Multi-Agent Surveillance using Limited Resources

Aditi Mishra, Dumitru Verşebeniuc, Jack Waterman, Laurence Nickel, Noah Scharrenberg, Tjardo Neis
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*— Multi-agent surveillance games are common practice in both nature and the digital world, such as lions chasing antelopes (nature), bank security stopping robbers (nature and digital) or companies trying to stop hackers (digital). This paper addresses algorithms for two game modes, where the first game mode defines "exploration" and the second game mode defines "pursuit-evasion".**

In exploration two approaches are introduced, namely frontier-based (Yamauchi) and stack-based-orthogonal (SBO), while pursuit-evasion takes much more complex approaches while also utilizing exploration when necessary. Pursuit-evasion introduces a reinforcement learning agent (RL), acting as a guard, deep-q-learning agent (DQN), acting as an intruder, and a probabilistic-rule-based agent (PRB), acting as both guard and intruder. Constraints are introduced for the agents, as direct communication between the agents is not possible. However, indirect communication is possible through the usage of marker placements and yelling.

Experiments conducted on different parametric-values have shown that markers increase the amount of times that intruders are able to escape, while an increased vision allows the guards to capture more intruders. For which the DQN Agent wasn't able to perform more than a one versus one, caused by time constraints.

***Index Terms*—multi-agent surveillance, pursuit-evasion, reinforcement learning, deep Q network, yamauchi, frontier, stack-based orthogonal, probabilistic, guards, intruders, agents**

I. Introduction

Automated surveillance of different environments through cooperating mobile agents (robots) is among the most urgent and promising topics in multi-robot research. Since the past decade, automated surveillance systems have been used as a mean to improve security and safety. In recent years, there has been an increased interest in using teams of mobile agents for automatically surveilling environments of different type and complexity.

This paper presents an approach to implementing a multi-agent surveillance game. The game consists of strategically designed algorithms for both the guards and the intruders. The goal of the intruder is to reach a target area, while the goal of the guard is to capture the intruder before this happens.

Previous research on Multi-agent surveillance games involved an openAI Hide and Seek game which indicates the state-of-the-art for the game described in this article. The difference being that the hiders (intruders) in the game hide whereas the similarity is that the seekers (guards) need to find the intruders. One of the big challenges experienced during this research was that by using standard reinforcement learning algorithms, agents learned to build multi-object shelters using movable boxes which in turn led to agents discovering that they could overcome obstacles using ramps. Modifications to the game were made to overcome this issue.

Another research was done which gave an insight on exploration algorithms of multi-robot systems. One of the big challenges in this research was to complete a task simultaneously in a shorter time than a single robot. The algorithms applied in this research, that are in-line with the approaches in this article are: Frontier based exploration and Rule based exploration. This research used these common algorithms to perform the exploration process of an unknown cell-based environment.

The main research question that this article focuses on is: *How to surveil or intrude an area with limited knowledge and resources using intelligence?*

To answer this question, several other research questions were formulated such as:

1) How to explore an unknown area?
2) What impact does communication have on agents?
3) How to catch an intruder as a guard?
4) How to reach the target as an intruder?
5) How could collision between teammates be avoided?
6) What is the difference in performance of the agents with the use of obstacles (shadows, teleporter, etc)?

This article will briefly explain the implementation and design specifics of the game (chapter II), the methodologies of all the different agents (chapter III), implementation of all the agents (chapter IV), experiments (chapter V), results of the experiments performed (chapter VI), discussion of these results (chapter VII) and conclusion of the entire article (chapter VIII).

II. Multi-agent Surveillance

Before getting into the specifics of the algorithms, it's important to have an understanding of the core mechanics and rules of the game. This section describes the core game principles of the multi-agent surveillance game and lists the nuances of the version used in this paper.

The multi-agent surveillance game is an autonomous pursuer-evader game of artificial agents playing either a guard or intruder (explained in II-A), while having limited knowledge and resources (explained in II-C and II-D)

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University. Supervisor(s): E.A.H Ghaleb and K. Schneider

A. Game types

These objectives may vary depending on the defined game type. The game allows for three game types to be played, namely:

- Exploration
- All intruders Escaped
- All intruders Captured

1) Exploration: Exploration is a game type that is only played by the guard. It has to explore all accessible and viewable tiles within the shortest possible amount of time-steps.

2) Guards vs Intruders: Guards vs Intruders is a game type where the guards have to capture the intruders and where the intruders have to escape by reaching the target area and staying on it for X amount of time-steps (explained in II-B).

This game type is to be defined as either "All Intruders Escaped" or "All Intruders Captured".

a) All Evaders Escaped: This game type utilizes the Guards vs Intruders principle with the condition that all intruders must escape for them to win the game.

If the guards manage to capture one intruder, then they will win the game.

b) All Evaders Captured: This game type utilizes the Guards vs Intruders principle with the condition that all intruders must be captured by the guards for them to win the game.

If one intruder manages to escape, then they will win the game.

B. Target

The target area is the area of the game that the intruders have to reach in order to potentially win the game.

Intruders are being given indicative directions to the target area to help them receive information on the general direction of the target area.

C. Knowledge

The environment is completely unknown to the agents. They may only know what certain objects pertain, but not necessarily it's meaning.

The agents need to gather the knowledge throughout the game, meaning they have a lot of uncertainty to uncover while exploring, searching, chasing or fleeing.

D. Resources and Communication

In addition to being completely unknown to the environment, agents can only indirectly communicate with each other. Agents can not use communication such as talking, writing, texting, ..., instead they are provided with markers and the ability to create noise (movement and yelling).

Agents also cannot see infinitely long, but are limited in what they are able to see, and do not know everything going on around them, but only what it sees and the sounds it perceives.

1) Noise Generation: When making movements, agents create noise which can be heard within a certain radius by all other agents. The strength of the noise depends on the type of movement made by the agent.

- Standing Still - Generates (almost) no noise
- Rotating - Generates a small amount of noise
- Walking - Generates an audible amount of noise
- Running - Generates a very audible amount of noise
- Yelling - Generates an extremely audible amount of noise

Within these possible noises only yelling can be used as a form of communication, while the other noises made by the agents are done by moving around and cannot be influenced by the agent itself.

2) Vision: All agents are equipped with a cone-based vision. This limits their ability to know what is happening around them, and makes it rely not only on its vision, but also on its hearing.

3) Markers: Agents should be able to use some tools to communicate with each other. Therefore markers are provided, which can be used by the agents to indicate a certain meaning, and provide potentially vital information to the other agents.

There are 6 markers in total, 5 for guards and 4 for intruders of which 2 unique to their respective role. Namely:

- Guard spotted
- Intruder spotted
- Dead-end found
- Teleporter Found
- Target area found
- Shadow spotted

E. Map Building and Map Components

Maps can be made using in two way:

- Manual generation through a text file
- Visual generation through the Tiled map editor (external program) [1]

Where maps can contain components such as walls, teleporters and shaded area's.

A more detailed explanation of the map builders and the map components is given in Appendix A.

A graphical representation of the map can be seen in figure 1.

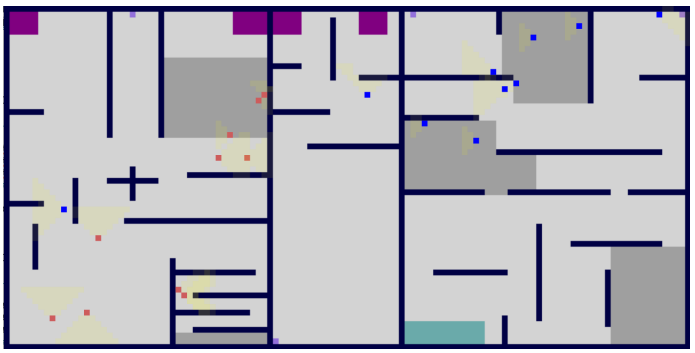


Fig. 1: Example Map Representation

F. Limitations

Listed below are some nuances made to the game:

- No sprinting capabilities
- Cone-vision instead of rectangular vision
- Markers remain for X amount of time-steps.
- Intruder must be on the target for X amount of time-steps, and not seconds.

Nuances regarding to map properties are mentioned in Appendix A-B2a.

III. Methodology

This section covers the exploration and Guards vs Intruder Algorithms used throughout this project.

A. Exploration

1) Random Agent: The random exploration agent functions fairly straight-forward, as it takes either one of four moves: up, down, left or right as its action.

The random agent only has one "non-random" element in its logic, which is to navigate a different direction when it's hitting a collision (either left or right).

2) Yamauchi Frontier Agent: The Yamauchi Frontier Agent is based on the Frontier algorithm developed by Brian Yamauchi. [2]

Tiles are distinguished as open, unknown and occupied spaces. Open Spaces are known spaces that can be visited, while occupied spaces are known but inaccessible and unknown spaces are simply not known to the agent.

The main idea about frontier-based exploration is to gain the most information about the world, by moving to boundary areas in the known space of the agent. Frontiers can therefore be defined as regions on the boundary between the open known space and unexplored space. [2]

The (possible) navigation to the frontier(s) is done through path-finding algorithms, this paper handles both breadth-first search [3] and A* search [4] to reach the frontiers. These algorithms only take open spaces into account, and do not consider occupied or unknown spaces.

In Figure 2 a simple scenario is given, where the agent knows and sees that it is encapsulated by two walls with three open boundary tiles, to which the marked tile is the best possible frontier to take (assuming a maximum frontier size of one).

Generally, the size of the frontier should be bigger than one, with a centroid defined as the space to utilize. In most frontier-based algorithms the centroid is defined as the cartesian average point of the frontier [5]. However, a modification has been done to try and find a more fitting frontier with the disadvantage of taking more computation time, in which we take the shortest Manhattan distance and open space with the most information gain as the centroid of the frontier. Once all possible frontiers have been detected, path-finding using A* search and the same information gain is used to detect the best frontier based on these evaluation criteria. [2]

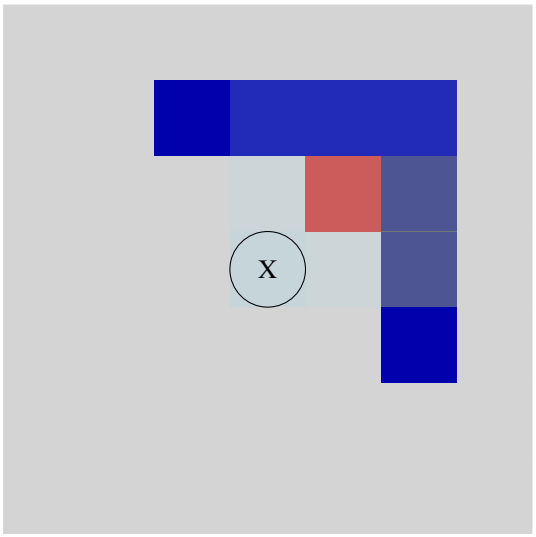


Fig. 2: Possible Frontier Detection

3) Stack-based Orthogonal Agent: The Stack-Based Orthogonal (SBO) agent is a deterministic explorer agent. The core concept is build around the principle of utilizing stacks to keep track of all the tiles the agent wants to explore. Tiles are continuously added to the stack and removed from the stack based on the vision input it is receiving.

The tiles within the perimeter of the agents vision are stored and examined, before it is added to the stack, certain conditions must be satisfied:

- The tile has been explored
- The tile is open (not occupied)

Every timestep, the tile on the top of the stack is retrieved for its current goal. A* search is then used to determine a path from its current position to the temporarily defined goal. Once this temporarily defined goal is in the agents vision, it will be removed from the stack and a new tile will be selected as its new temporarily defined goal.

This follows a similar approach to frontier-based algorithms with a frontier size of one. However, going through the frontiers in chronological order, this takes fewer computations than the Yamauchi Frontier, and would therefore be faster although less efficient in its pathing than the Yamauchi Frontier.

B. Guards vs Intruders

1) Probabilistic Agent: The Probabilistic Agent is loosely based on an existing approach for pursuit-evasion control strategies under uncertainties [6].

a) Guard Control Strategy: The guard utilizes three types of possible modes it can be in:

- Sweeping
- Investigate
- Chasing

The default mode of the guard is to be sweeping through the map in an attempt to find the intruders while also getting to know the layout of the map.

It does this through the use of the Yamauchi Frontier algorithm.

If the guard receives any external sensory input (except visual input), it will investigate said sensory input to find out whether or not it is an intruder.

If the guard has an intruder in its vision, it will start to chase the intruder and give out a yell for other nearby guards to know that an intruder is spotted, these guards will then investigate and chase the intruder.

b) Intruder Control Strategy: The intruder utilizes two types of possible modes it can be in:

- Search Target
- Flee

The default mode of the intruder is to be search through the map for the target.

It does this through the use of the Yamauchi Frontier algorithm. However, the intruder uses the indicative target direction to favour frontiers going towards the target.

If the intruder receives any sensory input (including visual input), it will flee from it, until it knows it is safe (sensory is confirmed to be from another intruder).

c) Adjustments from original algorithm: The main differences between the algorithm used in this project and the algorithm on which it is based on, is that no probability of the opponent team is necessarily considered throughout the decision making process. Which gives the main disadvantage that the algorithm can not plan ahead passed the known information and consider the potential direction the opponent went to when all sensory input has been lost.

2) Reinforcement Learning Agent: The exploration component of this agent uses the Yamauchi Frontier algorithm. Path finding from its current position towards other tiles is done by using the A* algorithm. All other environmental information goes through the reinforcement learning model.

a) Learning model: The model used for the reinforcement learning agent contains 3 main elements, assessment, evaluation and feedback [7]. The assessment phase converts all input into parameters for the evaluation. The input consists of all data that the agent can perceive, through vision, audio and smell. After all of the input is analysed. It starts evaluating each parameter, based on its typing, strength, direction and priority. These evaluations are then compared and the most prominent value is converted into an action sequence for the agent to perform. These 2 parts are performed at every time step during the game. At the end of each game, the performance of the agents is analysed and this is what creates the feedback to adjust the priority values in the evaluation function. Performance is measured based on a wide variety of metrics. Time until completion, exploration percentage, intruder capture-rate, efficiency and task completion.

b) Adaptability to the game: The reinforcement model is very adaptable to its environment, which was ideal for a game with no predetermined exact information. Knowledge is very limited and clear indications of what the agents best moves are, is very hard to ascertain. The complexity and size of the maps required the agent to become progressively more intelligent as its knowledge expanded.

3) Deep Q-Network Agent: This agent is used for the intruder to find an approximation of the optimal path towards the target area, whilst avoiding being caught by the guard agent. This approximation algorithm is trained us Q-learning to evaluate the agent's moves and gradient descent to update the weight and biases in the network.

a) Agent Design: The agent takes a tensor representing the agent's current state as an input. This input is then passed to the agent's convolutional neural network. This network outputs 4 values. These values correspond to the agent movement options.

0 - Up
1 - Down
2 - Left
3 - Right

The max argument of this output is the action with the highest predicted payoff for the agent. Next, the action is executed in the game and a reward is observed. The agent receives a positive reward if it reaches the target area and a negative reward for being caught by the guard. Additionally, smaller rewards are also given, these are discussed in the rewards section. After the move is executed the next state is then calculated.

b) Input Tensor: The input tensor consists of 3 channels made up of 13×13 matrices.This size is so the agents full field of view can be captured whilst maintain the agent orientation in the input.

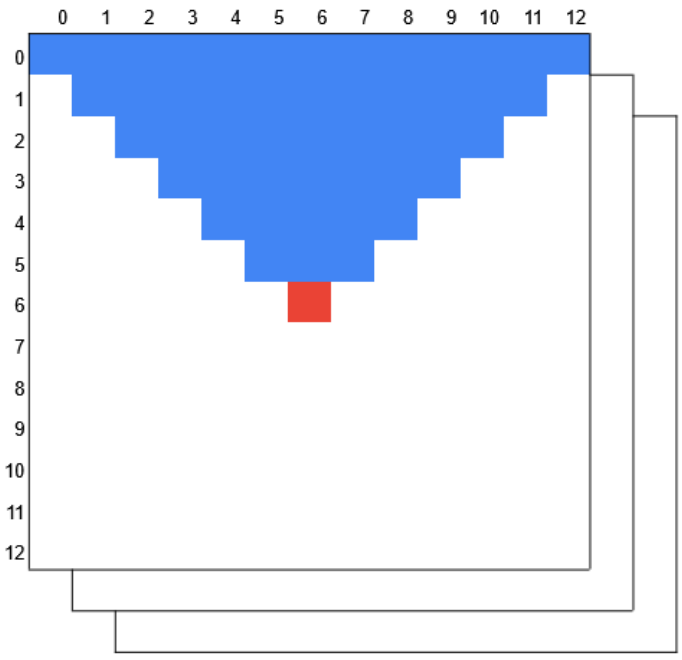


Fig. 3: Input Tensor

In figure 3 the agent's input tensor shows it's looking upward with its field of view highlighted in blue and the agents position in red.

The first channel has 1's in the indices where the agent sees intruder agents and -1 in the indices of guard agents. The second channel has -1's in the indices the agent sees walls and -0.5's in the indices of shadow tiles. The final channel has the sound intensity values the agent hears coming from above, bellow, left and right. Additionally, in this channel there's a 1 in the direction of the target.

c) Training the Agent: The maximum output of the network is the Q-value for a given state, action pair $Q(s, a)$. The goal of the agent is to maximize future rewards using the Bellman equation. Where the agent is trying to maximise $Q^*(s', a')$. Here s' is the next state after executing an action and a' is the optimal action in the next state. $Q^*(s', a')$ is calculated using the following equation [8]:

$$Q^*(s', a') = \mathbb{E}\left[r + \gamma \ \text{max}Q^*(s', a')\right] \tag{1}$$

Here γ is the discount factor and r is the reward observed from the previous state. The max$Q^*(s', a')$ is the high output from the network using the next state as input. The discount factor is included to reduce the priority placed on future rewards.

d) Rewards: As mentioned before the agent receives a positive reward for being inside the target area and also for successfully escaping. If the agent is caught it receives a negative reward or if it sees the guard agent. If the agent selects a move that would result in a collision it receives a positive reward in every other direction that doesn't result in a collision. This is done to prevent too many negative rewards from causing the weights and biases to decay too quickly. The agent tracks the minimum distance it has been from the target. Each time the agent gets closer it receives a small positive reward. Similarly, each time the agent faces the target direction it receives a small positive reward.

e) Network Design: Each agent consisted of two networks. One policy network and one target network. The policy network is used to calculate $Q(s, a)$ and the target network is used to calculate $Q^*(s', a')$. The target network is set to the policy network each time the agent is caught or it reaches the target area. The agent has a move limit, if the move limit is reached in a given game the agent's policy network is reset to the target network. The reasoning behind having two networks for each agent is to develop an optimal policy for both reaching the target area and avoiding being caught by the guard. When training back-propagation is only performed on the policy network.

Additionally, a Markov Decision Process has been used to find a policy that leads to long-term rewards [9].

f) Gradient Descent: The loss function used for this network is mean square error:

$$\text{MSE} = \frac{1}{2}(y_i - \hat{y}_i)^2 \tag{2}$$

Where $y_i = Q^*(s', a')$ from equation (1) above and $\hat{y}_i = Q(s, a)$ predicted Q-value from the previous state. This gives the following equation for the derivative of the loss function for non terminal states.

$$\frac{dE}{dY} = ((r + \gamma \ \text{max} \ Q^*(s', a')) - \text{max} \ Q(s, a)) \tag{3}$$

$\frac{dE}{dY}$ is the derivative of the cost function with respect to the output of the network. In the case of a terminal state, the derivative of the cost function is just the reward observed for reaching the final state.

$$\frac{dE}{dY} = r \tag{4}$$

Batch gradient descent is used for this agent. This is preformed when two conditions are met.

1) The number of experiences $>$ batch size
2) The agent is in a terminal state (caught or escaped)

Once these conditions are met the agent preforms gradient descent on a $U[0, 1]$ random sample of experiences.

IV. Implementation

This section describes the designs and pseudo-code of the methods utilizes in this paper.

A. Game Architecture

Appendix D figure 15 shows the class architecture of the base structure of the game.

The data is managed using a repository pattern on in-memory repositories. Here three distinct repositories are defined, namely:

- Game
- Map
- Player

The "game" ensures that all initial game configurations are maintained. The "map" keeps track of everything that happens on the map and contains all the elements to build a map. The "player" keeps track of the players that are on the map while also handling initial player spawns and their movements.

All repositories are connected to their own "contract" in case the data managed is changed (such as moving to file reading/writing or databases).

B. Agents

The following sub-section contains the designs and psuedocode of the core concepts of the implemented agents.

1) Random: Appendix D figure 16 shows the class architecture of the random agent.

Following the explanation from section III-A1 the pseudocode of the random agent is represented in Appendix B algorithm 1.

2) Yamauchi: Appendix D figure 17 shows the class architecture of the yamauchi agent.

Following the explanation from section III-A2 the pseudocode of the yamauchi agent is represented in Appendix B algorithm 2.

3) SBO: Appendix D figure 18 shows the class architecture of the SBO agent.

Following the explanation from section III-A3 the pseudo-code of the SBO agent is represented in Appendix B algorithm 3.

4) Probabilistic: The probabilistic handles both the guard and intruders. However, both utilize slightly different algorithms.

a) Guard: Appendix D figure 19 shows the class architecture of the Probabilistic agent.

Following the explanation from section III-B1 the pseudo-code of the probabilistic agent is represented in Appendix B algorithm 4.

b) Intruder: Appendix D figure 19 shows the class architecture of the Probabilistic agent.

Following the explanation from section III-B1 the pseudo-code of the probabilistic agent is represented in Appendix B algorithm 5.

5) Reinforcement Learning: Appendix D figure 20 shows the class architecture of the RL agent.

Following the explanation from section III-B2 the pseudo-code of the RL agent is represented in Appendix B algorithm 7.

6) Deep-Q Network: Appendix D figure 21 shows the class architecture of the DQN agent.

Following the explanation from section III-B3 the pseudo-code of the DQN agent is represented in Appendix B algorithm 6.

a) Network Implementation: When determining the size of the network performance was the most important factor. After testing a good compromise for speed and accuracy was found to be 3 convolutional layers followed by two dense layers.

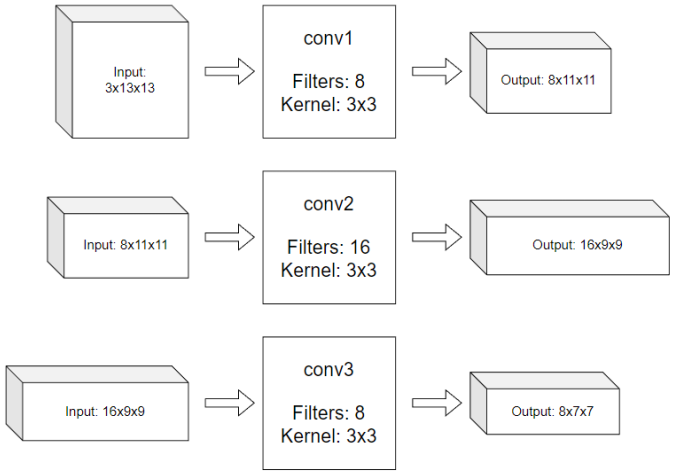


Fig. 4: Convolutional Layers

The final output from the convolutional layer $8 \times 7 \times 7$ in figure 4 is flattened to a vector of length 392. This is passed to the dense layer as an input.

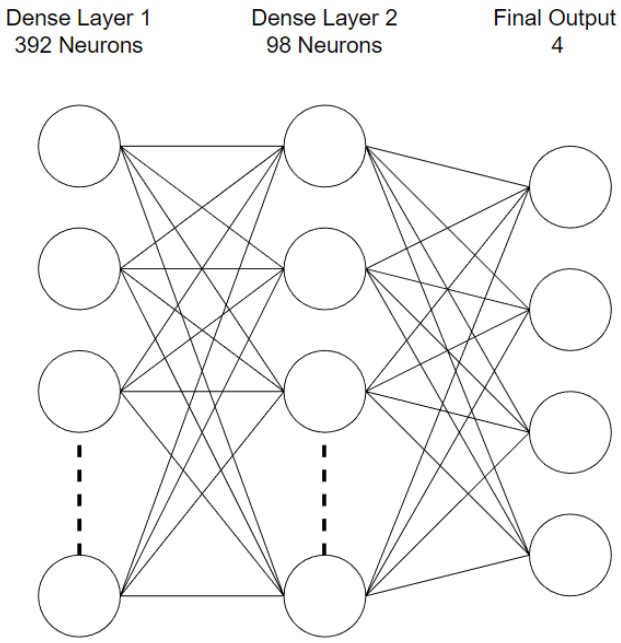


Fig. 5: Dense Layers

For all the convolutional layers and the first dense the Relu as their activation function.

b) Network Parameters:

1) Learning rate = 0.001
2) Discount factor γ = 0.69
3) Batch size = 256

The learning is the same for each layer of the network. The discount factor was decreased from 0.99 to 0.69 to improve the stability of the Q-Values in order to train the agent for more games. The batch size is the number of random sample that are trained against when the agent reaches a terminal state and back-propagates.

V. Experiments

In total, nine different experiments were performed. For the first four experiments, the trained Reinforcement Learning agents played as the guards against the Evader agents as the intruders. For the second four experiments, the Pursuer agents played as the guards against the Evader agents as the intruders. For the last experiment, the DQN agents played as the intruders against the Reinforcement Learning agents as the guards. All intruders focus on escaping and all guards focus on capturing the intruders. For each of the experiments 1000 games are played between the relevant agents with the goal to discover the effect of changing parameters of the game on the number of intruders caught by the guards.

In order to do this, first the standard parameters utilized for the experiments need to be defined. The map on which the games were played is the map provided in Appendix C. This map was carefully created and altered by testing as to provide the least amount of bias, subjectively, towards either the guards or the intruders. The map features all of the implemented

map components, such as teleporters and shaded areas, and allows for indirect communication by the agents placing and recognizing markers as each agent is given five markers of each type. The map also allows for possible advantages for either of the two agents given that they can exploit them, such as avoiding dead ends by placing and recognizing markers. For each game played, three guards and six intruders of the relevant agent are spawned. Some of the other standard parameters, which are worth mentioning as they are altered for the experiments, are the range of the vision and the range of the markers. These are respectively six and three.

For each of the experiments, the intruders caught by the guards are collected and a runs test is performed on this data to discover whether the number of intruders caught between games are independent [10]. The null hypothesis, H_0, is that the number of intruders caught between games are independent. In addition, a chi-square test is performed on the data of the number of intruders caught to discover whether the number of intruders caught follow a normal distribution [10]. The null hypothesis, H_0, is that the number of intruders caught follow a normal distribution.

For both of the tests mentioned above, a significance level of 0.05 is considered.

A. Reinforcement Learning agents vs Evader agents & Pursuer agents vs Evader agents

For both the trained Reinforcement Learning agents and the Pursuer agents, the same four experiments have been performed. Hence, these experiments are discussed in the same section. As mentioned before, the reinforcement agents used are trained and contain the following priority values:

- SoundWave: 0.50
- MarkerDeadEnd: -0.85
- MarkerGuard: -0.45
- MarkerIntruder: 0.90
- MarkerTeleporter: 0.20
- MarkerShaded: -0.30

The first experiment performed are the 1000 games where all the parameters are standard as described above. It is referred to in the 'Results' section by 'Standard Parameters'. The second experiment performed is where a parameter is changed to examine the effect on the number of intruders caught. This changed parameter is the range of the markers, which is increased from the original three to five meaning that the markers can be noticed earlier by the agents. It is referred to in the 'Results' section by 'Marker Range to 5'. For the third experiment, the marker functionality is turned off completely for the guard agents and is referred to in the 'Results' section by 'No Markers'. For the fourth and final experiment, the markers are reintroduced with their range being equal to three again but the range of the vision is changed to be twice as big as specified as part of the standard parameters. It is referred to in the 'Results' section by 'Vision Range Times 2'.

B. DQN agent vs Reinforcement Learning agent

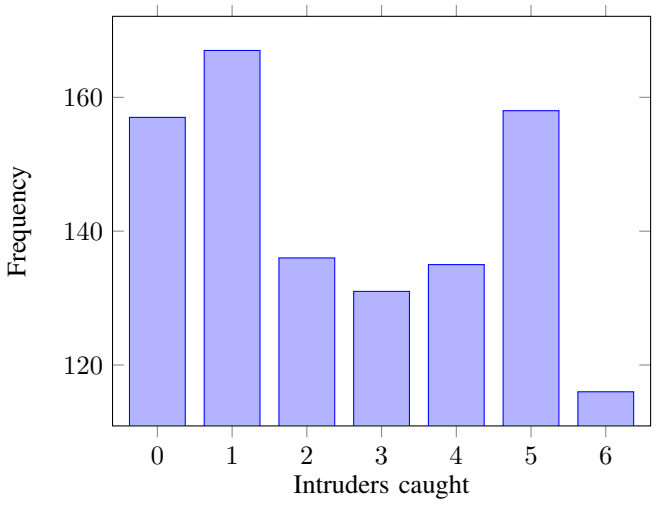
The experiment performed for the DQN agents as the intruders against the Reinforcement Learning agents as the guards are the 1000 games where all the parameters are standard as described above. One exception to this is that the number of intruders and number of guards is equal to one. The reason only one guard and intruder was used is because when multiple intruders trained on the map in Figure 14 the tight spaces caused the intruders to continually collide with one and other. This caused the network weights to decay too fast resulting in NaN outputs.

VI. Results

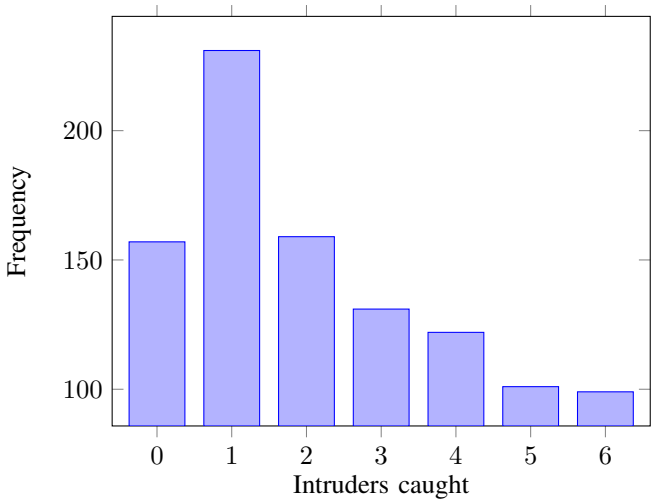
This section displays the results of the experiments described in the 'Experiments' section.

A. Reinforcement learning agent vs Evader agent

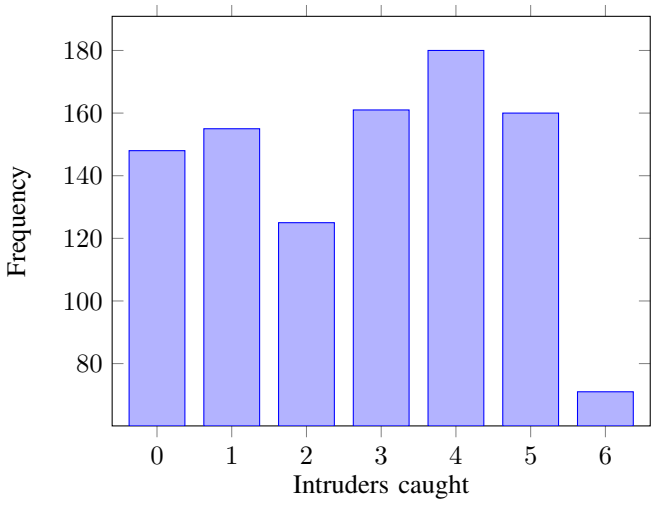
The following four subfigures contain the histograms representing the number of Evader agent intruders caught by the Reinforcement Learning agent guards over 1000 games with the parameters being slightly adjusted for each experiment.



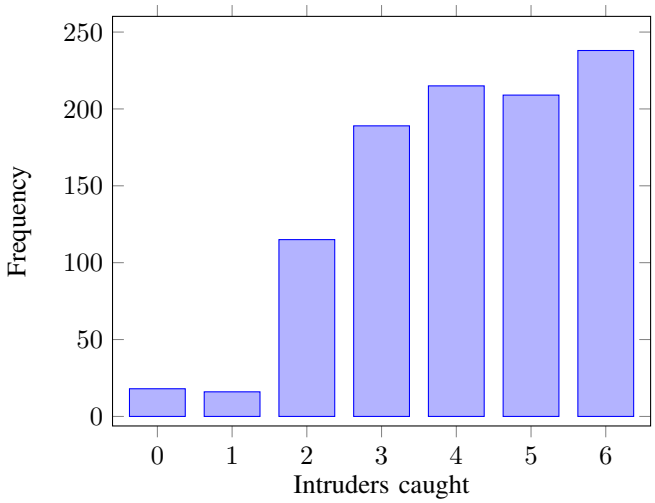
(a) Histogram of experiment 'Standard Parameters' representing the number of Evader agent intruders caught by the Reinforcement Learning agent over 1000 games with standard parameters



(b) Histogram of experiment 'Marker Range to 5' representing the number of Evader agent intruders caught by the Reinforcement Learning agent over 1000 games with range of markers set to five



(c) Histogram of experiment 'No Markers' representing the number of Evader agent intruders caught by the Reinforcement Learning agent over 1000 games with no markers



(d) Histogram of experiment 'Vision Range Times 2' representing the number of Evader agent intruders caught by the Reinforcement Learning agent over 1000 games with vision range times 2

Fig. 6: The histograms representing the number of Evader agent intruders caught by the Reinforcement Learning agent guards over 1000 games with the parameters being slightly adjusted for each experiment

The results of the independence test, the runs test, can be found in Table I. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic R to be: $-1.96 \leq R \leq 1.96$. This means that when the test statistic R is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data is independent. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data is independent.

Experiment	Test statistic R	Result
Standard Parameters	≈ 0.4904	Cannot reject null hypothesis
Marker Range to 5	≈ -0.1968	Cannot reject null hypothesis
No Markers	≈ -0.5255	Cannot reject null hypothesis
Vision Range Times 2	≈ -0.0032	Cannot reject null hypothesis

TABLE I: Results of the independence test, the runs test, performed on the number of Evader agent intruders caught by the Reinforcement Learning agent guards over 1000 games

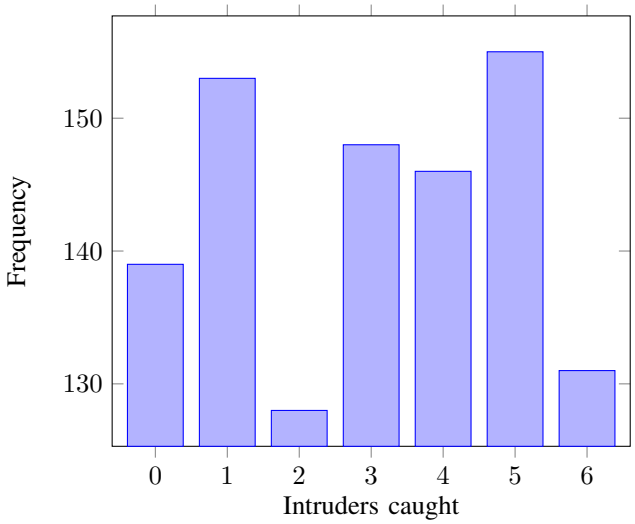
The results of the normality test, the chi-square test, can be found in Table II. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic T to be: $-1.635 \leq R \leq 1.635$. This means that when the test statistic T is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data originates from a normal distribution. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data originates from a normal distribution.

Experiment	Test statistic T	Result
Standard Parameters	≈ 431.9	Reject null hypothesis
Marker Range to 5	≈ 481.6	Reject null hypothesis
No Markers	≈ 259.1	Reject null hypothesis
Vision Range Times 2	≈ 172.1	Reject null hypothesis

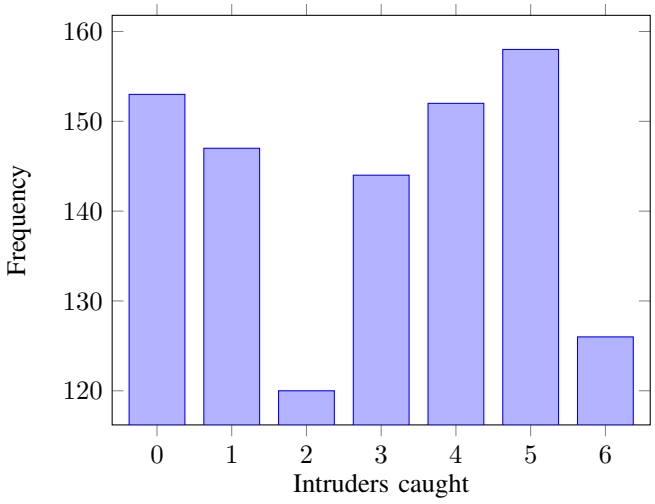
TABLE II: Results of the normality test, the chi-square test, performed on the number of Evader agent intruders caught by the Reinforcement Learning agent guards over 1000 games

B. Pursuer agent vs Evader agent

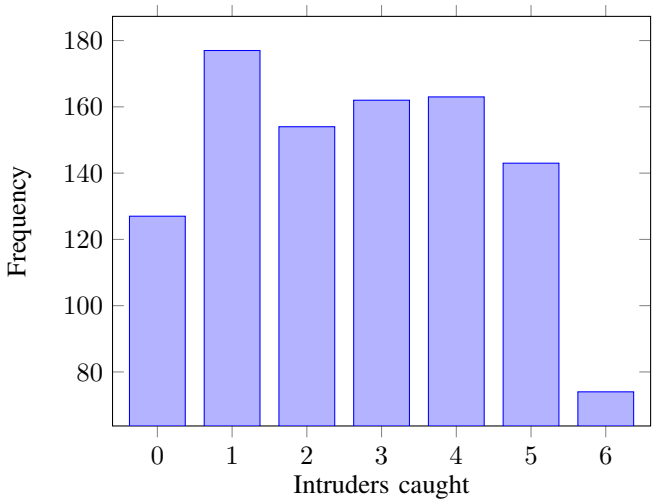
The following four subfigures contain the histograms representing the number of Evader agent intruders caught by the Pursuer agent guards over 1000 games with the parameters being slightly adjusted for each experiment.



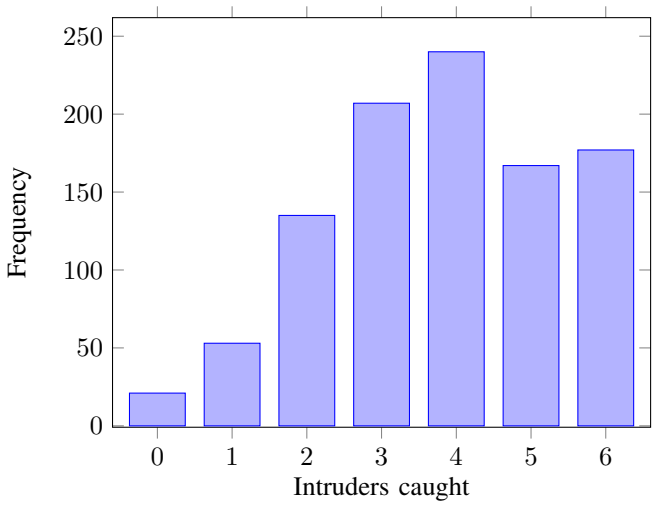
(a) Histogram of experiment 'Standard Parameters' representing the number of Evader agent intruders caught by the Pursuer agent over 1000 games with standard parameters



(b) Histogram of experiment 'Marker Range to 5' representing the number of Evader agent intruders caught by the Pursuer agent over 1000 games with range of markers set to 5



(c) Histogram of experiment 'No Markers' representing the number of Evader agent intruders caught by the Pursuer agent over 1000 games with no markers



(d) Histogram of experiment 'Vision Range Times 2' representing the number of Evader agent intruders caught by the Pursuer agent over 1000 games with vision range times 2

Fig. 7: The histograms representing the number of Evader agent intruders caught by the Pursuer agent guards over 1000 games with the parameters being slightly adjusted for each experiment

The results of the independence test, the runs test, can be found in Table III. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic R to be: $-1.96 \leq R \leq 1.96$. This means that when the test statistic R is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data is independent. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data is independent.

Experiment	Test statistic R	Result
Standard Parameters	≈ -1.7339	Cannot reject null hypothesis
Marker Range to 5	≈ 0.4741	Cannot reject null hypothesis
No Markers	≈ -0.0619	Cannot reject null hypothesis
Vision Range Times 2	≈ 1.3422	Cannot reject null hypothesis

TABLE III: Results of the independence test, the runs test, performed on the number of Evader agent intruders caught by the Pursuer agent guards over 1000 games

The results of the normality test, the chi-square test, can be found in Table IV. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic T to be: $-1.635 \leq R \leq 1.635$. This means that when the test statistic T is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data originates from a normal distribution. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data originates from a normal distribution.

Experiment	Test statistic T	Result
Standard Parameters	≈ 359.5	Reject null hypothesis
Marker Range to 5	≈ 405.1	Reject null hypothesis
No Markers	≈ 188.7	Reject null hypothesis
Vision Range Times 2	≈ 89.8	Reject null hypothesis

TABLE IV: Results of the normality test, the chi-square test, performed on the number of Evader agent intruders caught by the Pursuer agent guards over 1000 games

C. DQN agent vs Reinforcement Learning agent

The following figure contains the histogram representing the number of Reinforcement Learning agent intruders caught by the DQN agent guards over 1000 games with standard parameters.

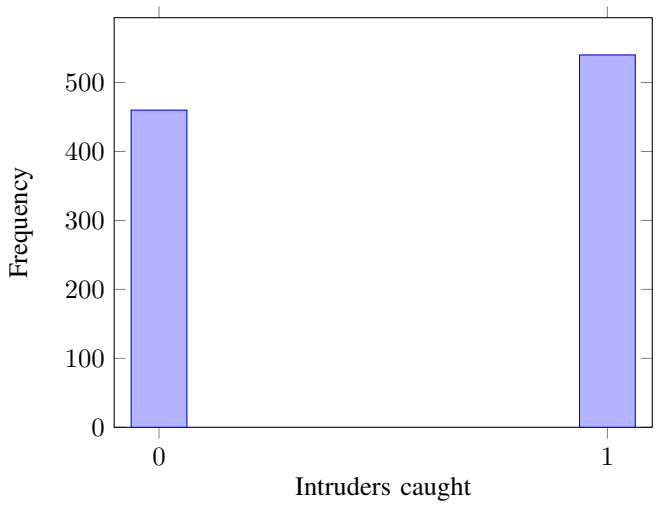


Fig. 8: Histogram of DQN experiment representing the number of Reinforcement Learning agent intruders caught by the DQN agent guard over 1000 games with standard parameters

Here it can be seen the DQN agent won 46% of its games versus the RL agent. The result of the independence test, the runs test, is a value of ≤ -0.4012 considered as our test statistic. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic R to be: $-1.96 \leq R \leq 1.96$. This means that when the test statistic R is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data is independent. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data is independent. In this case, the test statistic falls within that range meaning that it cannot be rejected that the generated data is independent.

The results of the normality test, the chi-square test, is a value of ≤ 102.1 considered as our test statistic. As mentioned before, the significance level chosen is 0.05 resulting in the critical range of the test statistic T to be: $-1.635 \leq R \leq 1.635$. This means that when the test statistic T is in this interval, the null hypothesis cannot be rejected meaning that it can also not be rejected that the generated data originates from a normal distribution. When it is not in the interval, the null hypothesis can be rejected meaning that it can also be rejected that the generated data originates from a normal distribution. In this case, the test statistic falls outside of the range meaning it can reject that the generated data originates from a normal distribution.

VII. Discussion

A. Reinforcement Learning agent

Regarding the results of the Reinforcement Learning agent as the guards against the Evader agent as the intruders, the effects of the marker communication between guards proved to be rather impactful, but not necessarily in a positive way. This can be seen in Figure 6a, Figure 6b, and Figure 6c. In Figure 6a the histogram representing the intruders caught with standard parameters is displayed and in Figure 6b the histogram representing the intruders caught with the range of markers set to five. From these figures can be seen that for the experiment with standard parameters the number of intruders caught starting at around three is much higher than the number of intruders caught than for the experiment with the range of markers set to five. A possible explanation for this could be that in a quite dense map, the smell of the markers would reach different structures. For example, it could happen that a marker is placed over a passage way while denoting a dead end that appears just before that passage way. This could instruct the guard to not walk through the passage way while this could have been a better decision.

In addition, for the experiment with no markers the number of intruders caught is much higher starting at around three than the number of intruders caught for the experiment with standard parameters. These figures can be respectively found in Figure 6c and 6a. This denotes that the Reinforcement Learning agent performs worse when having access to placing and recognizing markers, which could indicate that the markers confuse the guards instead of helping them. A possible explanation for this could be that the Reinforcement Learning would need more training to decide how much importance to

attach to each of the different markers. It might, for example, attach not enough importance to the markers that denote whether an intruder is spotted which could directly lead to catching more intruders. The fact that placing markers takes time is also something that should be taken into consideration. This might cause the markers to backfire. Chasing the intruder could be more important than placing a marker, as it will lose precious time during its chase.

For the experiment with the range of the vision being multiplied by two, the number of intruders caught is much higher starting at around three than the number of intruders caught for the experiment with standard parameters. These figures can be respectively found in Figure 6d and 6a. A possible reason for this could be that the guards are able to spot the intruders more easily resulting in more attempts to chase them. Simultaneously, the intruders see the guards earlier as well which could result in them fleeing from the guards and temporarily abandoning their end goal. This is in fact one of the characteristics of the Evader agent.

As can be seen from Table I, the generated number of intruders caught for all of the experiments discussed above is independent. This can be concluded because the calculated test statistics all appear within the specified interval $-1.96 \leq R \leq 1.96$. This means that the number of intruders caught in a game does not influence the number of intruders caught in the next game, which makes sense since the game is restarted after the previous one is finished.

As can be seen from Table II, the generated number of intruders caught for all of the experiments discussed above do not follow a normal distribution. This can be concluded because the calculated test statistics all appear outside of the specified interval $-1.635 \leq R \leq 1.635$. The consequence of this is that one cannot perform parametric tests as the data is not normally distributed, and also not continuous of course.

B. Pursuer agent

Regarding the results of the Pursuer agent as the guards against the Evader agent as the intruders, it can be seen that with the standard setup Figure 7a the number of intruders caught decreases after one and increases right after two. However when the marker range is set to five as seen in Figure 7b, the number of evader agent intruders caught are significantly higher. A possible explanation for this could be that in a quite dense map, the smell of the markers would reach different structures.

Figure 7c represents the experiment conducted with no markers. As compared to the previous two figures, the number of intruders caught are quite high with only the sixth one being lesser than all the previous ones. This denotes that the Pursuer agent performs worse when having access to placing and recognizing markers, which could indicate that the markers confuse the guards instead of helping them. A possible explanation for this could be that the Pursuer would need more training to decide how much importance to attach to each of the different markers. It might, for example, attach not enough importance to the markers that denote whether an intruder is spotted which could directly lead to catching more intruders.

When the vision range is multiplied by two, which is shown in Figure 7d, the count of the number of intruders being caught is low in the beginning and further increases significantly. A possible reason for this could be that the guards are able to spot the intruders more easily resulting in more attempts to chase them. Also, the intruders can see the guards earlier as well which could result in them fleeing from the guards and temporarily abandoning their end goal.

As it can be seen from Table III, the generated number of intruders caught for all of the experiments discussed above is independent. This can followed from the fact that the test statistic R falls within the interval $-1.96 \leq R \leq 1.96$. This further means that the number of intruders caught in a game has no influence on the number of intruders caught in the next game.

Furthermore, from Table IV, it can be seen that the generated number of intruders caught for all of the experiments discussed above does not follow a normal distribution. This is also proved by the fact that the test statistics does not fall within the interval $-1.635 \leq R \leq 1.635$. Thus, it can be concluded that a parametric test cannot be formed because the data is not normally distributed.

C. DQN agent

The DQN agent won 46% of it's games versus the RL agent. The suspected reason for this discrepancy is believed to be the complexity of the map (see figure 14). As the map had many maze like elements it made training the agent to avoid colliding with walls difficult. Additionally, designing a reward function to incentives backtracking when a dead end is reached was challenging.

As can be seen from the 'Results' on the DQN agent against the Reinforcement Learning agent, the generated number of intruders caught for the experiment discussed above is independent. This can be concluded because the calculated test statistic appears within the specified interval $-1.96 \leq R \leq 1.96$. This means that the number of intruders caught in a game does not influence the number of intruders caught in the next game, which makes sense since the game is restarted after the previous one is finished.

As can be seen from the 'Results' on the DQN agent against the Reinforcement Learning agent, the generated number of intruders caught for the experiment discussed above does not follow a normal distribution. This can be concluded because the calculated test statistic appears outside of the specified interval $-1.635 \leq R \leq 1.635$.

VIII. Conclusion

Surveilling or intruding an area with limited resources using intelligence is utilizes through the usage of exploring a map with algorithms such as Yamauchi Frontier-based or Stack-Based Orthogonal algorithms. Additionally, communication between the agents gave the intruders an improvement on their win-capabilities. The guards could capture intruders by using

Reinforcement learning or a probabilistic (pursuer) approach, for which the RL agent may still improve with more training. The intruders could escape the guards by utilizing the given target direction to their advantage. This is used by both the DQN agent and probabilistic (evader) agent. At times both agents may arrive at a situation where they are colliding with each other on their movement path, to which these situations are resolved by utilizing their communication and knowledge.

All in all, the agents are giving some promising results on a near-equal performance, with a current performance advantage by the intruders. However, some additional training may increase performance of the learning based agents.

REFERENCES

[1] Map Editor, "Tiled." [Online]. Available: https://www.mapeditor.org/

[2] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997, pp. 146–151.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[5] "Frontier exploration - ros wiki," [Online; accessed 16. Jun. 2022]. [Online]. Available: https://wiki.ros.org/frontier$_exploration$

[6] L. Zhang, A. Prorok, and S. Bhattacharya, "Pursuer assignment and control strategies in multi-agent pursuit-evasion under uncertainties," *Frontiers in Robotics and AI*, vol. 8, 2021.

[7] T. Yang, H. Tang, C. Bai, J. Liu, J. Hao, Z. Meng, and P. Liu, "Exploration in deep reinforcement learning: A comprehensive survey," 09 2021.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[9] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," 2015. [Online]. Available: https://arxiv.org/abs/1507.06527

[10] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*. Mcgraw-hill New York, 2007, vol. 3.

APPENDIX A
MAP BUILDER AND MAP COMPONENTS

A. Components

This section explains each separate map component that is present within the game, namely:

- Basic Tiles (see figure 9a)
- Walls (see figure 9b)
- Teleport Source (see figure 9d)
- Teleport Destination (see figure 9e)
- Shadow (see figure 9c)
- Target Area (see figure 9f)
- Vision (see figure 10a)
- Knowledge (Visible in exploration only) (see figure 10b)
- Markers (see figures from 11 and 12)
  - Dead-end (see figures 11a and 12e)
  - Teleporter Spotted (see figures 11b and 12d)
  - Intruder Spotted (guard only on figure 11c)
  - Guard Spotted (intruder only on figure 12c)
  - Shaded Area spotted (see figures 11d and 12a)
  - Target Area spotted (intruder only on figure 12b)

A visual representation of a complete map can be seen in Section II figure 1.

B. Map Builder

Maps can be build and importer using two different approaches:

- Manual Programming (MP)
- Tiled Map Editor (Tiled)

MP uses a text file with a custom structure for both map building and map importation, while Tiled is used for map building and a standardized JSON structure is used for importation.

Both ways of importation use the same naming convention for its keys, however are defined differently through their structures.

1) Manual Programming: Both Map Creating and Importing is done through the same file. There is no GUI for building the maps. Importing the map requires little preparation, but building the map may take a lot of trial-and-error.

```
height = 100
width = 200
gameMode = 0
numGuards = 1
baseSpeedGuard = 5
distanceViewing = 10
numberMarkers = 5
smellingDistance = 5
spawnAreaGuards = 2 2 20 10
wall = 50 0 51 20
wall = 0 0 1 80
wall = 0 79 120 80
wall = 119 0 120 80
wall = 0 0 120 1
wall = 0 0 1 100
wall = 0 99 200 100
wall = 199 0 200 100
wall = 0 0 200 1
teleport = 20 70 25 75 50 90 0.0
```

MP files can only be imported using the MapImporter.

2) Tiled: The Map Creation and Importing are separated from each other. The Map Creation process is done through an external program called "Tiled" [1]. While the map importation requires a little more attention, due to the way Tiled provides the map.

Here Tiled will provide a 1D array of textures, with each index representing it's sequential position from $(0, 0)$ to (X, Y) where X represents the last horizontal (column) element and Y represents the last vertical (row) element.

So a $30x30$ map may look like figure 13.

Here values need to first be mapped to the correct tile type (wall, teleport, shadow, ...), which takes a bit more time in comparison to MP, but the map creation makes it much quicker to make maps using Tiled itself.

Tiled maps can only be imported using the TiledMapImporter.

a) Properties: Both importers support the following game configuration properties:

- height
- width
- name
- gameMode
- numGuards
- numIntruders
- distanceSoundRotating
- distanceSoundWalking
- distanceSoundWaiting
- distanceSoundYelling
- distanceViewing
- markersEnabled
- hearThroughWallsEnabled

Some made nuances to the game is that we do not utilize the following configuration properties:

- scaling
- baseSpeedGuards
- baseSpeedIntruders
- springSpeedIntruders
- timeStep

MP supports the following map properties in their file:

- wall
- teleport
- shaded
- teleporter
- targetArea
- spawnAreaGuards
- spawnAreaIntruders

Tiled supports the following map properties in their file:

- wallTexture
- floorTexture
- guardSpawnTexture
- intruderSpawnTexture
- teleportSourceTexture
- teleportDestinationTexture
- targetAreaTexture
- shadedTexture
- map
- teleporter

Tiled uses the "texture" properties to map the values in the "map" property to their correct tile-type. The "teleporter" property is then used to define the teleport source and destinations.

APPENDIX B
AGENT ALGORITHMS

Algorithm 1 Random Agent

function RANDOMAGENT($player$, $board$, $vision$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
    $value \leftarrow$ randomly generated value between $(0, 100)$
    $move \leftarrow$ get $player$ direction
    $nextTile \leftarrow$ get the $tile$ in front of the agent from the $board$.
    $items \leftarrow$ the items that are on $nextTile$
    $nextBlocked \leftarrow$ False
    for $item$ in $items$ **do**
        if $item$ is collidable **then**
            $nextBlocked \leftarrow$ True
        end if
    end for
    if $value \leq 30$ OR $nextBlocked$ is True **then return** random direction (UP, DOWN, LEFT, RIGHT)
    end if
    return $move$
end function

Algorithm 2 Yamauchi Frontier Agent

function YAMAUCHIFRONTIERAGENT($player$, $board$, $vision$, $knowledge$, $plannedMoves$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $plannedMoves$ a stack of moves to be executed next
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
Ensure: $plannedMoves \geq 0$
2: $markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
if $markerChecked$ is not Empty **then return** $markerChecked$
4: **end if**
if $plannedMoves$ is not Empty **then return** next $move$ from $plannedMoves$
6: **end if**
$detectedFrontiers \leftarrow$ detects all the frontiers on the edge of the $knowledge$ of the $agent$
8: **if** $|detectedFrontiers| \leq 0$ **then return** $randomAgent$ move
end if
10: $chosenFrontier \leftarrow$ pick the best frontier
$plannedMoves \leftarrow$ all the planned moves from $chosenFrontier$
return next $move$ from $plannedMoves$
12: **end function**

Algorithm 3 Stack-Based Orthogonal Agent

function SBOAGENT($player$, $board$, $vision$, $knowledge$, $plannedMoves$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $plannedMoves$ a stack of moves to be executed next
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
Ensure: $plannedMoves \geq 0$
$markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
3: **if** $markerChecked$ is not Empty **then return** $markerChecked$
end if
if $plannedMoves$ is not Empty **then return** next $move$ from $plannedMoves$
6: **end if**
$scanned \leftarrow$ stack of all tiles within $vision$
while scanned is not Empty **do**
9: $top \leftarrow$ top of the $scanned$ stack.
if top is in $knowledge$ **then**
$scanned \leftarrow$scanned$without$top **else**
12: $goal \leftarrow top$
end if
15: **end while**
if path can be computed from $player$ to $goal$ **then**
$plannedMoves \leftarrow$ all moves from $player$ to $goal$
$goal$ is not in $knowledge$
18: $adjacentTiles \leftarrow$ get adjacent tiles from $goal$
for adj in $adjacentTiles$ **do**
if adj is in $knowledge$ AND a path can be computed from $player$ to adj **then**
21: $plannedMoves \leftarrow$ all moves from $player$ to adj
end if
end for
24: **end if**
if $scanned$ is Empty and $plannedMoves$ is Empty **then return** $randomAgent$ move
end if
return next $move$ from $plannedMoves$
27: **end function**

Algorithm 4 Probabilistic Guard Agent

function PROBABILISTICGUARDAGENT($player$, $board$, $vision$, $knowledge$, $audio$, $smell$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $audio$ as the sensory input the agent can hear on its current position
Require: $smell$ as the sensory input the agent can smell on its current position
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
$markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
if $markerChecked$ is not Empty **then return** $markerChecked$
end if
$currentState \leftarrow$ evaluate the current state of the user based on $vision$ and $audio$.
if $currentState$ is Hunting and ($|vision| > 0$ or $|audio| >$) **then**
$huntingSteps \leftarrow$ compute a route to one local target **return** take next move of $huntingSteps$
end if
if $currentState$ is Searching **then return** search near target for X moves
end if
return $yamauchiAgent$ move
end function

Algorithm 5 Probabilistic Intruder Agent

function PROBABILISTICINTRUDERAGENT($player$, $board$, $vision$, $knowledge$, $audio$, $smell$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $audio$ as the sensory input the agent can hear on its current position
Require: $smell$ as the sensory input the agent can smell on its current position
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
$markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
if $markerChecked$ is not Empty **then return** $markerChecked$
end if
$currentState \leftarrow$ evaluate the current state of the user based on $vision$ and $audio$.
if $currentState$ is Fleeing and ($|vision| > 0$ or $|audio| >$) **then**
$fleeingSteps \leftarrow$ compute a route to one local target **return** take next move of $fleeingSteps$
end if
return $yamauchiAgent$ move with preference towards $targetDirection$
end function

Algorithm 6 DQN Agent

function DQNAGENT($player$, $board$, $vision$, $knowledge$, $audio$, $smell$, N)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $audio$ as the sensory input the agent can hear on its current position
Require: $smell$ as the sensory input the agent can smell on its current position
Require: N the capacity of the Neural Network
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
Ensure: $N \geq 0$
$markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
if $markerChecked$ is not Empty **then return** $markerChecked$
end if
$replayMemory \leftarrow$ array of size N
$Q \leftarrow$ array with random weights
for $episode = 1$, M **do**
$s_1 \leftarrow \{x_1\}$
$\phi_1 \leftarrow \phi(s_1)$
for $t = 1$, T **do**
$performRandomAction \leftarrow$ random value between $(0, 1)$
if $performRandomAction \leq \epsilon$ **then**
$a_t \leftarrow$ random move (UP, DOWN, LEFT, RIGHT)
else
$a_t \leftarrow max_a Q^*(\phi(s_t), a; \theta)$
end if
Emulate a_t
Observe r_t
Observe vision x_{t+1}
Set $s_{t+1} \leftarrow s_t, a_t, x_{t+1}$
Pre-process $\phi_{t+1} \leftarrow \phi(s_{t+1})$
$D \leftarrow$ add transition $(\phi_t, a_t, r_t, \phi_{t+1})$
Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
$$y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta))^2 & \text{for non-terminal } \phi_{j+1} \end{cases}$$
perform Gradient descent on $(y_j - Q(\phi_j, a_j;))^2$
end for
end for
return the predicted $move$
end function

Algorithm 7 Reinforcement Learning Agent

function RLAGENT($player$, $board$, $vision$, $knowledge$, $audio$, $smell$, $modelBaseline$)
Require: $vision$ as a $2D$ array of tiles.
Require: $board$ as a $2D$ array representation of the map
Require: $player$ as the player to be considered for the move
Require: $knowledge$ as the known $board$ to the $player$.
Require: $audio$ as the sensory input the agent can hear on its current position
Require: $smell$ as the sensory input the agent can smell on its current position
Require: $typePriority$ as the priority value assigned to an input
Require: $modelBaseline$ as the minimum value an input parameter must surpass
Ensure: $|vision| \geq 0$
Ensure: $|board| \geq 1$
Ensure: $player$ exists on the map
Ensure: $|knowledge| \geq 1$
Ensure: $|modelBaseline| \geq 0$
$markerToBePlaced \leftarrow$ decide whether a $player$ should drop a marker and which marker
if $markerChecked$ is not Empty **then return** $markerChecked$
end if
$assessInput \leftarrow$ prepares and converts all input into readable parameters for evaluation
$parameterEvaluation \leftarrow$ value between -1 and 1
if $parameterEvaluation \geq 0$ **then**
$redirectedMoves \leftarrow$ sequence of actions
else
$redirectedMoves \leftarrow$ sequence of inverted actions
end if
if $vision$ contains $intruder$ **then**
$redirectedMoves \leftarrow$ computed path of all moves from $player$ to $intruder$
end if
$compareParameters \leftarrow$ select max from $assessInput \times typePriority$
if $compareParameters \geq modelBaseline$ **then**
$plannedMoves \leftarrow redirectedMoves$
end if
if $plannedMoves$ is not Empty **then return** next $move$ from $plannedMoves$
end if
$detectedFrontiers \leftarrow$ detects all the frontiers on the edge of the $knowledge$ of the $agent$
if $|detectedFrontiers| \leq 0$ **then return** $randomAgent$ move
end if
$chosenFrontier \leftarrow$ pick the best frontier
$plannedMoves \leftarrow$ all the planned moves from $chosenFrontier$
return next $move$ from $plannedMoves$
end function

APPENDIX C
EXPERIMENTS MAP

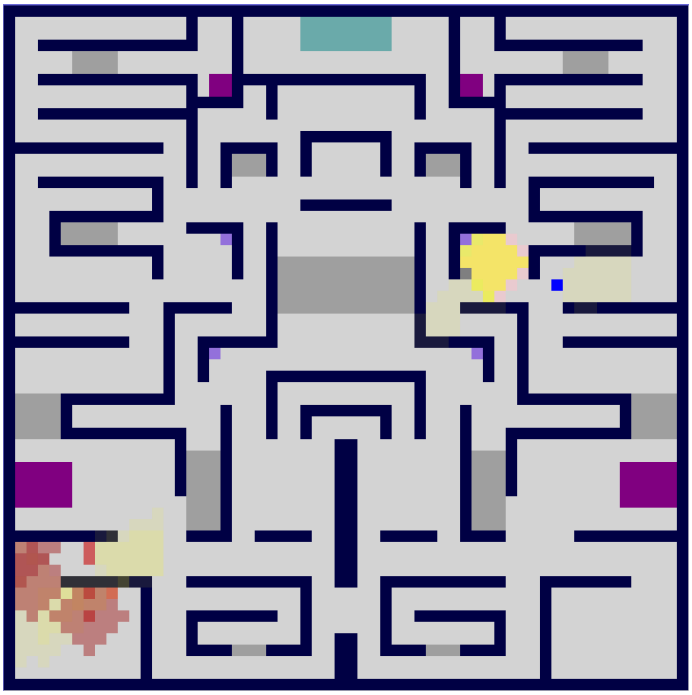


Fig. 14: Map used for experiments

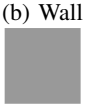
APPENDIX D
SOFTWARE ARCHITECTURE



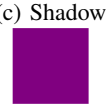
(a) Basic Tile



(b) Wall



(c) Shadow



(d) Teleport Source



(e) Teleport Destination



(f) Target Area

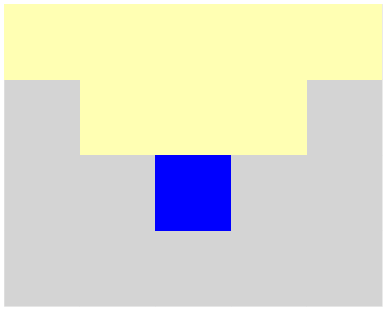


(g) Guard

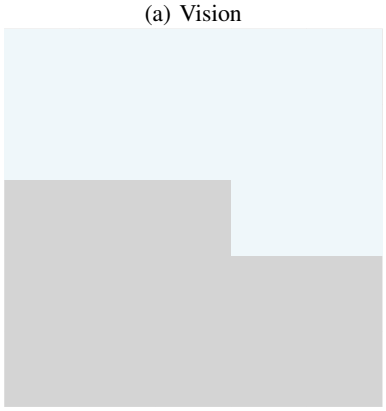


(h) Intruder

Fig. 9: Map Components



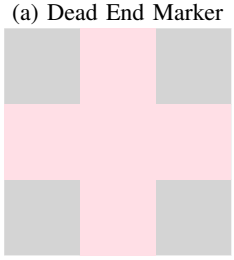
(a) Vision



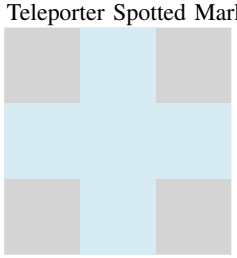
(b) Knowledge

Fig. 10: Vision and Knowledge

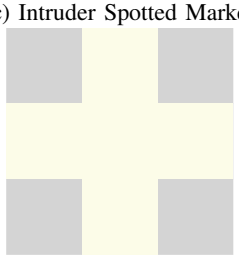
(a) Dead End Marker



(b) Teleporter Spotted Marker



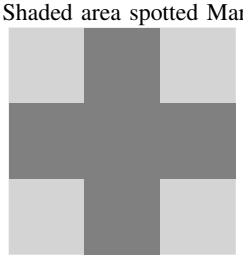
(c) Intruder Spotted Marker



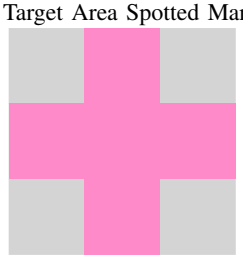
(d) Shaded area spotted Marker

Fig. 11: Guard Markers

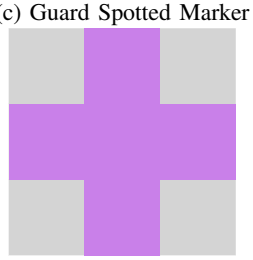
(a) Shaded area spotted Marker



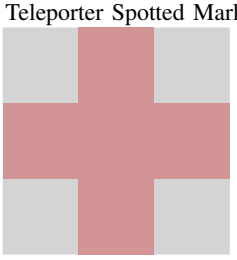
(b) Target Area Spotted Marker



(c) Guard Spotted Marker



(d) Teleporter Spotted Marker



(e) Dead End Marker

Fig. 12: Intruder Markers



Fig. 13: Example 30 by 30 Tiled Map

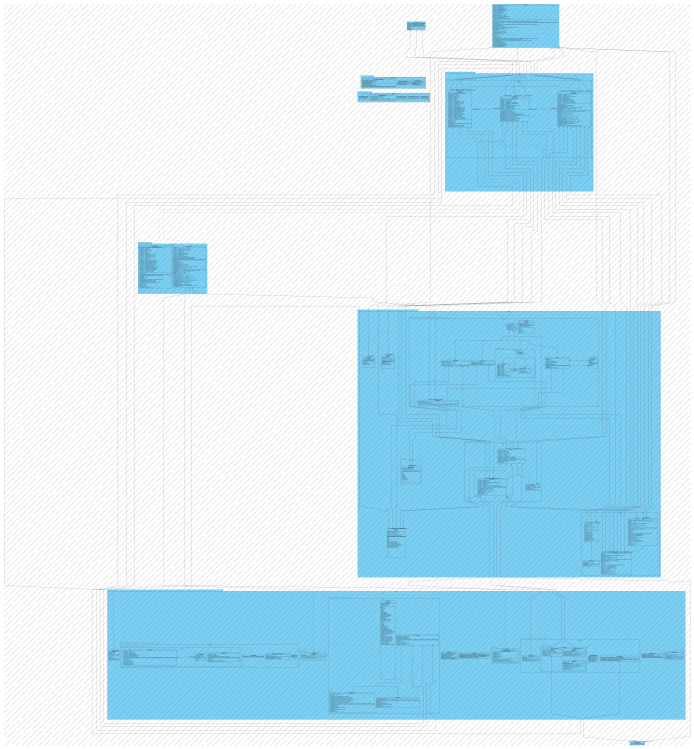


Fig. 15: Logic Architecture

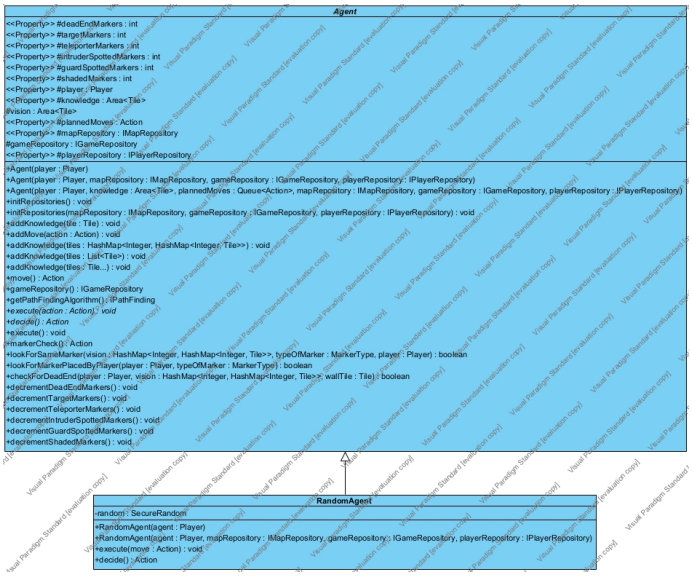


Fig. 16: Random Agent Architecture

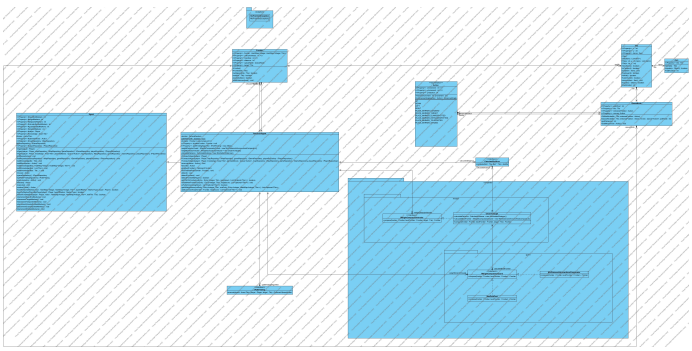


Fig. 17: Yamauchi Agent Architecture

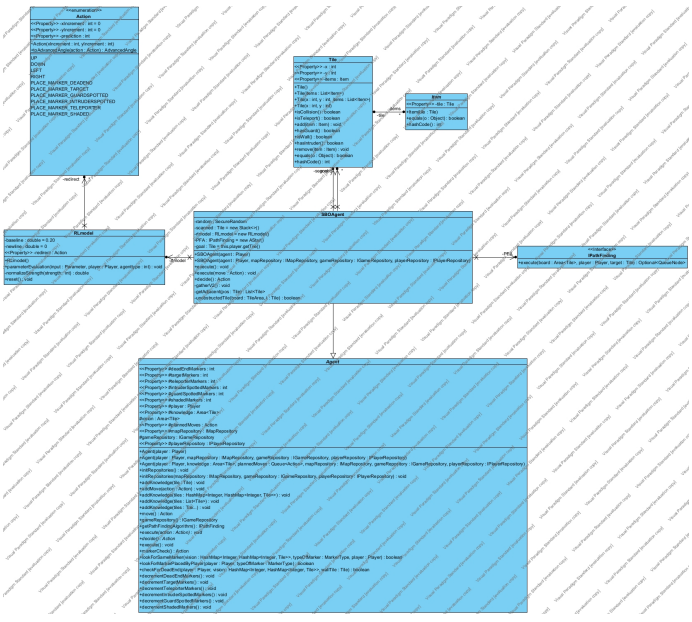


Fig. 18: SBO Agent Architecture

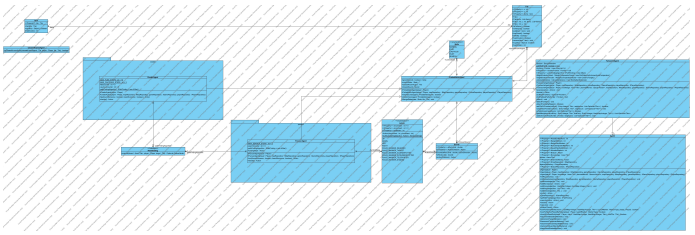


Fig. 19: Probabilistic Agent Architecture

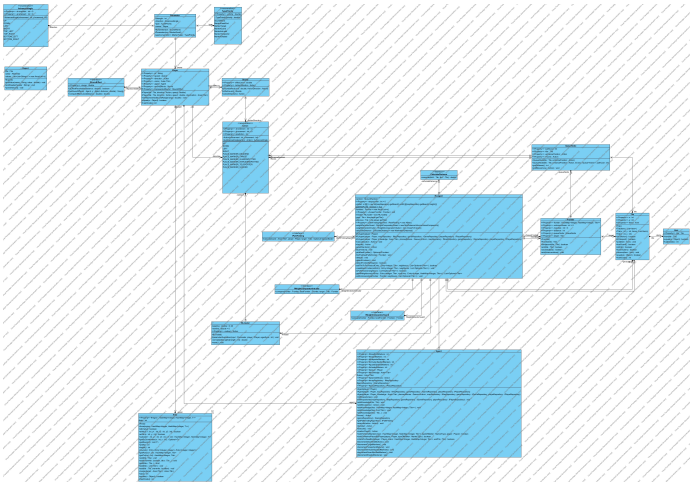


Fig. 20: Reinforcement Learning Agent Architecture

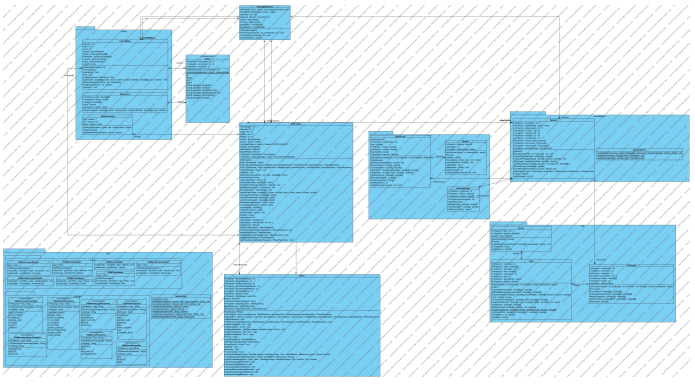


Fig. 21: DQN Agent Architecture