, very dense crowds etc.). They can potentially escape the realm of just being roaming robots to accurately recreat human behaviour down to the agent, expanding the usefulness of crowd simulation and presenting much more accurate and dynamic crowd simulation. Overall, this report has shown the existing literature in these areas can be combined and used to go forward in the field and find appropriate applications of reinforcement learning/multi- agent reinforcement learning to augment the human behaviour seen in crowd simulation.

Chapter 3

Methodology

All of the experimentation, testing and implementation is done via Unity. Unity is primarily a game engine, but its powerful scripting integration using C# and integration of packages allow for my objective of crowd simulation and the use of reinforcement learning all with the powerful visualization features of the game engine to conduct thorough research. To create a base comparative approach, I will make use of Unity's built-in "NavMesh agent". This is a simple agent that makes use of a Navmesh, which is created via baking the geometry in place, creating a navigable area that can be carved, meaning obstacles, walls etc. will not be included in this navigable area if so configured to do so. The agent can then navigate this area using a simple A* algorithm, finding the optimal path to its goal. In the below, we can see an example where blue is navigable and the walls carve away from this navigable area.

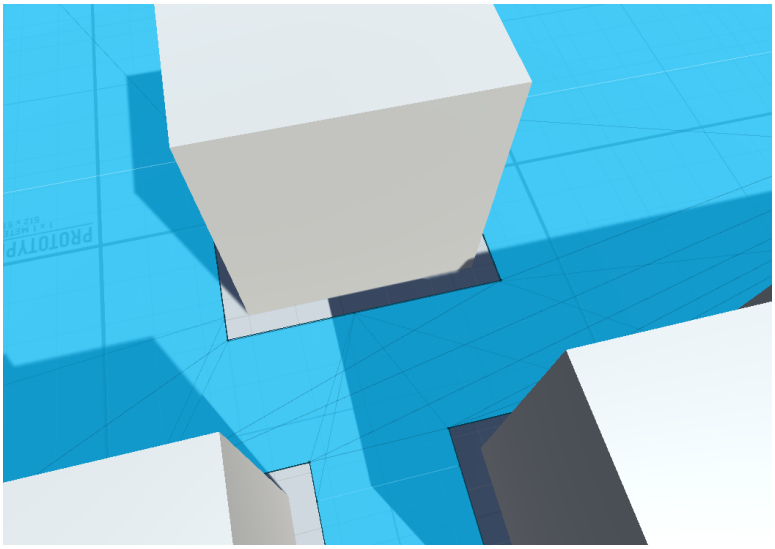

Figure 3.1: Unity NavMesh example.

These NavMesh agents also have support for obstacle avoidance via RVO, reciprocal velocity obstacles. This allows the agent to calculate a velocity obstacle for each object within its defined range. These are a representation of the possible velocities in which a given object would result in a collision, which it can then use to adjust its velocity to avoid potential collisions. Below is an example of this, where the bottom right blue agent's RVO visualization is displayed. The red arrow represents the direction the agent needs to travel, and the blue arrow represents the adjustment to be made to avoid collision. The white and red boxes represent potential collision velocities, where red is a potential collision and white is safer. Even though the agent is close to the neighbouring blue agents, as it can read their velocity, it knows it is moving in the same direction, preventing a potential collision if it follows.

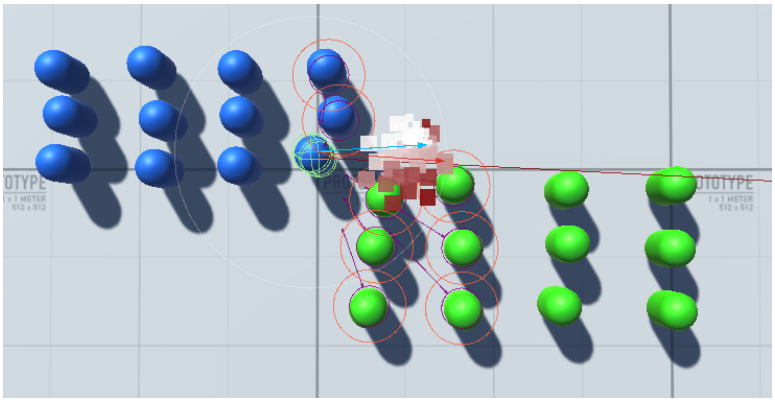

Figure 3.2: RVO Visual.

This is very functional for many use cases such as video game NPCs, where the ability for rough realistic behaviour is acceptable and works as a great comparative for an RL implementation, due to it being the ”normal” approach for such use cases. For this implementation, Unity developed the ML agents toolkit [2], which is an open-source package that allows for PyTorch integration into the engine.

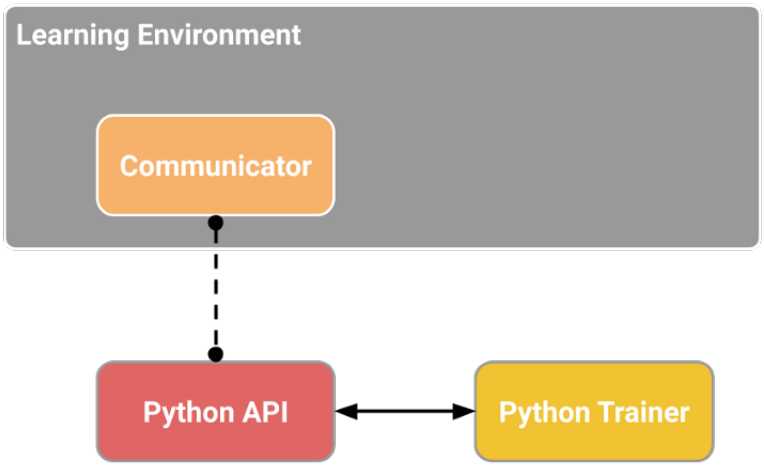

Figure 3.3: Unity ML-Agents Toolkit [2]

To kick off an implamentation to reinforcment learning, I had to first explore suitable algorithms.

- Proximal Policy Optimization (PPO)
- Deep Q-Network (DQN)
- Asynchronous Advantage Actor-Critic (A3C)
- Multi-Agent Posthumous Credit Assignment (MA-POCA)

First the above were all analysed for the applicability towards the research questions and to ensure compatibility with the actions I had planned to be available to the agents. Initially, DQN was disqualified from making it to scenario tests due to the potential complexity involved in a multi-agent scenario for avoidance and navigation. The potential randomness between the agent/goal/obstacle position and the number of agents present could lead to instability and further issues in training. On top of this, DQN is based around centralized learning [16] which could lead to problems such as delayed decision-making and issues in the scalability of crowds. Overall it leads to potential issues in crowd simulations of varying sizes, removing itself from the pool of potential RL algorithms. PPO and A3C shared one drawback, and that was a lack of communication between agents. I believe for genuine replication of human crowds this is a requirement. Each agent is an individual trying to navigate, but in areas of dense movement, crowds effectively work as one entity [17] [18]. You may have experienced this in person when navigating densely packed areas such as concerts, high-traffic university buildings and museums. This left MA-POCA for implementation.

MA-POCA has many benefits for this application, such as inherent support for agent creation/destruction on a per-episode basis while sharing a reward function [19]. This is useful for making use of varying the agent count per training episode, to let the agents learn in both sparse and densely packed environments. MA-POCA also makes use of decentralized execution and centralized training [19]. This allows agents to act based on their local states, but they are privy to all

globally available information during the training. This was going to be useful for allowing agents to navigate the environment as they see best while keeping the common goal between all agents to avoid each other in a cooperative manner, as to conform with before mentioned crowd flows. This is handled by a central critic who is informed of all agent actions and observations (where observations = states). It can use this information to consider the combined behaviour of all agents and provide optimal centralized value estimations. This value can then be used to allow better adaption by the agents to the current environment scenario, e.g. potentially forming lanes to pass through narrow hallways etc. MA-POCA also tracks this via a "Group Reward".

To begin tuning the MA-POCA algorithm, initial test areas were established. This included Unity GameObjects to represent the agent(s), wall(s), obstacle(s), floor/base, goal and environment centre. Unity allows for the "tagging" of game objects, which groups them under a common reference. E.g all agents carry the tag "agent". These can then be referenced as required in the code.

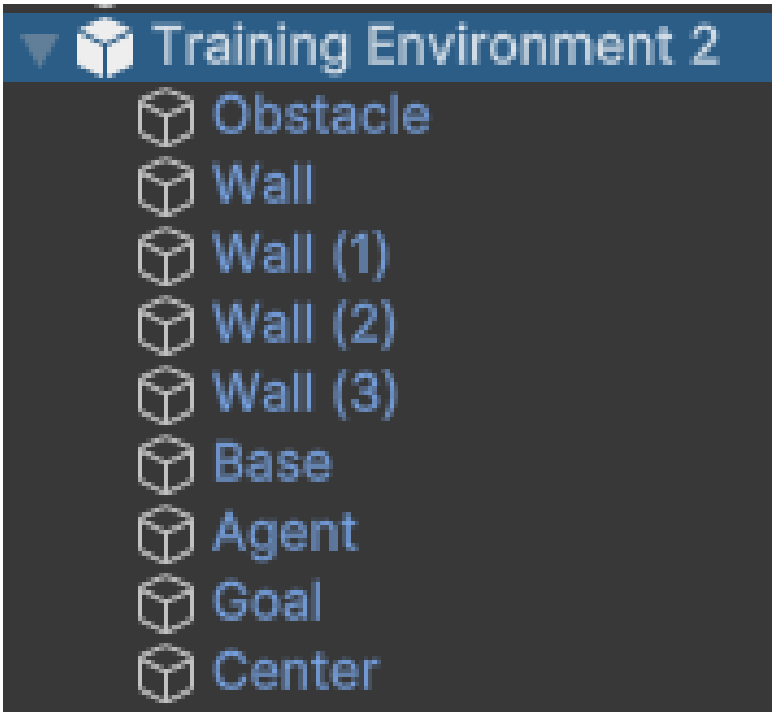

Figure 3.4: Training Environment Hierarchy.

I wanted crowds to be able to function within dynamic environments, to do this I had to build multiple environments for testing encompassing a few key scenarios.

- Large open areas
- Narrow hallways and intersections
- Emergency exit scenarios
- Random/changing environments of all of the above.

I could build out these environments using the above-mentioned game objects and create encompassing environments. From here I needed to also take into account that crowds are almost always unique in how they flow. The inclusion of randomness in the environments can help in training agents for these scenarios

and avoid picking up bad behaviour traits. The agent will need to navigate to a goal, avoiding other agents, obstacles and walls. Walls are static and are created for the given scenarios listed above. Obstacles, agents and goals are scripted to spawn in random positions from the centre of the environment upon the start of each episode. These are sequentially spawned in, with location noted and checked to avoid objects spawning in the same coordinates (e.g. a goal inside of an obstacle) via a set buffer zone. Agent count also increments from 1 to 30 agents and reverses to allow for adaption to denser crowds.

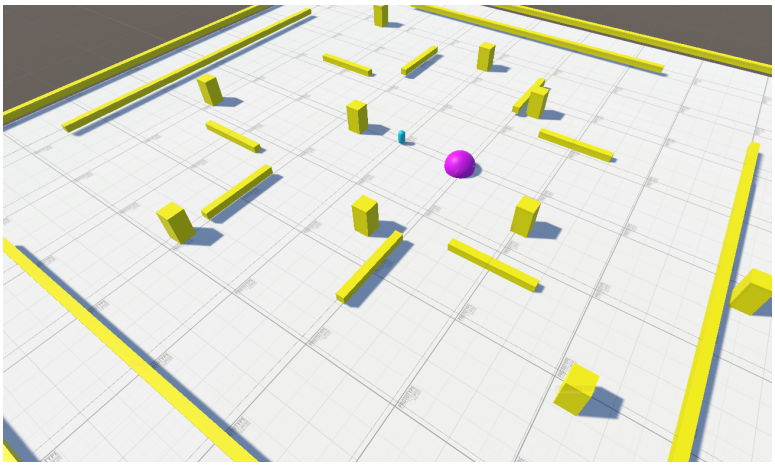

Figure 3.5: Training Environment Example: 10 obstacles (yellow), an agent (blue) and the agent's goal (purple)

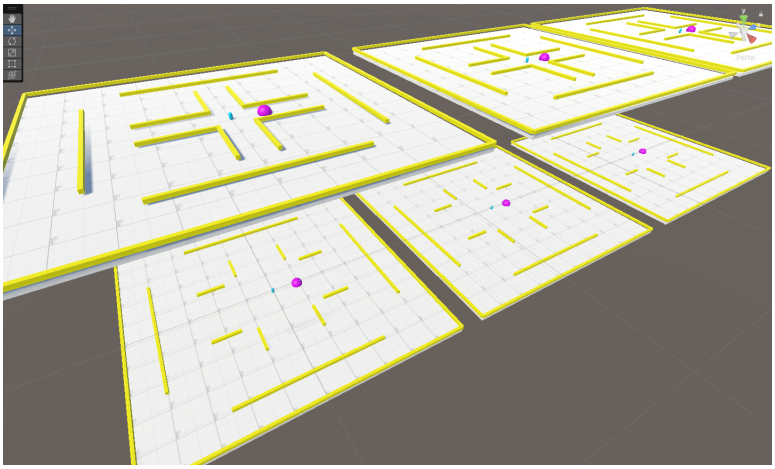

Figure 3.6: Training Environment Example: Alternate static room layouts.

The actions are continuous and kept simple for agents, they only need to be able to move on a 2D plane. MoveX and MoveZ were the available actions that dictate the direction the agent moves. To ensure realistic movement, these two values were clamped to a range of [-3,+3], where +3 is a relatively fast walking speed. These values then create a "moveDirection":

```
1   Vector3 moveDirection = new Vector3(moveX, 0, moveZ).
       normalized;
```

This is then used in creating a new velocity, which is smoothed to create more realistic movement amongst agents. Being continuous, these actions encompassed all required movement including slowing/accelerating, turning and coming to a stop.

For rewards, several things had to be accounted for. These are briefly shown below.

- Reaching goal: +5
- Touching another agent: -3 (Global and Agent)
- Touching wall/obstacle: -1

- Time without touching agent: +.01 (Global)

The main goal of the agent is to reach its goal, this is the highest reward. Touching agents is a more severe negation and touching walls/obstacles is not as harsh. MA-POCA can allow the definition of "global" rewards, where agents as a whole are given rewards for their combined efforts. In this case, I chose the toching agents as both a local and a global negative reward, and as time goes on a very small positive reward is awarded for not touching any agents. This keeps the entirety of the crowd alerted to avoid each other and work together to do so. I chose to not include any negative reward for time taken or distance (although I did cap the steps of an agent per episode to 10,000, to avoid long training/sticking) as this could discourage longer routes around potential bottlenecks or dense areas and allow for a more "human" element to the agents, in the sense they are not punished for meandering in an area or exploring the environment. They need to reach the goal, but not as quickly as possible as in most cases, this does not reflect human behaviour. These sets of rewards were tuned along with parameters to keep the agents reacting in a realistic manner, avoiding any strange movement or decisions when navigating the test environments. The above looked to give the best results when measured empirically.

The states (or observation as I will refer to them as) are presented in these environments and is generally dynamic. The below list shows the potential observations present:

- Agent's current position
- Goal's current position
- Distance from agent position to goal position
- 15 ray casts from the agent

The goal position is static for each episode, whereas the agent position and distance to the goal are updated as it moves through the environment. The 15 ray casts are spread 180 degrees around the agent and travel to a distance of 40 units. These raycasts only report back given detectable tags: [Agent, Obstacle, Wall, Goal]. Anything tagged with these are relevant observations for avoidance and navigation. This ends up totalling 18 observations or states per agent. Initially, it was just agent/goal positions but as testing was carried out, more observations were added to ensure the agents had enough information about their surroundings to learn from their observations and adjust accordingly.

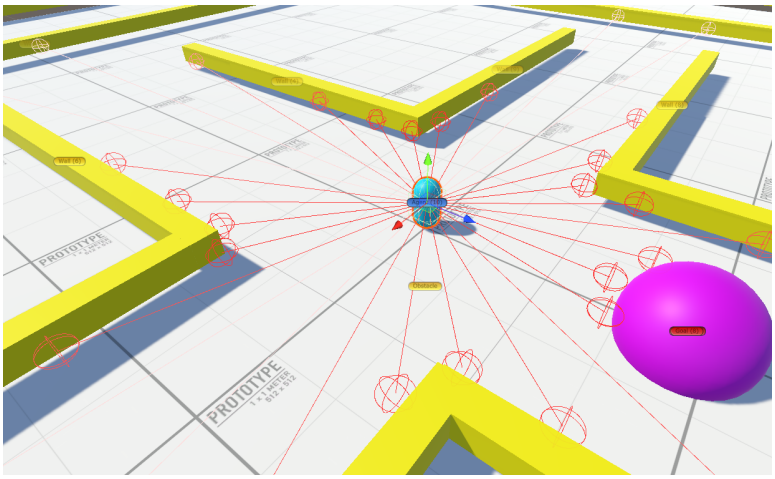

Figure 3.7: Agent Raycasts.

To collect the data and observations required to judge if our RL implamentation is working, standardized tests are created post training. These tests will introduce targeted scenarios to create a comparitive between both NavAgent and RL agents. In single tasks of navigation avoidance, and in tasks involving multiple agents. Each test area is create in unity and each agent is set exactly the same for both RL and NavAgent. Unity allows us to toggle on/off GameObjects, such as a group of RL agents and NavAgents are placed in the same environment at the same coordinates, one set can be disabled and when the simulation is ran, only

the required set of agents will act and be monitored. This can then be swapped and the exact same scenario will play out with the other set of agents allowing for observations to be recorded and any numerical statistics to be captured without adding potential advantages/disadvantages through the initial environment conditions on initialization. These tests will be explored further in the results chapter.

Chapter 4

Experimental Settings

Initially, one agent was tested to align with RQ1, ensuring that using reinforcement learning (Specifically MA-POCA) can allow an agent to learn the core ideas of navigation and avoidance. A simple test environment was initially used, placing a single agent and a single goal randomly in an enclosed "room". No random obstacles or other agents to avoid, except for the boundary wall around the room which would reset the episode, spawning the agent and goal randomly again. This random spawner method was modified to spawn the agent and goal in a smaller radius to improve training times.

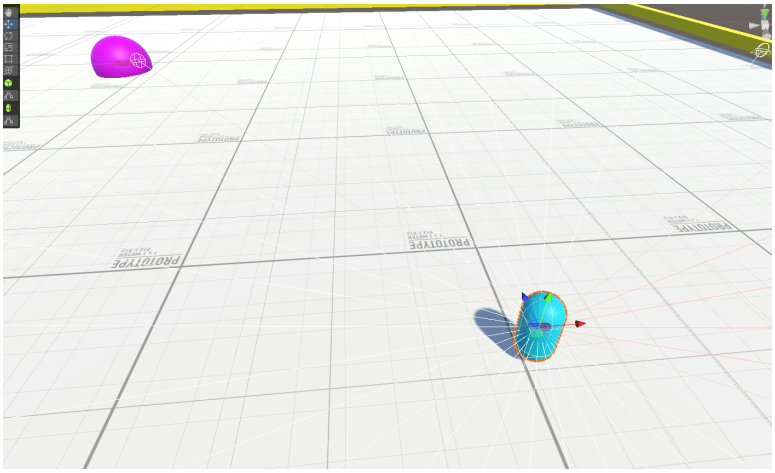

Figure 4.1: Initial single agent testing

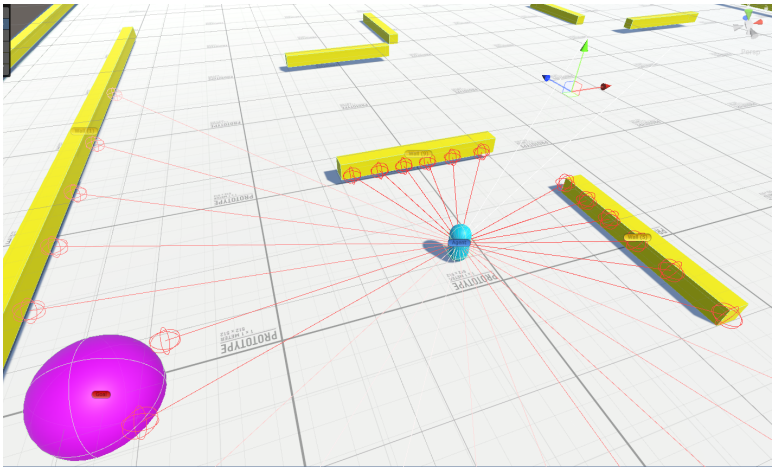

Figure 4.2: Advanced single agent testing

To assist in testing the configuration of the environment, agent and interactions, Unity ML-Agents allows the use of a Heuristic behaviour type. This allows for user input to directly control the actions of the agent(s).

```
1        public override void Heuristic(in ActionBuffers
             actionsOut)
2    {
3            ActionSegment<float> continuousActions = actionsOut.
                 ContinuousActions;
```

```
4           continuousActions[0] = Input.GetAxisRaw("Horizontal");
5           continuousActions[1] = Input.GetAxisRaw("Vertical");
6           }
```

The above was included in the agent core script, which allowed for arrow key input to direct the agent up/down/left/right and included previously mentioned movement smoothing etc, this was useful for testing things like speed but also for testing agent interactions. The OnTriggerEnter(Collider agentSelf) method was used to observe any interaction between the agent and other objects present in the environment, such as the goal or an obstacle.

```
1           if (agentSelf.TryGetComponent(out Goal Goal))
2           {
3               SetReward(5f);
4               floorMesh.material = winMaterial;
5               EndEpisode();
6               RegenerateObstacles();
7           }
```

The above code snippet shows if the agent touches the Goal object, a reward is set, the base material is updated, that episode is ended and the RegenerateObstacles() method is called, which covers the random coordinates for the agent/goal/obstacles present in the environment, and creates them. This was all tested manually via the heuristic behaviour type at first to ensure it all worked as intended. Updating the floor colour helps in visualising how training is going in real-time. Any time an episode failed (e.g. touching a wall/obstacle), it would update to a red material. Green if it reached its goal.

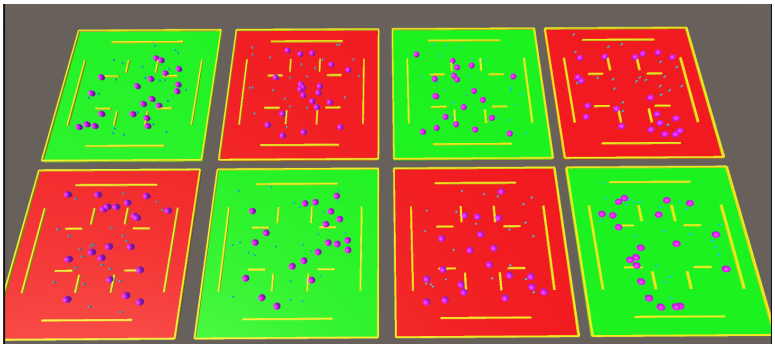

Figure 4.3: Example environments in early training.

This colour change allowed me to observe how training was going with the current parameters and configuration. This in conjunction with the output of mean reward over time and Tensorboard, allowed me to adjust and ensure the single agent was viable. It did work fine with no visual hitches I could note (e.g. erratic behaviour, "unrealistic" movement etc.). Once a training environment is created, it can easily be copied/pasted into 2/4/8/16 instances at once, allowing for increased training speed (assuming the hardware available can handle the load).

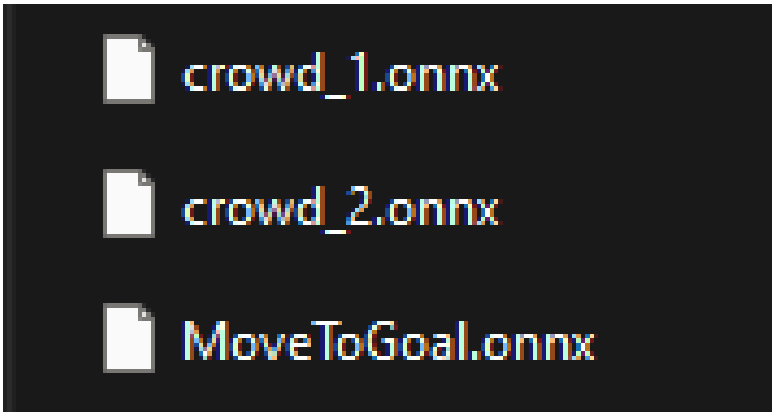

Figure 4.4: Early exported models from testing.

When it comes to tweaking model parameters, Unity Ml-Agents is loaded via the command line, in which your command contains a reference to a .yaml

configuration file, that stores any overridden parameters: "mlagents-learn /Config/MAPOCAagentsConfig.yaml –force". This .yaml file is vital in terms of tweaking parameters appropriate to the goal of the learning.

```yaml
behaviors:
  crowd:
    trainer_type: poca
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: constant
    network_settings:
      normalize: false
      hidden_units: 512
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 1000000
    time_horizon: 1000
    summary_freq: 10000
```

Figure 4.5: MA-POCA config file.

They key hyperparameters that were tweaked for multiple agents are listed below.

- batch_size
  - Due to the potential complexity of environments where more agents and complex building structures and the use of continuous actions, this

value is generally higher than other methods such as PPO. This was relatively low initially (512) and was increased over different training sessions.

- buffer_size
  - I kept this as a multiple of the batch size to allow for more stable training. As this is essentially the replay memory of our agents, it needed to be high to allow for storage of the different scenarios encountered. Due to the fluctuation and randomness of crowds (and the training scenarios), it was eventually increased in tandem with batch size.
- learning_rate
  - This value adjusts the strength for the gradient descent updates. Initially a higher value (.001), it was lowered due to slow changes seen in the rewards initially.
- beta
  - This essentially covers the "randomness" aspect of the policy. This was altered slightly to allow for a more accurate decline in entropy as reward increases.
- epsilon
  - This value represents the divergence threshold of old and new policies. Increased slightly to allow for quicker training turnaround.
- max_steps
  - This value represents the total steps of the simulation/training.

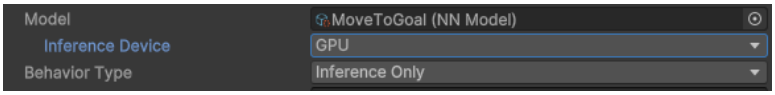

Figure 4.6: Example of loading agent with trained model.

Once you're happy with a single agent, you can begin to add more agents. I set the number of agents and goals per environment to 20. This gave a reasonably busy environment, without overcrowding and causing potential poor or slow training. With this multi-agent environment, we can repeat similar steps when working with a single agent. Train, adjust parameters, repeat. This can be completed with multiple environments or different sizes/layouts to accompany more agents present in a single environment. To account for different scenarios I had kept the same size dimensions of the environments, but altered layouts and number of random obstacles. Some had 0 spawned obstacles, only walls and other agents. Some had more complex room designs, with narrow corridors, intersecions, rooms and others had more simplistic looks but randomly spawning obstacles. This is to account for all the potential scenarios that may arise and indirectly targets the natural changes seen in pedestiran dense areas. Where depending on density, one path can be come entirely unnavigable or other routes may open up. The random obstacles can cause these altering paths as one spawned obstacle could block 1 out of the 4 pathways through an intersection so the entire dynamic of the environment has now shifted slightly, allowing agents to learn.

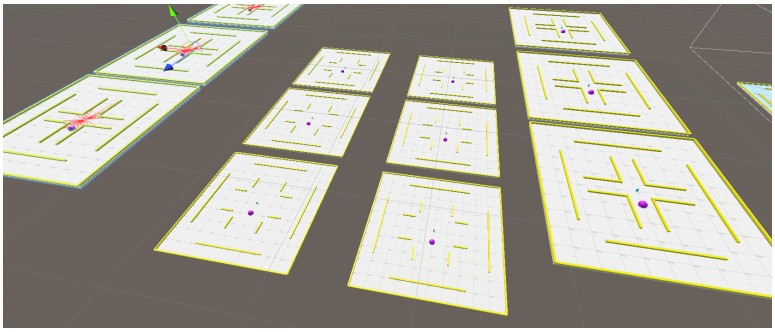

Figure 4.7: Environment setup.

Figure 4.8: Unity Hierachy example for multiple agents.

Once training has been completed, the model is output to a .onnx (Open Neural Network Exchange) file, which is an open-source format that represents machine learning models. [20] The agent can then be updated to load this exported model, once the behaviour type is set to Ingerence only. You can now move the agent into different environments/scenarios and see how the model performs. Removing the EndEpisode() call and agents from the RegenerateObstacles() method allows for new goals to continue to populate the map upon contact trigger, allowing for ambient movement around the environment, without a sudden reset of the agent's position.

Chapter 5

Results

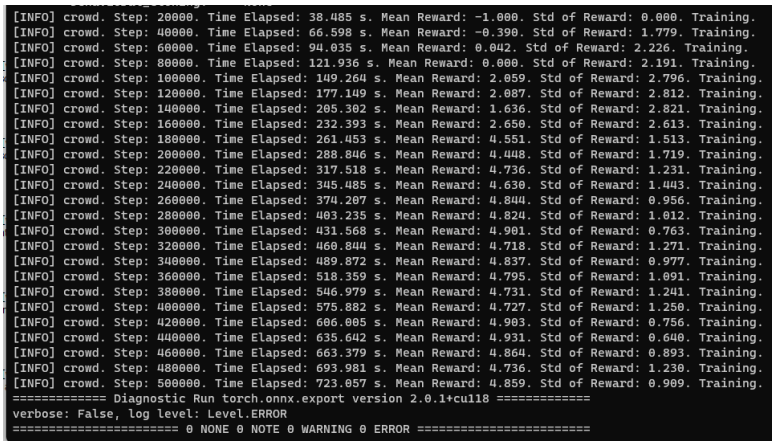

Figure 5.1: Single Agent training reward over time.

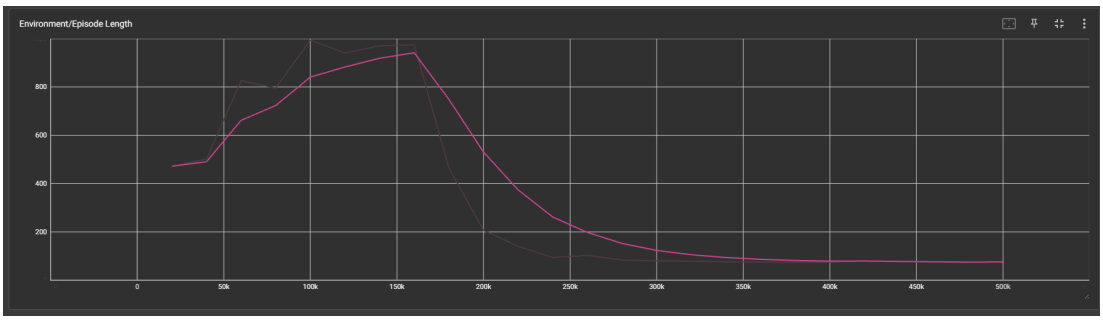

Figure 5.2: Single Agent Tensorboard episode length.

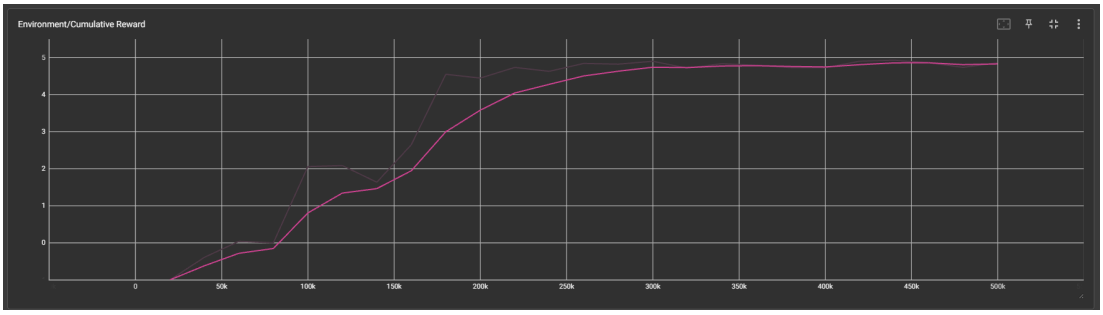

Figure 5.3: Single Agent Tensorboard cumulative reward.

The results shown when using MA-POCA for a single agent were promising. Looking at the reward shown over time 5.1, it takes some time for the agent to adapt to the observations and correct its actions and begin reaching the target. Over time we can see an increase in both mean reward and the time taken to complete the checkpoint intervals (20,000 steps). This is more easily observed in the two Tensorboard graphs 5.2 and 5.3, where both episode length begins to drop around the 150k mark and cumulative reward rises. The mean reward is close to its peak (roughly 4.5) after only 200k steps, but the standard deviation of this reward is still rather high (1.7). For the rest of the steps we don't see the score move much higher, but the standard deviation comes down showing a more consistent reward across the instances of single-agent training.

In references to RQ1 we can see by the above figures that we are completing the intended goal, but calling it a recreation of human behaviour is a bit more difficult to judge. As this is partially judged based on observation I have to watch for certain agent behaviour after training by loading the exported ONNX file and visually inspecting agent interactions. When an agent is using a pre-trained model, I remove the episode resets and random spawning for only the agent. This allows the goals to spawn around an environment and I can watch for the agent's reaction.

- Agent's acceleration/deceleration.

- Agent's turning behaviour.
- Agent's reaction to nearby obstacles.
- Agent's navigation around walls.

Of the above, no behaviour or reactions were "unrealistic". Any interaction looked to represent a solid reflection of human behaviour when walking from objective to objective, with no sharp turns, speed-ups, slowdowns or abrupt behaviour. This was good enough to proceed into RQ2.

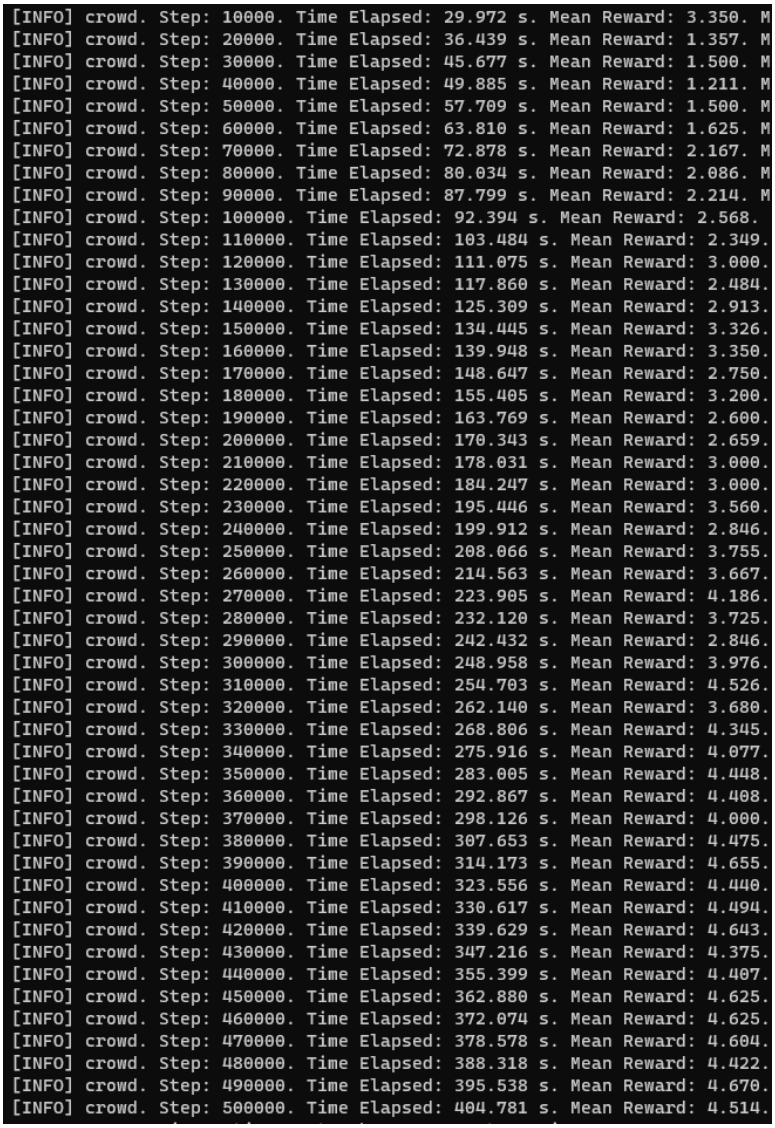

Figure 5.4: Console output of reward over time.

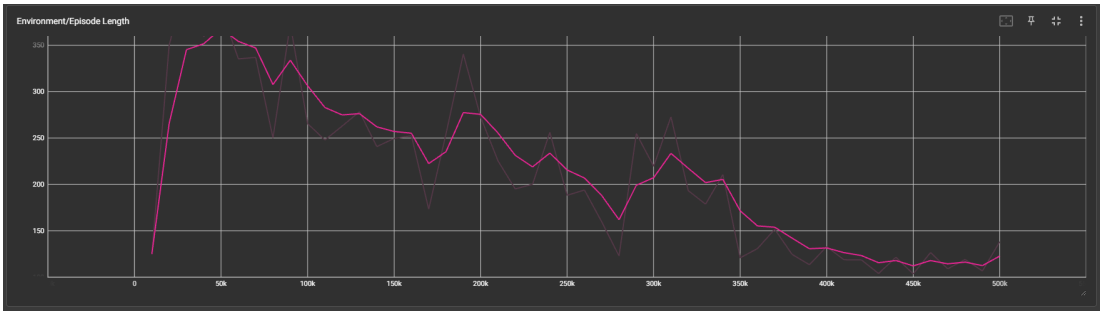

Figure 5.5: Multiple Agents Tensorboard episode length.

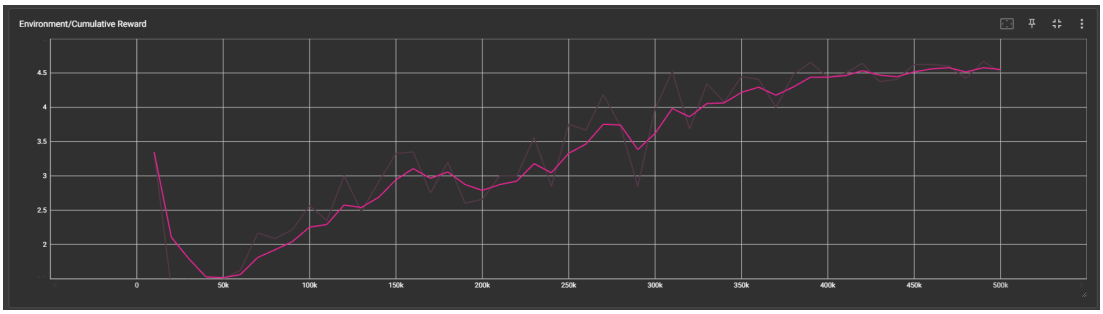

Figure 5.6: Multiple Agents Tensorboard cumulative reward.

For the multi-agent reinforcement, we could expand the environment scale and include multiple agent/goal pairs. Initial results were rather poor and slow, but after hyperparameter tweaking things began to reflect the results seen in the single-agent training. 5.4 shows us a similar result, in which as steps are completed the mean reward creeps upwards and the time to complete an episode begins to fall off. Layering the two graphs 5.5 and 5.6 one above the other results in a candlestick holder/vase shape where it is wide at the start and begins to narrow. This is a great result to see as it shows the agent is picking up correct behaviours that allow them to reach optimal rewards, fast.

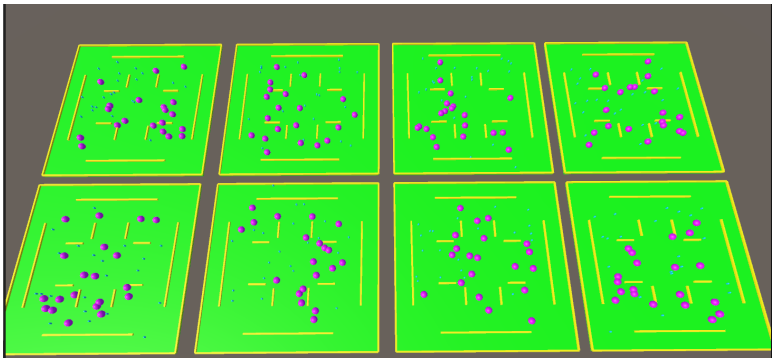

Figure 5.7: Agent output ONNX in use with 20 agent scenarios.

Again, the numerical results don't account for the core of the RQs. They just show us that reinforcement learning is working, but not if the agents are producing optimal behaviour or human-like behaviour. 5.7 shows a set-up of 8 environments in which the pre-trained model for multi-agent MA-POCA was loaded. I kept the colour-changing base to reflect collisions or reach the goals. Again, looking for the same behaviours mentioned prior, but with the addition of multiple agents present in the environment. Results looked very promising with few collisions and a nice feel looking at it from a macroscopic point of view. I think this resonates well with RQ2, in which an RL approach for multiple agent scenarios and crowds is effective and can be implemented when tuned and trained correctly.

To test for finer behaviours, I created several test scenarios to purposely check agent reactions to potentially awkward situations. This is to benchmark reactions between the two implementations on a microscopic level.

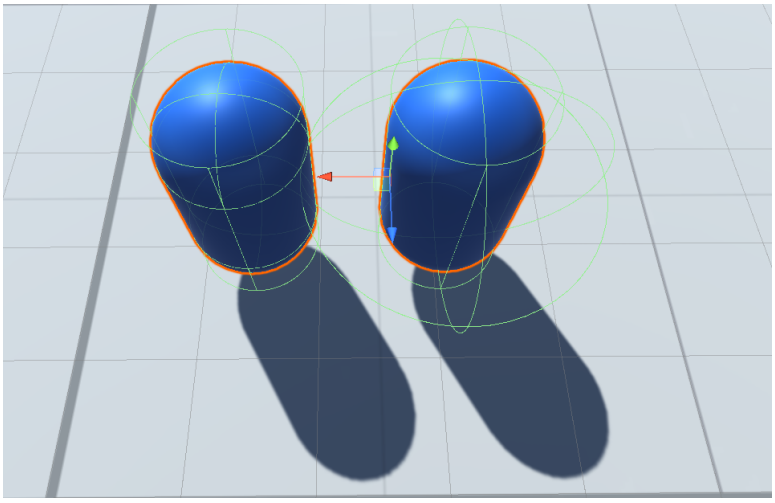

Figure 5.8: Agent collision boundry(Left) Agent counter boundry(Right).

I needed a variable to judge the accuracy of agents while also observing their behaviour and movement. A collision counter was produced that would count once-off collisions. E.g if an agent touches an agent and stays within a certain radius, no further collision would be counted until the agent's collision boundary left the other agent's counter boundary, as seen in 5.8. This allowed for accurate collision counting in different scenarios. To get a general sense of the agent's accuracy in navigating the environment and avoiding obstacles/agents. The pre-trained model was loaded and a 30-agent environment was created. A mirrored environment was created alongside it, but using the classic NavMesh agents. The agents were then spawned at random once, from there the goals would spawn randomly and would spawn in a new random location after contact with their corresponding agent. This would proceed over a 5-minute time period, in which the agents and goals would despawn. Collisions were counted per engine tick and totals were logged every 30 seconds. This will show a rough average of how well the collision avoidance works in comparison to one another.

The table 5.1 shows these collisions over the 30-second time intervals. On average MA-POCA produces 3 fewer collisions per 30-second intervals or 24.8%.

Time	NavMesh	MA-POCA	Difference	%Difference
00:30	5	3	2	40.0
01:00	11	8	3	37.5
01:30	14	14	0	0.0
02:00	20	16	4	25.0
02:30	24	21	3	14.3
03:00	29	24	5	20.8
03:30	33	29	4	13.8
04:00	37	32	5	15.6
04:30	41	37	4	10.8
05:00	46	42	4	9.5
		Averages=	3.00	24.8

Table 5.1: Collision comparative over time.

This was repeated to ensure the randomness was not affecting results, but the percentage difference was relatively equal each time. We can potentially see why below.

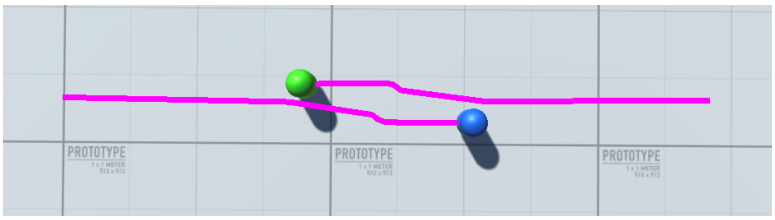

Figure 5.9: 2 Non RL agents straight line test.

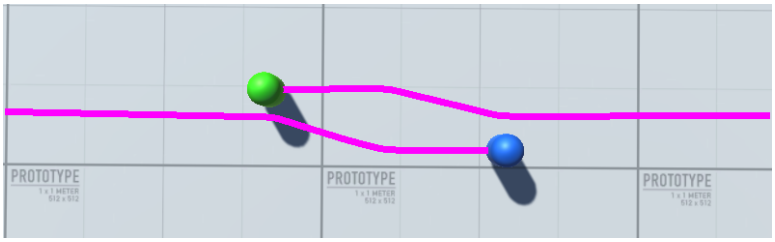

Figure 5.10: 2 RL agents straight line test.

A series of tests were performed to show the comparative performance of agents in certain scenarios, such as coming head to head as seen in figures 5.9

and 5.10. Both the pair of NavAgents (RVO+A*) and RL agents were given a particle trail to show the finer movements made when in collision trajectory with one another. The NavAgents agents 5.9 trails show a sharp adjustment made once the two agents come too close. This is not a realistic representation of avoidance, e.g if you've ever been walking down a footpath and come across another pedestrial, you'd both make slight adjustments to your direction and gracefully pass eachother. The NavAgents don't do this until the algorithm sees an opportune moment to do so, resulting in a bizzare interaction where both agents turn sharply before collision. The RL agents 5.10 have a much better representation of what you would expect a human interaction like this to do. The adjustment to their direction is much finer and happens earlier, preventing any sharp turns or collisions while keeping a more human look to the interaction.

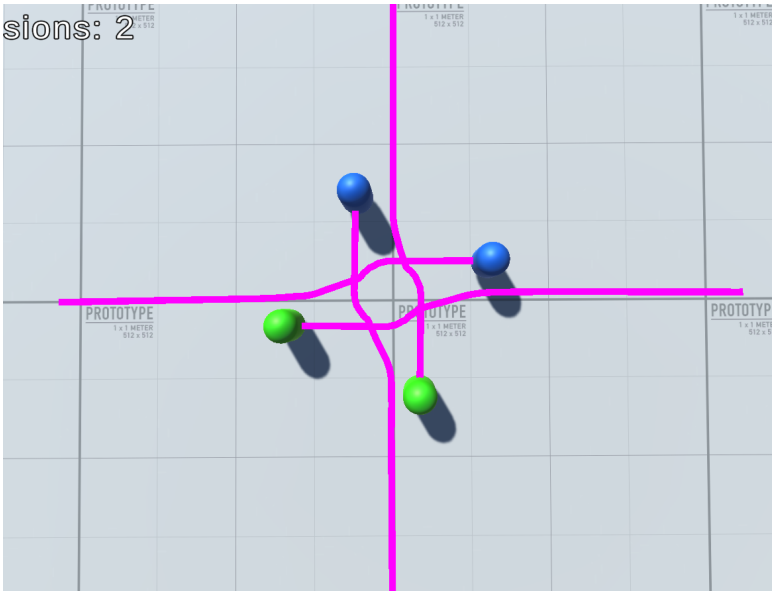

Figure 5.11: 4 Non RL agents straight line test.

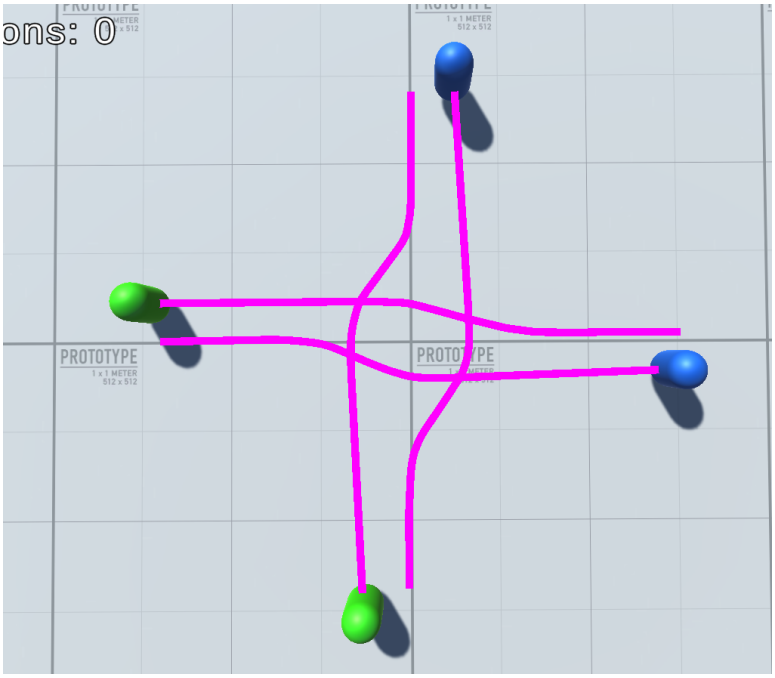

Figure 5.12: 4 RL agents straight line test.

To expand on this and add some more complexity, 4 agents were used in a four-way intersection. The NavAgents in figure 5.11 show similar behaviours to the head-on collision test, where rather robotic/sharp adjustments are made in the path. Resulting in strange-looking avoidance, including 2 collisions as seen in the top left of figure 5.12. This is due to RVO not being able to appropriately deal with oncoming collisions quick enough, leaving too little space for adjustment. The RL agents again show a more realistic result, with earlier, smoother adjustments made to their velocity, with the added benefit of 0 collisions counted. The shape is similat to the NavAgent, but you can see the agents moving up and down give a wider radius, this is mutually beneficial to all agents as they continue at a constant speed and avoid collisions. This may not be a genuine thing that we see happen in the real world, but it does represent the potential advantages of the RL approach on finer movements.

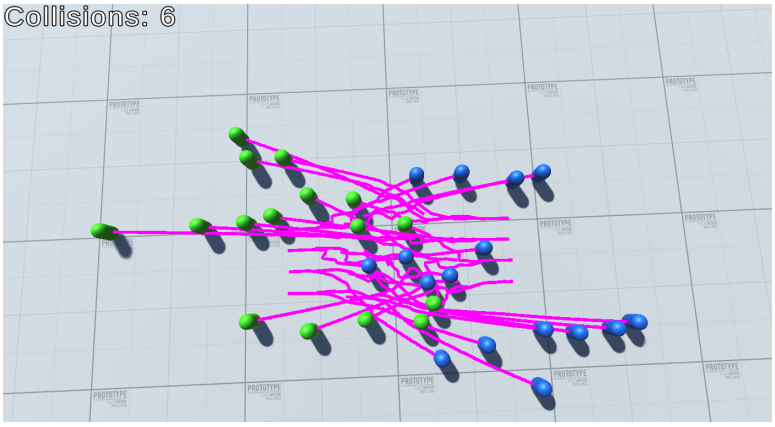

Figure 5.13: Multiple Non RL agents straight line test.

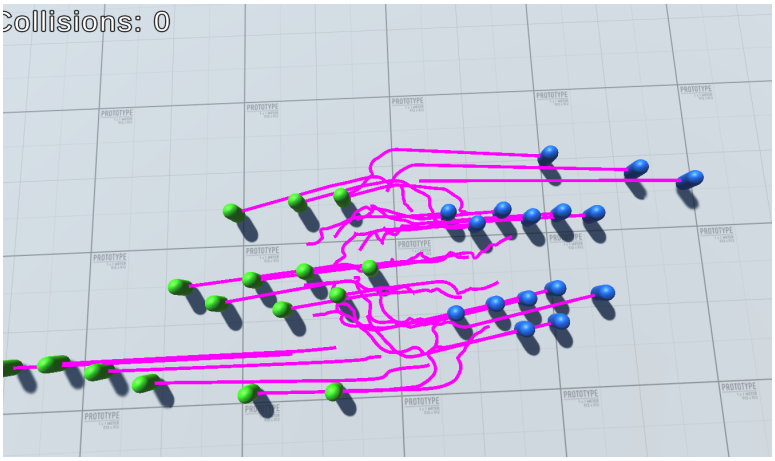

Figure 5.14: Multiple RL agents straight line test.

The head-on collision test was scaled up to 32 agents, 16 on either side in a 4:4:4:4 layout. This visually is a bit more complex, but there are some distinguishing differences. The RL agents 5.14 form lanes when passing through one another, allowing for collision-free movement, which may represent the more realistic result. These lanes that open up allow for all agents to navigate collision free and quicker. The NavAgents 5.13 initially collide and push past one another, creating openings for the agents behind to pass through. This produced 6 collisions total and a less realistic observed result. Time is lost initially due to

the centre 2 agents RVO not seeing any way out besides pushing past oncoming agents, this is the same with the opposing side. This can be observed in the pink trails left by the agents.

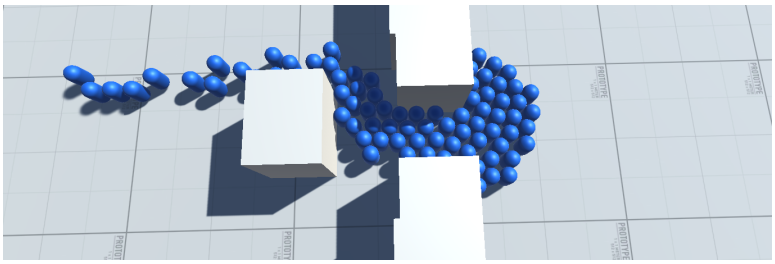

Figure 5.15: Non RL Agent bottleneck: 24.63 seconds to clear.

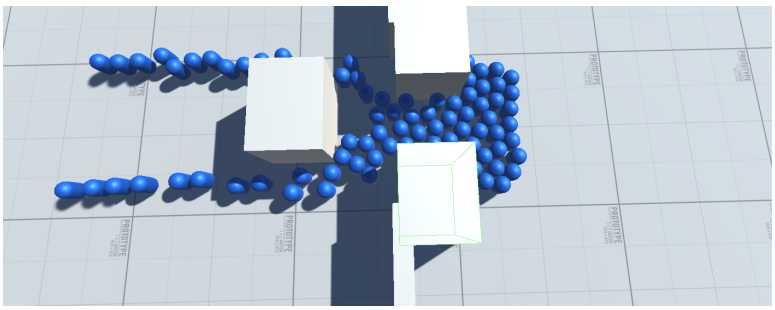

Figure 5.16: RL Agent bottleneck: 20.21 seconds to clear.

The final comparative test was a bottleneck situation, such as exiting large venues such as arenas and concerts or in evacuation scenarios such as fire escapes. This was timed as well as observed, collisions were voided as there were expected collisions on either side. The NavAgents leaned into the A* algorithm and all went for what was observed as their optimal route, rather than what was the most realistic way out. All agents waited to move up and around the obstacle when exiting. In contrast to this, the RL agents 5.17 split around the obstacle. I believe this is inherited from the group policy in use by MA-POCA, as it resulted in the fastest way to the goal.

The timings between the two were also quite different. 24.63 seconds for the

NavAgents, and 20.21 seconds for the RL agents. This is almost a 22 percent decrease in time taken to exit the bottleneck and pass the obstacle. This also happens to be rather close to the average percentage difference in collisions observed in the prior timed collision test 5.1. The NavAgent approach produces in many "pushing" instances, where agents cram through the doorway and shove one another, this is reciprocated causing the slow down and is highlted at corners such as the entrance and exit to the doorway where some agents will become stuck as other agents shove passed them. This can be interpreted in two ways; One in which it is a genuine emergency situation and agents behave in a greedy manner, and another in which agents are exiting a concert for example. The RL agents are much more orderly, where as the NavAgents are more hectic as they are programmed to get to their destination. The Collision avoidance is only trying to update trajectory to avoid collisions, but in situations like this dense bottleneck that cannot really calculate correctly and so they push to get to the objective. The RL agents have learned that shoving is not an ideal action and so this does not really happen. At times it is unavoidable, such as being in the middle of the bottleneck, they cannot slow or turn to avoid a collision so it is just a part of the scenario. No strange behaviour is noted by RL agents, such as trickling through the doorway.

If we look at RQ3, the main question it raises is if an RL solution to crowd simulation is worth it. Using the combination of all the results I believe it is fully worth using for more delicate or detailed scenarios. A combination of A* and RVO is great for less detail orientated simulations, such as massive crowds (10,000+ agents) in the background of some historic war film where scale is the main goal. It is much more performant on an engine level, especially when agents peak into the 1000's due to the simplicity of the algorithms. A test was run for 16,364 agents (1024*16) on a large open space, with one goal to move to for all

agents. The NavAgent performance was great as per 5.17. When running this with the RL agents, CPU timings averaged 1900ms compared to the NavAgent's 50ms and FPS was round .5 frames a second compared to the NavAgent's 20fps. None of the implamentation was created with performance in mind, but this gives a rough idea on how it transaltes.

Figure 5.17: NavAgent Unity profile snippet.

If detail and realism is key over realtime performance, I belive the MA-POCA implamentation has shown it can imrpove on standard algorithmic approaches, giving acceptable levels of human behaviour replication. I am sure with design for performance in mind, the performance can be impoved but with current states available to the agent, there is too much information available and this cascades into too much complexity for what they trained for.

Chapter 6

Conclusion

The point of this project was to venture into the use case for reinforcement learning in crowd simulation, for work in dynamic scenarios such as in entertainment sectors like video games, TV and movies or in real-world simulation for use in building planning for traffic flows of pedestrians or in fire escape scenario planning. By taking a staggered approach of a single RL agent and expanding on the agent count, these results have shown that it is indeed feasible for RL agents to provide a realistic-looking simulation of agents, accounting for both microscopic and macroscopic crowds with the use of MA-POCA. Smoother intersection vectors, fewer collisions and an overall natural-looking simulation was produced. RQ1 and RQ2 were more than answered for during the simulation and with regards to RQ3, I would have to say it is indeed worth using an RL solution for agent simulation in crowds. The RL approach is more work to implement due to the rigorous hyperparameter tuning, careful training environment set up to account for scenarios required and general resource consumption required for repeated training. The gains we see from this, however, are not superficial. There are genuine statistical improvements, as well as visual improvements observed which in scenarios such as emergency simulation, is vital. The more accuracy we

see in human simulation in these types of scenarios the better. As these types of simulations can be the closest virtual mirror we have to potential disasters, the more realistic, the better-prepared things can be in the real world. MA-POCA is a new algorithm. where first publishings were only from 2021 [19]. Since then I have not found any research papers or studies using it for crowd simulation and I believe this paper outlines the potential usefulness of this algorithm in recreating human behaviour and cooperation. This could hopefully guide further studies and deeper analysis into the algorithm's applications in the simulation field. In saying this, some limitations are inherent in most RL approaches. To successfully implement MA-POCA for crowd simulation, I had to account for the basic actions and states required to give a basic recreation of human behaviour, but RL can only work with what is provided to it. Humans are much more complex than this and the simulation does not account for outliers, such as more selfish human actions. E.g not moving out of the way for an oncoming person or in emergency situations, fight or flight can kick in and humans would become more concerned with their actions and less so for those of strangers around them. This is not caught in the MA-POCA algorithm as there is a general sense of one for all via the group reward. I cannot say it gives a genuine one-to-one recreation of what human crowds may do, but it does provide a sufficient recreation for general simulation use. Catching these abstract behaviours is very difficult to achieve in any RL algorithm so further research would have to be conducted here, even looking to hybrid approaches to merge different algorithms or machine learning techniques to try to encapsulate as much of these human "quirks" as possible. The environment types here are also incredibly important to what the agent learns. If I had used a single environment for training, and attached this output ONNX file to new agents in a brand new environment, it would all quickly fall apart and this is a fairly major limitation in my opinion. If I could work on this further

and expand, one of my goals would be to allow agents to adapt to new, unseen surroundings better. E.g a user would have access to environment-building tools to recreate a space they wished to simulate, or import sketch-up files for buildings already designed and then let the agents run wild. I believe I did a good enough job in terms of varying the training environments and the inclusion of randomness allows the agents to adapt to new situations in each episode, but I think it can be vastly improved upon. Another limitation is the reliance on human observation to dictate if the simulation is realistic or not. There is numerical data produced but that indicates accuracy in collision detection more than anything. A genuine inspection of whether the agents are producing realistic navigation through a busy virtual world is too reliant on sitting and observing scenarios. With more time I'd have liked to implement something to analyze this automatically. I toyed with heatmaps for density, pinpoints of collisions and flowlanes but this did not produce great results and was scrapped from the project. This could be built upon and even the use of other machine learning algorithms purely for analysing recorded results may help in this process. Another point is performance, while it was not the aim or really a major factor of this paper, with further research and time I believe it can be vastly improved upon to allow for much larger simulations as currently you would be limited to a few hundred agents in an environment before performance begins to fall off and the result is too resource heavy to be usefull.

Overall I believe crowd simulation is a highly interesting, complex landscape with many routes to tackling it. Reinforcement learning is still rapidly and evolving and the use cases for recreating human behaviour will continue to grow and encapsulate it. Humans are arguably the most complex organism on this planet and as different technology fields expand and mature, we can inch towards recreating human behaviours and work towards creating a safer, more responsive world

for our ever-growing population. I believe this project was a drop in the ocean of understanding complex human idiosyncrasies, but still a step in the right direction.

References

[1] R. Narain, A. Golas, S. Curtis, and M. C. Lin, "Aggregate dynamics for dense crowd simulation," vol. 28, no. 5, pp. 1–8, publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/1618452.1618468 ii

[2] "Unity ML-agents toolkit," original-date: 2017-09-08T21:09:04Z. [Online]. Available: https://github.com/Unity-Technologies/ml-agents v, 10

[3] M. Shahrizal, D. Daut, B. A. A. M. A. B. Mohamed, and Sunar, "Survey on real-time crowds simulation," in *Technologies for E-Learning and Digital Entertainment*, Xiaopeng, E. R. Abdennour, W. Woontack, L. Y. P. Zhigeng, and Zhang, Eds. Springer Berlin Heidelberg, pp. 573–580. 1

[4] J. Lee, J. Won, and J. Lee, "Crowd simulation by deep reinforcement learning," in *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3274247.3274510 1, 7

[5] S. Yang, T. Li, X. Gong, B. Peng, and J. Hu, "A review on crowd simulation and modeling," vol. 111, p. 101081. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1524070320300242 2

[6] M. Haghani and M. Sarvi, "Human exit choice in crowded built environ-

ments: Investigating underlying behavioural differences between normal egress and emergency evacuations," vol. 85, pp. 1–9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0379711216300844 2

[7] D. Thalmann, *Crowd Simulation*. John Wiley & Sons, Ltd, publication Title: Wiley Encyclopedia of Computer Science and Engineering. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470050118.ecse676 5

[8] D. M. S. R and Thalmann, "A model of human crowd behavior : Group inter-relationship and collision detection analysis," in *Computer Animation and Simulation '97*, M. T. Daniel and v. d. Panne, Eds. Springer Vienna, pp. 39–51. 5

[9] H. Shteingart and Y. Loewenstein, "Reinforcement learning and human behavior," vol. 25, pp. 93–98. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959438813002286 5, 7

[10] S. R. Musse and D. Thalmann, "Hierarchical model for real time simulation of virtual human crowds," vol. 7, no. 2, pp. 152–164. 5

[11] Q. Wang, H. Liu, K. Gao, and L. Zhang, "Improved multi-agent reinforcement learning for path planning-based crowd simulation," vol. 7, pp. 73 841–73 855. 6

[12] K. Hu, M. B. Haworth, G. Berseth, V. Pavlovic, P. Faloutsos, and M. Kapadia, "Heterogeneous crowd simulation using parametric reinforcement learning," p. 1. 6

[13] A. Braun, S. R. Musse, L. P. L. d. Oliveira, and B. E. J. Bodmann, "Modeling individual behaviors in crowd simulation," in *Proceedings 11th IEEE International Workshop on Program Comprehension*, pp. 143–148. 6

[14] D. Thalmann, H. Grillon, J. Maim, and B. Yersin, "Challenges in crowd simulation," in *2009 International Conference on CyberWorlds*, pp. 1–12. 6, 7

[15] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *ICML*. 7

[16] P. K. Sharma, E. G. Zaroukian, R. Fernandez, A. Basak, and D. E. Asher, "Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, T. Pham, L. Solomon, and M. E. Hohil, Eds. SPIE, p. 84. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11746/2585808/Survey-of-recent-multi-agent-reinforcement-learning-algorithms-utilizing-centralized/10.1117/12.2585808.full 11

[17] D. Thalmann and S. R. Musse, *Crowd Simulation*. Springer. [Online]. Available: https://link.springer.com/10.1007/978-1-4471-4450-2 11

[18] O. J. Urizar, E. I. Barakova, C. S. Regazzoni, and M. Rauterberg, "Modeling crowds as single-minded entities," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1186–1191. 11

[19] A. Cohen, E. Teng, V.-P. Berges, R.-P. Dong, H. Henry, M. Mattar, A. Zook, and S. Ganguly, "On the use and misuse of absorbing states in multi-agent reinforcement learning." 11, 43

[20] "Use ONNX," original-date: 2017-09-07T04:53:45Z. [Online]. Available: https://github.com/onnx/onnx 27

Chapter 7

Tools and links

- Code Repo (Note, most of the work completed inside Unity project): https://github.com/caolanmc/RLAgentCrowdSim
- Unity: https://unity.com/
- Unity ML-Agents: https://github.com/Unity-Technologies/ml-agents
- Base grid texture: https://assetstore.unity.com/packages/2d/textures-materials/gridbox-prototype-materials-129127