Multi-Agent Surveillance

Jan Grapenthin, Gunda Karnīte, Pepijn van der Meer, Oscar von Moeller,
Imanol Mugarza Pascual, Moath Al Najjar
Department of Advanced Computing Sciences
Maastricht University
Maastricht, The Netherlands

I. Abstract

The integration and utilization of intelligent, autonomous agents in surveillance related tasks has been increasing with the advancements of such technologies and demand for them. What if these agents took bigger roles, for instance being the actual security guards? This would mean that the agents would have to be mobile in the given area that they have to guard, and also be able to communicate with one another. In this paper, we design autonomous agents that have different tasks. Agents are split into two: guards and intruders. The aim of the guards is to catch intruders, while the intruders aim to reach a target state within the map, before getting caught. The intruders are given the general direction of the target area. Both agents use different intelligent algorithms, including A-star (A*), a Rule-based algorithm and NEAT; Q-Learning is specifically implemented for the intruders, while the EVAP algorithm is implemented for the guards. We experimented with the different algorithms to determine which of them is the one that yields the most wins. The guards and the intruders can both see and hear; which led us to experiment with how hearing and vision effected the win rates of both types of agents. From our experiments, we were able to find that both vision and hearing have a mainly positive correlation with the success of either agents, and that the A* was the most successful algorithm for the guard agents.

II. Introduction

Intelligent agents have been utilized in a plethora of different fields [5] [6]. This project aims to program such agents that can simulate an intruder-guard scenario, by splitting the agents into these two groups. The goal for the intruders is to find a given target location before being caught by the guards. The intruders, only know the general direction of the target. The objective of the guards is to explore the environment, locate the intruders and run after them, catching them before the intruders get to the target area.

To successfully analyze and describe the performance of our agents, we formulated the following problem statement: The increase in crime, and more specifically, burglaries has caused a spike in the interest of implementing intelligent, autonomous agents in place of security guards. According to research performed by 'Budget Direct', which analyzed reliable crime rate data from 85 different countries, burglary rates are nowhere near nonexistent, even in lockdown conditions [7]. According to that same research, 31 percent of detained burglars have admitted to that burglary rates would decrease by implementing better security in buildings [7]. Finding a way to improve security likely will correlate with lower burglary rates, making areas with higher burglary rates safer than before. In this report, we aim to explore implementing intelligent agents as both guards and intruders to simulate a burglary into a building. Various algorithms including A-star, Q-Learning, Heuristics, an EVAP algorithm are implemented to determine and simulate the performance of the security guards in catching the intruders (burglars).

This research paper was prepared in partial fulfillment of the requirements for the Project 2-2 of the Bachelor in Data Science and Artificial Intelligence, Maastricht University. Supervisor(s): Marieke Musegaas, Barbara Franci and Franceso Barile

Based on our problem statement, we came up with several research questions:

- Which of the implemented intelligent AIs performs the best at its respective task?
- How are agents' performances affected by changing the viewing distance?
- How are agents' performances affected by changing the hearing distance?

To find an answer to our proposed research questions, we conducted several experiments. Afterwards, the obtained data was analyzed and discussed, helping us arrive to a conclusion.

III. Methods

This section highlights the different algorithms which were used throughout this project. It starts with the Overview Of Agents subsection and is then followed by more specific descriptions of the algorithms that were mentioned.

A. Overview Of Agents

In this section, we describe and explain the different algorithms we used for our agents. Firstly, there is a Baseline agent, which is the simplest agent. It's followed by several intelligent agents based on the following algorithms: A-Star Search, Rule-based, Q-Learning, EVAP Algorithm, NEAT Algorithm.

B. Baseline

A baseline model is generally a simplified one, which executes the necessary tasks in a primitive manner. Although the baseline agent completes the task, most of the time, it does so in a very inefficient way. There is no one specific way of creating a baseline agent, as the skill level and tasks of a baseline agent differ in each model. We chose to give our baseline agents weights for each of the four discrete directions, and these weights change depending on what direction the agent is facing. A more detailed description of how we implemented the agents is given under the Implementations section.

C. A-Star Search

A-Star search is a greedy, path-finding algorithm. The term 'greedy' means that it always chooses the optimal choice at each time step. It finds the most efficient path from a starting point to an end point using a cost function *f*. This cost function is simply the sum of g - the actual cost from the starting point to the current point, and h - the approximated cost from the current point to the target state, referred to as the heuristic cost [1]. The heuristic cost chosen must never overestimate the actual cost. This is to make it admissible, meaning that it always returns the most cost-efficient path to the target [1]. The A-Star method orders all the next steps after each move from the lowest cost to the highest, and chooses the most cost-efficient step. In the Implementations section, we discuss how we utilize the efficient path-finding of the A-Star algorithm and which heuristic cost we use.

D. Rule-based

This agent utilizes rules to determine the action he is taking at every time step. The implementation of this is discussed later.

E. Q-Learning

Q-Learning is an off policy algorithm that finds the best action at every given state. The algorithm seeks to maximize the reward and learns from random actions outside the policy. To rate an action, there is a reward table for every state. At the beginning, the algorithm takes random steps to learn about the environment and update Q-Values. The Q-values are updated with the formula below. To decide if the maximum action should be taken or a random one there is a random number generated, if it is below a certain threshold, a random action is taken, otherwise the max Q-value path is taken. The threshold decays over time, so that probability for the agent to explore the environment, gets smaller over time.

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{Learning Rate}}{\alpha}[\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma}\overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

F. EVAP Algorithm

The EVAP algorithm is an 'ant-inspired' algorithm [3]; this algorithm heavily mimics ants' use of smell markers to communicate. This algorithm relies on the agents secreting pheromones, which are smell markers, whenever they cover any area of the environment they're in. These secreted pheromones evaporate over time, meaning that they become weaker. Agents thus choose paths with the lowest quantities of pheromones to ensure that all areas of the map are frequently visited. This algorithm's mechanism of prioritizing paths with the least pheromones, makes it ideal for discovering unknown environments, as it not only ensures that agents frequently visit the different areas of the environment, but also cover the most ground.

G. NEAT Algorithm

The neuro-evolution of augmenting topologies algorithm (NEAT) is genetic programming applied to feed forward artificial neural network (ANN) training [2]. The networks are described by chromosomes. Using a fitness function, chromosomes are ranked by fitness and may produce offspring for the next generation using crossover. Also, mutation effects the next generation. Connections, weights, and neurons themselves are individually described as alleles and are therefore trainable. For our research, the main benefit is due to its compatibility with sparse learning, allowing us to avoid making assumptions about strategies and instead train based on numbers like win rates.

IV. Implementations

In this section, we present how we built the Scenario environment and implemented the different types of algorithms previously discussed in the Methods section.

A. Scenario

The simulation takes place in an environment that is built from a map, and the agents use it as the play-field. It is built using method *UserInputFile*, which reads in a map given by the user and translates it into a virtual environment. Types of areas that can exist in a scenario are as follows:

- *Target area for intruder*
  The goal location that the intruders need to reach in order to win
- *Spawning area for guards*
  The initial location where the guards are dropped at the beginning of the simulation
- *Spawning area for intruders*
  The initial location where the intruders are dropped at the beginning of the simulation
- *Wall*
  Agents cannot walk through a wall, they limit areas
- *Teleport*
  Teleport takes an agent from one location to another one, where the exit of a teleport is defined
- *Grass*
  Texture to give a meaning to a field, does not mean or do anything

To visualize the simulation, we built a GUI using *JavaFX* libraries, that displays all aforementioned areas, as well as the agents and their movements. It visualizes the viewing area and angle the agent faces at each time step. For all objects, we use PNG images to imitate life-like appearance, making it is easy for everyone to follow and understand the simulation.

B. Baseline Agent

This algorithm implements a similar skeleton to that of the A-star algorithm, it randomly generates target states and uses a queue to order the next moves. What differentiates this algorithm from the A-star algorithm, however, is the way it chooses the node to which the agent is going to move. The choice of the next node is made based on directional weights. Directional weights are simply weights assigned to each of the four discrete directions, for instance, initially the weights, *weightNorth*, *weightEast*, *weightSouth*, and *weightWest*, parameters are initialized to 0.25, 0.5, 0.75, and 1, respectively. A random value between zero and one is generated and depending on which range the random number is in, the agent will move to a certain direction. At first, the probability that the agent goes to a given direction is equal in all four directions. Once the agent goes in a certain direction, the weights are reset, so that the probability that the agent stays on the same path is higher. We decided to do this so that the agent biases going in one direction rather than making a turn in every single step.

C. Random A-Star

For the choice of the next move, an evaluation function is used. This evaluation function consists of two parts: the actual cost it took to get to the current position from the starting point and the estimated cost to reach the goal state, from the current point.

- Cost function: Each move has a cost 10
- Heuristic cost: Cost of the Manhattan distance from the current point to the goal point.

To store and sort all the nodes in the map, we stored all our nodes in two different lists:

- *OpenNode*-a list that consists of nodes that have been visited but not expanded.
- *ClosedNode*- a Boolean matrix that consists of nodes that have been visited and expanded.

The algorithm proceeds as follows: the current state is added to the *OpenNode*, the algorithm then verifies that that position is not a wall or a teleport. In case it is a wall, the agent can not be moved to that square and in case it is a teleport, the agent is transported to the exit of the teleport (if it is neither of these two, *ClosedNode* is set to true). Once this has been verified, the costs of the adjacent squares are updated and the square with the lowest cost is the one chosen for the next step.

1) Diagonal A:* For the Diagonal A*, the cost of the adjacent squares that are in the diagonal is also updated. The cost to reach this square is going to be double because two normal movements need to be done. This process is repeated until we reach the goal state.

D. Agents

Both intelligent surveillance agents, known as guards and intruders, are built using the *Agents* interface. Agents have the same base speed, which varies based on the circumstances they are facing. They can face and move in 4 discrete directions-*up, down, left, right*. Each agent has a base visual range with a limited viewing angle and hearing distance. Vision can be affected by several factors. In a shaded area, their viewing angle is reduced. Agents hear, but they do not distinguish between sounds, meaning that they can hear other agents approaching, but do not know whether it is a guard or an intruder. In order to discover who they hear approaching, the agent must turn around and face them. If agents sprint, they can be heard from a further distance than if they are moving slower.

1) Guards: The goal of a guard is to locate and catch at least one intruder before any intruder gets to the target area, while simultaneously exploring the environment and choosing paths to cover the highest percentage of the map possible. All guards are dropped in the spawn area at the beginning, and they're not familiar with the map, the direction of the target area and even the location of the spawn area of intruders. Guards shout when they see an intruder, and shouting can be distinguished from the other sounds, helping the guards communicate with each other and catch the intruders. All agents can hear this shouting, so intruders use it to escape the guards when they hear them approaching.

2) Intruders: The goal of an intruder is to find the target area and get to it before a guard catches them. At the beginning of the simulation they drop in a spawning area for intruders and the general direction of the target area is known to them. They do not have any knowledge about the map when being dropped in the spawn area. Although intruders do not communicate with each other, they listen for the guards shouting, thus, they know when guards are near. In that case, intruders have an ability to sprint for 15 time steps and escape the guards. Sprinting makes them tired, so after each sprint they need to rest for 5 time steps to have the energy to move again.

E. Rule-based agent

Our rule-based agent uses heuristics to choose an action accordingly. These decisions are made through processing the surrounding observations of an agent and calculating the most desired goal for the current circumstances. Afterwards, A* is used to find the shortest path to it. The new goal is calculated based on the following circumstances:

- Whether a wall is within their visual range
- If they see an opposing agent
- If they hear a sound, they identify it whether it is walking, shouting or sprinting to act appropriately
- If the guard has reached their previously set goal, and they need a new goal

When calculating a new goal, it checks whether the agent can reach it, if not, a new goal is recalculated. To enable the guard to explore the map more efficiently, each guard is assigned to a quadrant and a new goal is generated within its bounds. If they discover a wall, they calculate the way again, because they do not know the initial positions of the walls and therefore need to check if the way they had calculated before is still feasible and not blocked by a wall that was unknown to the agent before. In the case they hear a new sound, they move towards it to identify whether the sound was coming from an intruder. If it is an intruder, they run after them and try to catch them. On the other hand, as soon as an intruder notices that he got discovered, he sprints away from the guard in the opposite direction. They can sprint for 15 time steps, where they move twice as fast as the guards. Obviously, that is not a very good escaping mechanism as it doesn't utilize the surroundings, which is why we decided to improve it with a Q-learning algorithm.

F. Q-learning

Q learning algorithm was chosen for intruders to improve the escape algorithm from the guards. The moment an intruder becomes aware that he has been discovered, the Q-Learning algorithm calculates the best escape route from the guards. Therefore, a reward matrix with the length and width of the intruders sprinting distance is calculated, to ensure that the intruder knows the best step in every direction, until he has to rest after sprinting. With this algorithm, the intruders can exploit the natural terrain of the map to their advantage, like a shaded area and a teleport to escape from the guards. The reward matrix takes different factors into account. The main factor is the distance from the guard that discovered them. The farther the distance is from a guard, the bigger the reward is. If the field is within the viewing distance from a guard, the reward becomes negative, giving intruders an incentive to stay out of the guards' sights. Other factors are shaded areas, portals and the target area. All of these fields get a higher reward, since they're advantageous for the intruders. The Q-Table is then trained on the 500,000 iteration with a decay rate of 0.999999999 and a learning factor and alpha value of 0.9. Based on the Q-table, the method returns an escape path for the intruder to run away.

G. EVAP Algorithm

We used the EVAP algorithm to help maximize the Guards' efficiency in exploring the map, when they haven't spotted any intruders. Guards leave pheromones, which are considered smell markers, whenever they visit a certain node. These pheromones decay over time. We made a copy of the actual map, which saves and updates the pheromone values dropped in the original map. The initial value of a node on the pheromone map, which was just visited by a guard, is set to 10. Every two time steps, the pheromone value decays by a rate of 0.0001 using the following formula:

$$Pn = Pn - 1 * (1 - 0.0001) \tag{1}$$

Where Pn is the pheromone value at the current time step, and Pn-1 is the pheromone value at the previous time step.

To check its surroundings, a guard averages the pheromone values of the eight nodes directly adjacent to it from the pheromone map. This checking process is done every time-step, and is compared to the threshold value, which we set to 1.0. If the value of the threshold is higher than the average value that the guard calculated, the guard patrols its local area, if the average value is higher, however, it means that this area has been visited recently, so the guard patrols a different area, by simply going to a different area of the map.

H. NEAT trained agent

The ANN has, as a stimulus, the knowledge the agent possesses over a three-by-three area of the map centered around a specific point (parts of that area outside of the map are presented as a wall). The response for that area contains:

- A numerical evaluation of the center of the area.
- A chance to the orientation of the agent.
- Desired length of movement.
- If an specific sound should be made, an marker left, or if no additional trace should be left
- If applicable the markers color or specific sound, and the range of smell/the sound

For every position, an efficient path is computed (if available), the evaluation is divided by the number of moves in that path and is contributed to the average of the direction of the first move of that efficient path. The direction with the highest averaged evaluation per move is selected as the agent's move. As for the other actions, those are selected from the current position.

The fitness function randomly pairs ANNs, one by one, of each type and runs for each pair a match for five rounds (every ANN thus plays only one match). If any of the agents of a type (guard or intruder) scored a point during the match (number of captured intruders, and intruders reached the target area respectively), the score is used as the fitness value for the chromosomes describing the ANN's behind the agent's moves. To make sure that the evaluations are also functional in the beginning of training, if no agent of a type scored any points, the average closeness (in distance) to scoring such a punt multiplied by -1 is used as the fitness function.

The ANN has weights of -500 to 500 and a sigmoid activation function. They are trained with a population of 10 per type, survival rate of 0.2, elitism selector with minimum species of one, prune mutation rate of 1, and mutation rates for adding a connection of 0.02, removing a connection of 0.01, adding a neuron of 0.01, and weight of 0.75 with a standard deviation of 1.5.

V. EXPERIMENTS

To find an answer to our research questions, we designed and performed several experiments on our simulation. These experiments can be split into two primary groups: one group experiments with altering the vision and or the hearing of agents, while the other group experiments with putting agents against each other to help us determine the best agent.

Firstly, we chose to test how changing the visual range and hearing distance will affect the performance of our agents. In order to reduce the randomness on the results, we used the same map for all tests. The numbers of guards and intruders were chosen based on the ratio of the two that yielded a fair game, meaning that each of the two groups won approximately 50 percent of the game. We altered the visual and hearing ranges of the two groups, while maintaining the number of agents, to test how the win rates are influenced. The parameters measured after each simulation were:

- the winning agent
- duration of the simulation measured in time steps

Data was obtained from different settings. One of them we set as a default with the following settings:

- Viewing distance = 4
- Hearing distance for walking sound = 7
- Hearing distance for sprinting sound = 10
- Hearing distance for shouting sound = 15

We ran each simulation 50 times, with each variation of either the vision or hearing altered from the default setting. Afterwards, we processed the obtained data and calculated:

- winning rate
- average times steps

for each agent individually and combined for both agents.

Secondly, we tested which of the implemented intelligent AIs performs the best at its respective task. For this analysis, all available agents, namely the Baseline, A*, Diagonal A*, Q-Learning and pheromones are tested against each other. Q-Learning testing only on the intruder and Pheromones only on the guard side, the others for both agents. The head-to-head competition of the agents, are run over at least 100 games to get significant results. The agents are measured by the same metric as the previous experimentation setup with the addition of:

- average coverage of the entire map before a win was materialized

Besides this change, the agent configuration experiments were conducted with the default setup outlined in the previous experimental setup.

The third set of experiments compares multiple generations of both NEAT agent types against the baseline (where the agent instead of using a neural network, uses a list of random values simulating a neural network).

VI. Results

After running several configurations with different amount of guards and intruders (Table IX, Table X), a 5 guards to 4 intruders ratio was found to perform with a win rate closest to 50 percent for both groups (Table I).

Agent	Win Rate	Average Time	Coverage
Guards	58%	300	26%
Intruders	42%	432	36%
Overall	100%	354	30%

TABLE I
Test results using A* for all agents with default settings with 5 Guards and 4 Intruders.

This ratio was used as a default to generate results with changed sensors of the agents. Only one group of agents at a time had its abilities reduced.

For our first set of experiments, the viewing distance was reduced at first by 50% (Table II & Table XI) and by 75% (Table III & Table XII). To show more significant results, tests were run also for making the agents blind (Table IV & Table XIII). Below are three experiments conducted for the vision of the guards. The remaining experiments for visual range can be found in the Appendix.

A. A* with varying vision

1) For Guards:

Agent	Win Rate	Average Time
5 Guards	72%	238
4 Intruders	28%	378

TABLE II
Test results using A* for all agents with Guard viewing reduced by 50%.

Agent	Win Rate	Average Time
5 Guards	4%	369
4 Intruders	96%	378

TABLE III
Test results using A* for all agents with Guard viewing reduced by 75%.

Agent	Win Rate	Average Time
5 Guards	0%	-
4 Intruders	100%	379

TABLE IV
Test results using A* for all agents with Guards being blind.

Hearing ability is split into hearing 3 distinctive sounds - walking, sprinting, and shouting. Hearing distance for walking sound was reduced by 75% (V & VI). Hearing distance for sprinting sound was reduced by 80% (Table XIV & Table XV). Hearing distance for shouting sound was reduced by 75% (Table XVI & XVII). Finally, the agents were deaf, thus not hearing any of the distinctive sounds (Table XVIII & Table XIX). The following two tables highlight the experiments ran with guards' and intruders' hearing to walking beings reduced, the remaining experiments for hearing are also found in the appendix.

2) Walking sound:

Agent	Win Rate	Average Time
5 Guards	40%	312
4 Intruders	60%	444

TABLE V
Test results using A* for all agents with Guards hearing for walking sound reduced by 75%.

Agent	Win Rate	Average Time
5 Guards	62%	310
4 Intruders	38%	427

TABLE VI
Test results using A* for all agents with Intruders hearing for walking sound reduced by 75%.

B. Comparing agents

The second set of experiments focused on comparing the different Agent algorithms against each other. Doing this with A* against Baseline in both configurations (Table XXIX and Table XXV) and the same structure for the other agent combinations. The experiments, run with 5 surveillance agents and 4 intruding agents, showed that there is a high variance in the outcome of the testing. Some intelligent agent combinations were very close, like the pheromone surveillance against the A* intruding agent (Table VII), where the split is right down to 50-50. Other tests of intelligent agents, especially the Q-Learning algorithm for intruders, turned out worse than the other agents in the comparison (Tables XXVII or XXVIII).

Agent	Win Rate	Average Time	Coverage
Guards	50%	277	20%
Intruders	50%	416	30%
Overall	100%	346	25%

TABLE VII
A* PHEROMONES AS GUARD VERSUS A* AS INTRUDER

C. Neat agents

This set of experiments tested ANN's trained in populations sizes of 10 for 10 generations (low for time related reasons). In the tables VIII and XXXIV only the guards won, and only once out of the then games. In the underlying data it is clear that this is always on the same map, in the first turn. For time related reasons not all 10 generations where tested.

Generation	0	1	2	3	4	5	6	7	8
Neat guard wins	1	1	1	1	1	1	1	1	1
Baseline wins	0	0	0	0	0	0	0	0	0

TABLE VIII
GENERATIONS OF NEAT GUARDS COMPETING AGAINST INTRUDERS WITH A RANDOM OUTPUT AS NEURAL NETWORK, PER GENERATION 10 MATCHES ON THE SAME MAPS, 100X100 MAP, UP TO 10 ROUNDS, ALL WINS IN FIRST TURN

VII. DISCUSSION

In the first group of experiments, we tested the effect of altering the visual and/or hearing ranges of agents on their performance, specifically their win rate. Table I depicts the benchmark data with the normal viewing and hearing ranges of both the guards and intruders. The guards had a slight edge overall, and it took them considerably less time to win, on average, when compared to the intruders. This discrepancy in average time to win was expected, as the guards need to catch one of the many intruders to win, while the intruders need to get to the specific target area.

The test conducted in tables VI, V, enabled us to see the influence of decreasing the hearing capabilities of both the guards and the intruders to walking by 75% on their performances. We see that compared to the benchmark data, reducing the hearing of the guards by 75% reduces their win rate from 58% to 40%, which is expected as their hearing is heavily hindered. As for the intruders, reducing their hearing for walking by 75% resulted in their win rate only decreasing by 4%. This is likely because the intruders would be able to hear sprinting from a greater radius than walking, which meant that they'd react to the sprinting sound, rather than the walking sound. This is furthermore backed up by the 22% decrease in the win rate of the intruders when decreasing their hearing to sprinting by 80% in table XV.

The tests conducted in tables XVI highlights the effect of decreasing the guards' ability to hear shouting. Unlike the other experiments, where reducing the hearing abilities reduced the wining rate, the winning rate of the guards increased by decreasing their hearing ability. This result is quite surprising, as the guards' performance should've become worse. This surprising result may be because we didn't run enough experiments, making the chance of randomness higher.

The next set of experiments we conducted focused on altering the visual range. In figure XII, the intruders' vision was reduced by 75%, causing an expected dip in the win percentage, where it decreased from 42% in the benchmark data in table I to 34%. We expected this result, as the intruders were disadvantaged due to them not being able to recognize the guards from a farther distance. In figure III, the guards' vision is reduced by 75%. Surprisingly, the guards were only able to win 4% of the games, which is a significant reduction when compared to their 58% winning rate under normal circumstances. Although this is a major decrease, it is expected. This is because unlike the intruders, the guards don't have the general direction of their target state, therefore, reducing the vision of the guards would've heavily hindered their ability to spot intruders, thus win. For the intruders, on the other hand, having reduced vision would only worsen their ability to escape guards, but they would still be able to get to the target area with the same efficiency as if they had full vision, as they'd still have the general direction of their target area.

A. Comparing Agents

In the second part of the experiments, we tested how the agent's algorithm affected the performance. The surveillance agent was the most powerful, when the A* algorithm in its default setup was used (Tables XXVI, XXV or XXVII). While on the intruder's side, it was surprising to see that the Q-Learning algorithm was performing worse than the simpler A* algorithm and its diagonal derivative (Tables XXII, XX. or XXI). The later build agent for the surveillance part, the pheromones (EVAP), is also worse than the default A* setup that was introduced first (Tables XXX and XXIV). For the intruders no agent was as promising as the diagonal A*, the assumption can be made, that as the agent can walk diagonally in comparison to the other agents which are only able to perform in the horizontal and vertical axis, the agent can run away faster as the guard will take two moves to get to the same location as the intruder with diagonal movement (Tables XXIV and XXX).

For the third set of experiments, it is clear that this implementation of agents with a NEAT trained element is either no use or 10 rounds is not enough to declare a difference between it and the baseline. Potential causes for the first option vary from a fundamental problem with the non-NEAT parts of the agent, how the values generated by the ANN are used, that our fitness function does not provide proper assessments because it does not compare the full generation against the same opponent, or most likely not enough evolution steps to evolve the ANN's.

B. *NEAT trained agents*

Before using the agent models and competition function again, both should be thoughtfully vetted and more time for testing should be reserved.

VIII. Conclusion

By analyzing the data collected from the experiments we ran, the first research question, ”Which of the implemented intelligent AIs performs the best at its respective task?”, can be answered by looking at the results of the AI vs. AI experiments. There, the conclusion can be made that the A* algorithm yields the highest success rate for the surveying agent, while the Diagonal A* helped the intruders the most when tasked to get to the target area. As for the other set of experiments, where both vision and hearing were reduced, we can answer our second research question,”How are agents' performances affected by changing the viewing distance?”, and third research question,”How are agents' performances affected by changing the hearing distance?”. We see that when reducing the hearing ability of either agent for walking or sprinting, their win rate will decrease accordingly, the same goes with reducing the vision, where agents whose vision is reduced win less games than when their vision is normal. Decreasing guards' ability to hear shouting, however, seems to increase their win rate. We plan to explore this strange result in the future by possibly running more experiments and/or changing the rate at which the hearing for guards is reduced.

A. *Follow up*

Possible avenues for further research are running Q-Learning on reward table generated by a NEAT trained ANN, or a NEAT trained ANN deciding the move without post-processing (besides game rules) (most likely Hyper NEAT or a likewise NEAT variant fit for a lager stimulus). Another possible improvement for the future would be choosing the variable values of the EVAP algorithm (decay rate, checking rate, threshold etc.) through a genetic algorithm, to ensure that the chosen values are optimal; in this project we manually tested these values.

References

[1] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path Planning with Modified a Star Algorithm for a Mobile Robot. Procedia Engineering, 96, 59–69.

[2] Stanley, K. O., & Miikkulainen, R. (2002, July). Efficient reinforcement learning through evolving neural network topologies. In Proceedings of the 4th Annual Conference on genetic and evolutionary computation (pp. 569-577).

[3] Glad, A., Simonin, O., Buffet, O., & Charpillet, F. (2008). Theoretical study of ant-based algorithms for multi-agent patrolling. In ECAI 2008 (pp. 626-630). IOS press.

[4] H. N. Chu, A. Glad, O. Simonin, F. Sempe, A. Drogoul and F. Charpillet, ”Swarm Approaches for the Patrolling Problem, Information Propagation vs. Pheromone Evaporation,” 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), 2007, pp. 442-449, doi: 10.1109/ICTAI.2007.80.

[5] Iqbal, S., Altaf, W., Aslam, M., Mahmood, W., & Khan, M. U. G. (2016). Application of intelligent agents in health-care. Artificial Intelligence Review, 46(1), 83-112.

[6] Schleiffer, R. (2002). Intelligent agents in traffic and transportation. Transportation Research Part C: Emerging Technologies, 10(5-6), 325-329.

[7] Budget Direct. (2022, May 4). Home Burglary Rates Around the World. Auto & General. Retrieved June 18, 2022, from https://www.budgetdirect.com.au/home-contents-insurance/home-safety/home-security/global-burglary-rates.html

IX. appendix

A. *A* with default setting*

Agent	Win Rate	Average Time
3 Guards	8%	343
3 Intruders	92%	403

TABLE IX
Test results using A* for all agents with default settings with 3 Guards and 3 Intruders.

Agent	Win Rate	Average Time
6 Guards	68%	310
3 Intruders	32%	450

TABLE X
Test results using A* for all agents with default settings with 6 Guards and 3 Intruders.

1) *For Intruders:*

Agent	Win Rate	Average Time
5 Guards	76%	294
4 Intruders	24%	391

TABLE XI
Test results using A* for all agents with Intruder viewing reduced by 50%.

Agent	Win Rate	Average Time
5 Guards	66%	284
4 Intruders	34%	410

TABLE XII
Test results using A* for all agents with Intruder viewing reduced by 75%.

Agent	Win Rate	Average Time
5 Guards	16%	190
4 Intruders	84%	211

TABLE XIII
Test results using A* for all agents with Intruders being blind.

B. *A* with varying hearing*

1) *Sprinting sound:*

Agent	Win Rate	Average Time
5 Guards	60%	308
4 Intruders	40%	408

TABLE XIV
TEST RESULTS USING A* FOR ALL AGENTS WITH GUARDS HEARING FOR SPRINTING SOUND REDUCED BY 80%.

Agent	Win Rate	Average Time
5 Guards	80%	303
4 Intruders	20%	390

TABLE XV
TEST RESULTS USING A* FOR ALL AGENTS WITH INTRUDERS HEARING FOR SPRINTING SOUND REDUCED BY 80%.

2) *Shouting sound:*

Agent	Win Rate	Average Time
5 Guards	74%	313
4 Intruders	26%	432

TABLE XVI
TEST RESULTS USING A* FOR ALL AGENTS WITH GUARDS HEARING FOR SHOUTING SOUND REDUCED BY 75%.

Agent	Win Rate	Average Time
5 Guards	86%	240
4 Intruders	14%	380

TABLE XVII
TEST RESULTS USING A* FOR ALL AGENTS WITH INTRUDERS HEARING FOR SHOUTING SOUND REDUCED BY 75%.

3) *Agents being deaf:*

Agent	Win Rate	Average Time
5 Guards	40%	261
4 Intruders	60%	432

TABLE XVIII
TEST RESULTS USING A* FOR ALL AGENTS WITH GUARDS BEING DEAF.

Agent	Win Rate	Average Time
5 Guards	90%	227
4 Intruders	10%	379

TABLE XIX
TEST RESULTS USING A* FOR ALL AGENTS WITH INTRUDERS BEING DEAF.

C. AI vs AI

Agent	Win Rate	Average Time	Coverage
Guards	76%	200	18%
Intruders	24%	379	32%
Overall	100%	244	21,76%

TABLE XX
DIAGONAL A* PHEROMONE AS GUARD VERSUS Q-LEARNING AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	59%	230	20%
Intruders	41%	399	36%
Overall	100%	297	26%

TABLE XXI
DIAGONAL A* PHEROMONE AS GUARD VERSUS A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	52%	157	16%
Intruders	48%	294	27%
Overall	100%	224	21%

TABLE XXII
DIAGONAL A* PHEROMONE AS GUARD VERSUS DIAGONAL A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	100%	335	28%
Intruders	0%	0	0%
Overall	100%	335	28%

TABLE XXIII
DIAGONAL A* PHEROMONE AS GUARD VERSUS BASELINE AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	80%	126	11%
Intruders	20%	339	30%
Overall	100%	169	15%

TABLE XXIV
A* QUADRANT AS GUARD VERSUS DIAGONAL A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	100%	410	32%
Intruders	0%	0	0%
Overall	100%	0	32%

TABLE XXV
A* QUADRANT AS GUARD VERSUS BASELINE AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	59%	300	26%
Intruders	41%	432	36%
Overall	100%	354	30%

TABLE XXVI
A* QUADRANT AS GUARD VERSUS A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	86%	244	22%
Intruders	13%	410	35%
Overall	100%	268	24%

TABLE XXVII
A* QUADRANT AS GUARD VERSUS Q-LEARNING AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	75%	194	14%
Intruders	25%	390	28%
Overall	100%	243	18%

TABLE XXVIII
A* PHEROMONES AS GUARD VERSUS Q-LEARNING AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	100%	449	30%
Intruders	0%	0	0%
Overall	100%	449	30%

TABLE XXIX
A* PHEROMONES AS GUARD VERSUS BASELINE AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	54%	139	9%
Intruders	46%	301	21%
Overall	100%	214	15%

TABLE XXX
A* PHEROMONES AS GUARD VERSUS DIAGONAL A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	2%	281	5%
Intruders	98%	379	4%
Overall	100%	377	4%

TABLE XXXI
BASELINE AS GUARD VERSUS Q-LEARNING AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	4%	249	4%
Intruders	96%	379	4%
Overall	100%	373	4%

TABLE XXXII
BASELINE GUARD AGAINST A* AS INTRUDER

Agent	Win Rate	Average Time	Coverage
Guards	2%	154	4%
Intruders	98%	291	4%
Overall	100%	289	4%

TABLE XXXIII
BASELINE GUARD AGAINST DIAGONAL A* AS INTRUDER

Generation	0	1	2	3	4	5	6	7	8
Baseline wins	1	1	1	1	1	1	1	1	1
Neat intruder wins	0	0	0	0	0	0	0	0	0

TABLE XXXIV
GENERATIONS OF NEAT INTRUDERS COMPETING AGAINST GUARDS WITH A RANDOM OUTPUT AS NEURAL NETWORK, PER GENERATION 10 MATCHES ON THE SAME MAPS, 100X100 MAP, UP TO 10 ROUNDS, ALL WINS IN FIRST TURN