CSCI 599: ML FOR GAMES

Final Project Report

MACHINE LEARNING FOR GHOSTBUSTERS

Team Members
Aravind Jyothi
Ilanchezhian Iniya Nehru
Roshini Chakrapani
Sankareswari Govindarajan
Vipul Singh

Table of Contents

ABSTRACT

Ghostbusters is an asymmetric hide-and-search board game with incomplete information. It is a double-sided game with searchers, called ghost busters trying to catch the hider, ghost. The ghost will try to escape the ghost busters and stay hidden so that it doesn't get caught. The ghost busters' goal is to catch the ghost by moving onto the playing board area where it is in the hiding.

The goal of the project is to develop neural network agents in which one will play the ghost busters and the other will play the ghost. We are going to improve the performance of both the sides simultaneously by generating a neural network model.

The agents are modelled using the deep-Q-network based neural network, which is an improvement on Q-learning. It helps model a large state space efficiently.

BACKGROUND

Ghostbusters is a hide-and-search board game with two sides. A group of players, ghostbusters cooperate to track down a hiding player, say ghost which is on the playing board. The board is represented by the underground of an abandoned haunted city.

The game's asymmetric property [1] is contributed by the fact that both sides of the game do not stand on equal ground. The information that is available is incomplete because of the fact that the ghost busters are unaware of the location of the ghost and they are allowed to know the location only at some specific times.

The board game is played by a total of 6 players: 5 players called Ghost Busters and one hiding player caller Ghost. The game is a double-sided game in which the five ghost busters work as a team to capture the ghost in a haunted city which has different underground routes presented on the underground city map.

The game has three properties:

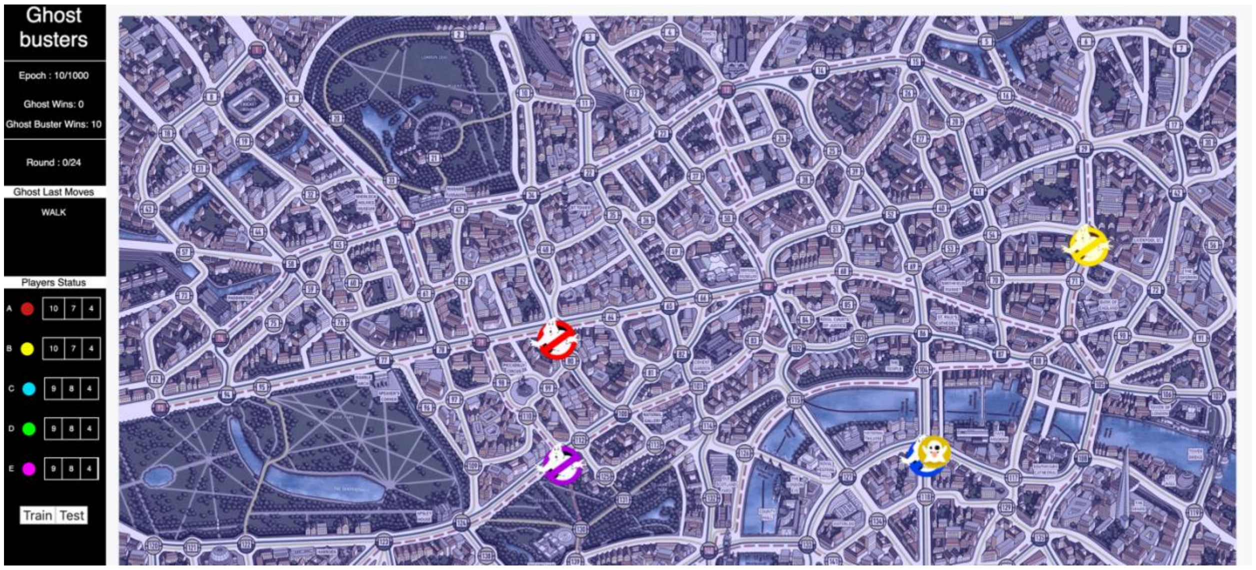
1. The ghost busters perform open moves while the ghost can perform both open and closed moves. Hence, this incomplete information is available to the ghost busters.
2. There is a total of five ghost busters, and it requires that all of them should cooperate and must follow a strategy to catch the ghost.
3. The game is asymmetric, and hence, the game is imbalanced in its goals that make automatic adaptation to certain situations harder.

These three properties are what that's making it a challenging problem to solve using adversarial learning mechanism.

OVERVIEW OF THE GAME ENVIRONMENT

GameBoard

There are 199 locations (stops) numbered from 1 through to 199. These numbered locations are connected by 4 different kinds of ways that represent different passages: walk, sewage, tunnel, and portal. Each numbered location can hold a maximum of one player. In turn each player can occupy a maximum of one location at one time. The point occupied by a player at a specific round is called the location of the player in that round. The six players can start at a random location which can be one of the 18 predefined numbered locations. The game UI is as follows:



Epoch : 10/1000
Ghost Wins: 0
Ghost Buster Wins: 10
Round : 0/24
WALK

A	🔴	10	7	4
B	🟡	10	7	4
C	🔵	9	8	4
D	🟢	9	8	4
E	🟣	9	8	4

Figure 1: Map UI

The action space is the moving of the unit (Ghost or Ghostbusters) from one node to a node that is immediately accessible (exactly one node jump away) using one of the three methods of transport.

- Tunnel allows the node jump to cover multiple nodes in between but is only accessible from a smaller selection of predefined nodes.
- Sewage can cover a smaller number of nodes per node jump but has a larger selection of nodes from which it can be accessed.
- Walk allows the unit to travel exactly through exactly one node per jump but can be accessed from every single node on the map.

Rules

Each ghost buster will be given a total of 22 tokens: 10 walk tokens, 8 sewage tokens, and 4 tunnel tokens. The ghost will start with a total of 12 tokens: 4 walk tokens, 3 sewage tokens, 3 tunnel tokens, and 2 portal tokens. The movement of the players with the ghost starting the game. A series of moves by all the six players (5 ghost busters and ghost) is called a Round. The Ghostbusters board game has a maximum of 24 rounds.

Each player loses a stamina token upon moving from one stop to another based on the type of the chosen passage. When a ghost buster loses a token, it gets added to the ticket collection of the ghost which can be used by the ghost for later rounds. However, when the ghost plays a token, the token is removed from the game and can never be used again.

In the game, the ghost keeps its location a secret, and only in the rounds 3, 8, 13, 18, and 24, it announces its location. The ghost informs the ghost busters about the token used by the ghost in its movement from one location to another. The ghost is caught when any one of the ghostbusters moves onto the location which the ghost is occupying. The goal of the ghost is to avoid being caught by the ghost busters until no ghost buster has any ticket left or none of them can perform a move. A ghost buster cannot perform the move when it does not have a valid token to move from its presently occupied stop.

PRIOR RESEARCH

Search-based heuristics and Learning-based heuristics [2] are the two most popular game-play methodologies. Search-based heuristics can be carried out in a tree-like structure say, game tree or graph. Its usage depends on the properties of the game. Learning-based heuristics' most successful application is the deep neural network to master the game of Go that also used tree search for enumerating all possible moves. It is suggested that the reinforcement of tree search can make the search more efficient and applicable in reality.

In recent times, Hide and Search Game was modeled by using heuristic search algorithms [3]. One of such techniques was attempted in which Monte Carlo Tree Search (MCTS) technique was used as a heuristic search algorithm. In a different development over the study of this kind of game, a complexity analysis has been performed by Sevenster [4], where it has been shown that the generalized version of the game is PSPACE-complete.

It is important that in a double-sided game, each side attempts to win by using its strategy. The strategy of one side becomes an adversary for another side and vice-versa. If both the sides are modeled using mathematical approximators [5], the decisive strategy developed by neural networks in one side is adversarial to the decisive strategy developed by the other side. Such a combination of neural network model is an adversarial neural network that learns through reinforcement, and therefore, the methodology could be called as deep reinforcement learning [6].

In a recent survey, it has been discussed that neuro-evolution can be a better alternative to the existing search-based heuristics. Furthermore, hyper-heuristic, which is possibly a combination of smaller heuristics [7], could solve the game problems in a more efficient way.

ARCHITECTURE FLOW DIAGRAM

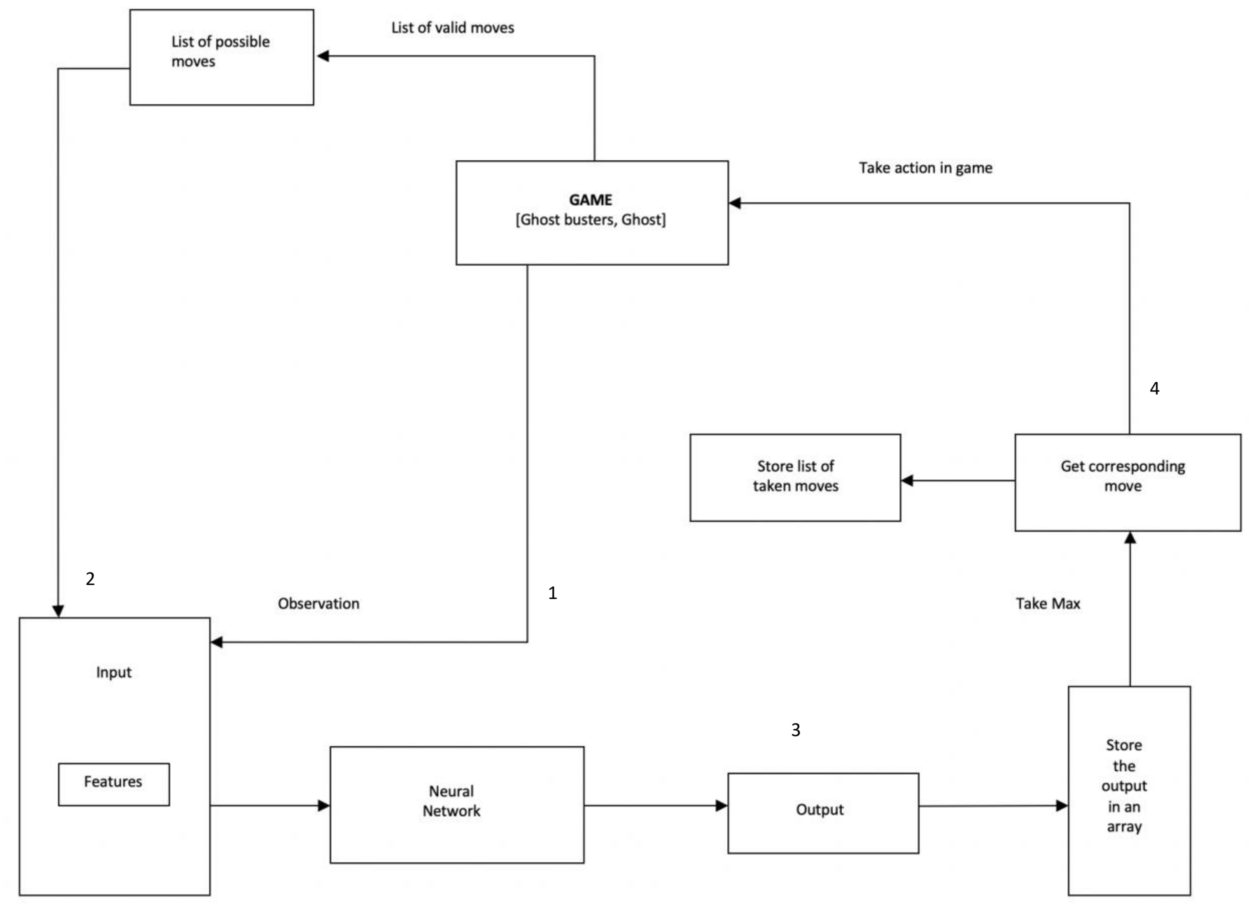


Figure 2: Architectural flow diagram of the Ghostbusters game

A process flow diagram of the model, and the numbers are the important milestones:

(1) The game decides whose turn it is and selects the particular neural network. The game consists of Ghostbusters G1–G5 and Ghost. The game gives out the general observation along with a list of valid moves.

(2) Each move is independently appended to the general observation and queried through the neural network.

(3, 4) From the list, the maximum action (optimum) is chosen, and the corresponding move is communicated back to the game to make the move. The list is stored as well, to learn from it once the final result is out.

ENVIRONMENT SETUP

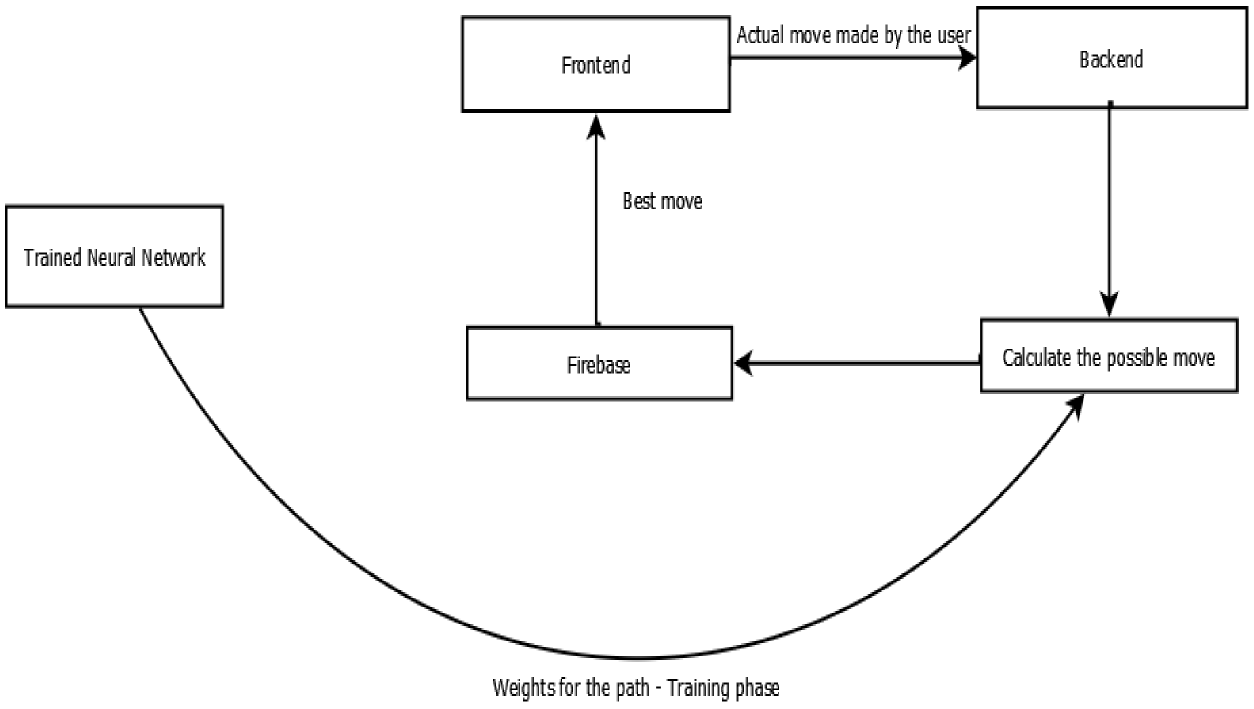

Frontend
Actual move made by the user
Backend
Best move
Trained Neural Network
Firebase
Calculate the possible move
Weights for the path - Training phase

Figure 3: Environmental Setup of the Ghostbusters game

Environment Details

- Neural Network
    - PyTorch
    - Python 3.6
    - Libraries: Matplotlib, NumPy
- Backend
    - Python 3.6
    - Firebase
- Frontend
    - ReactJS
    - Redux

METHODOLOGY UNDERTAKEN

In order to implement automated ghost busters and ghost as adversarial agents, we can implement using Deep Reinforcement based algorithm. The players are modeled as neural network agents that tries to approximate the Q values that are to be obtained at each stage of the game. In traditional Q learning based approach, Q values at each state of the game are stored in the Q table. But since there is an exponential increase in the state space and the possible actions at each stage, it is not possible or advised to use traditional Q learning based algorithm [8]. To replace the Q table to output Q values for each action at every stage, we use neural networks which are the best approximators. In order to explore the behavior of a bot as a neural network agent we are implementing the ghost as Deep Q-learning bot. The ghost busters generate random moves to train the ghost.

The ideal function approximator is the target neural network. The Policy network tries to recreate the values achieved by the Target network on a closer level [9]. The Target network gets its weight values from the Policy network after every 5 updates. The update of weight occurs using the Q values obtained from the target network in the Bellman equation.

Hyperparameters

- ADAM optimizer
- RELU activation
- Learning rate = 0.001
- RMSE loss

Deep Q learning model

- Reward function - Shortest distance between Ghost and any Ghost Buster
- Policy Neural network for Ghost
- Target Neural network for Ghost
- Replay memory size = 1000
- Replay memory sampling size = 256
- Exploration rate - initially set to 0.9 and gradually decreases
- Discount factor = 0.9

The hyperparameters related to neural networks and the Q-learning are set as follows. The neural networks are fully connected (i.e., multilayer perceptron) that uses ReLU (rectified linear unit) activation model [10]. Adam (a stochastic gradient descent technique) [11] optimizer was used for training the neural networks with learning rate being set to 0.001. The neural network architectures for the Ghost and the Ghostbusters are provided below.

<u>Model for Ghost:</u>

Reward function

- +100 for win
- -100 for loss
- 0 for intermediate rewards

Neural network for the Ghost:

Feature Space

There are 1213 input feature for the neural network of the ghost which comprises of the following:

- Encoded location of Ghost
- Number of resources that can be used by Ghost (Walk, Sewage, Tunnel, Portal)
- Encoded location of each Ghostbuster
- Number of resources that can be used by each Ghostbuster (Walk, Sewage, Tunnel)
- Current round number
- Current move number

Layers

LAYER	***SIZE***
Input	1213 x 1
Hidden 1	708 x 1
Hidden 2	708 x 1
Hidden 3	354 x 1
Output	16 x 1

Table 1: Layers of the NN for the Ghost

<u>Model for Ghostbusters:</u>

Reward function

For each ghostbuster: (1/dist(ghostbuster_i, Ghost))*100

Neural network for Ghostbusters:

Feature Space

There are 1215 input feature for the neural network of Ghostbuster which comprises of the following:

- Encoded location of Ghostbusters
- Number of resources that can be used by Ghost (Walk, Sewage, Tunnel, Portal)
- Encoded location of each Ghostbuster
- Number of resources that can be used by each Ghostbuster (Walk, Sewage, Tunnel)
- Current move number
- Ghostbuster number (to indicate the ghostbuster)

Layers

LAYER	***SIZE***
Input	1215 x 1
Hidden 1	708 x 1
Hidden 2	708 x 1
Hidden 3	354 x 1
Output	275 x 1

Table 2: Layers of the NN for the Ghostbusters

The network modeled for the ghostbusters is the single model, where there is one neural network that acts as a virtual head and provides the move for each of the ghostbusters.

For training, the algorithm is run for 5000 games and in each game, there is a maximum of 25 turns. During each turn there occurs a batch update of size of the replay memory sampling size. To analyze the performance of the model we are considering the metric in terms of difference between the number of games ghostbusters won and the number of games the Ghost won. For testing, the algorithm is run for roughly 1000 games and the outcome of each game is recorded.

RESULTS AND ANALYSIS

Bot for Ghost – Training

The Model was trained for 5000 games and the results were:

Ghost wins: 3157 times
Ghostbusters win: 1843 times
Accuracy: 63.14%

Explanation: The graph shows exploration on the x-axis and the performance measure on the y-axis which we measure as Ghostbuster winning – Ghost winning. As the exploration rate is high (leftmost bottom corner) we see the Ghostbusters winning more games as the Bot is currently exploring. As the exploration rate gets lower and the bot starting learning, we see the values getting more positive which means the bot is exploiting the environment and slowly winning more games.

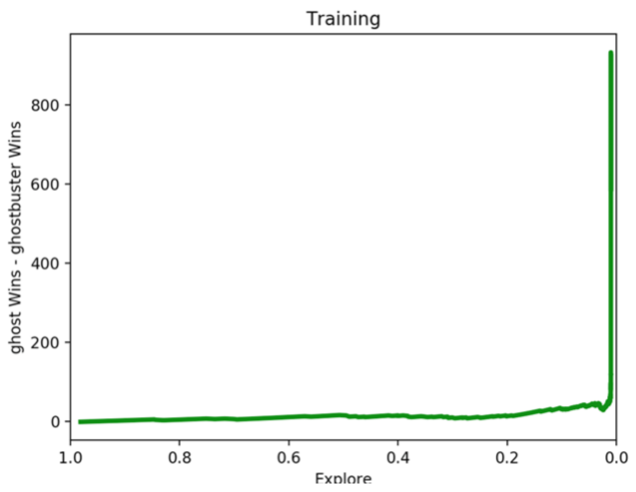


Table 3: Training Graph for Ghost

Bot for Ghost - Testing

The model was tested with 1000 games and the results were:

Ghost wins: 710 times
Ghostbusters win: 290 times
Accuracy: 71.0%

Explanation: The graph shows the game number on the x-axis and the performance measure on the y-axis which we measure as Ghostbuster winning – Ghost winning. We see an almost constant increase in the number of games won by Ghost as the number of games increase.

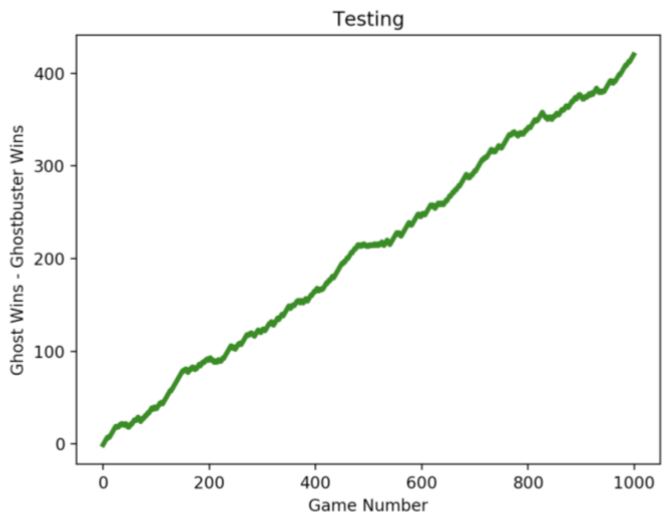


Table 4: Testing Result for Ghost

Bot for Ghostbuster – Training

The Model was trained for 5000 games and the results were:

Ghostbusters win: 1722 times
Ghost wins: 3278 times
Accuracy: 34.44 %

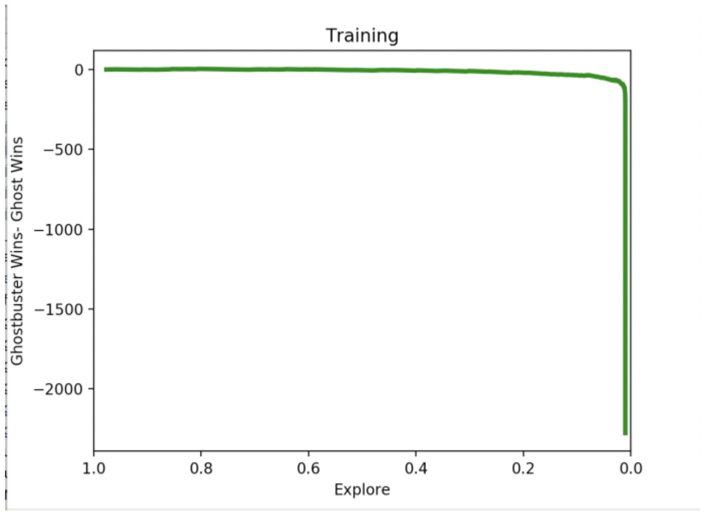


Table 5: Training Result for Ghostbusters

Explanation: The graph shows exploration on the x-axis and the performance measure on the y-axis which we measure as Ghost winning - Ghostbuster winning. As the exploration rate is high (leftmost top corner) we see Ghost winning more games as the Ghostbuster Bot is currently exploring. As the exploration rate gets lower, we see the values getting more negative on the y-axis which means the bot is exploiting the environment and slowly winning more games.

Bot for Ghostbuster – Testing

The model was tested with 1000 games and the results were:

Ghostbusters win: 304 times
Ghost wins: 696 times
Accuracy: 30.4 %

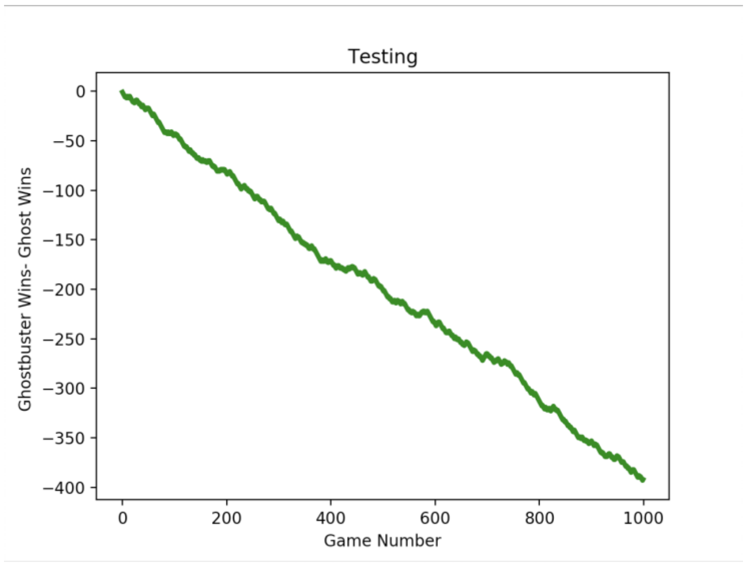


Table 6: Testing Result for Ghostbusters

Explanation: The graph shows the game number on the x-axis and the performance measure on the y-axis which we measure as Ghostbuster winning - Ghost winning. We see an almost constant increase in the number of games won by the Ghostbusters as the number of games increases.

Bot against each other Ghostbuster – Testing

The model was tested with 200 games and the results were:

Ghostbusters win: 152 times
Ghost wins: 48 times

Explanation: The ghost neural network performs better than the ghostbuster because the learning for ghost is based on a single player whereas the learning for ghostbuster is based on the consensus of 5 players targeting towards a single goal.

LIMITATIONS

Due to hardware resource constraints the testing and training has been constrained only to 1000 episodes and the results have been evaluated. The solution mainly focuses on automating the game play between the ghost and the ghostbuster, in the course of which the coordination among the ghostbusters is neglected. In our case, the neural network is not that big so the learning aspect for ghostbuster is not that great as well, which means it requires absurdly huge amounts of time and experience [12].

CONCLUSION

Based on the above analysis, we can assert that the bots for Ghost and the Ghostbusters were able to learn those moves that it made during the exploration phase, that resulted in the respective bot winning. Since the opponent for the training phase was a random move maker, the bot could not learn any strategy of the opponent. But the bots were able to learn from previously made wrong and right moves. Hence the bots were able to win more games in comparison with the random move generator.

FUTURE WORK

The game is currently tested on a very simplistic map-based UI, can be extended to the actual game environment and the performance can be benchmarked. Shared reward system, in which the ghostbusters communicate with each other the results should further improve the results. Individual neural agent for each of the ghostbuster can be developed and the results can be investigated [13].

The future algorithms should develop a better strategic communication among the ghostbusters so that they can coordinate to devise a plan at each stage to catch Ghost. The plan could be devised using statistical data from previous game which are observations of Ghost's behavior.

REFERENCES

[1] Coulom R. (2007) Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik H.J., Ciancarini P., Donkers H.H.L.M. (eds) Computers and Games. CG 2006. Lecture Notes in Computer Science, vol 4630. Springer, Berlin, Heidelberg.

[2] Jetal Hunt K, Sbarbaro D, Zbikowski R, Gawthrop PJ (1992) Neural networks for control systems—a survey.

[3] Georgeff, M. P., Ingrand François, F. (1989). Decision-making in an embedded reasoning system. In *IJCAI*, pp. 972–978.

[4] Nijssen, J.A.M. & Winands, Mark. (2012). Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard. IEEE Transactions on Computational Intelligence and AI in Games. 4. 282 - 294.

[5] Sevenster, Merlijn. (2006). The complexity of Scotland Yard. Journal of Pharmacology and Experimental Therapeutics - J PHARMACOL EXP THER.

[6] Dash, T., Dambekodi, S.N., Reddy, P.N. *et al.* Adversarial neural networks for playing hide-and-search board game Scotland Yard. *Neural Comput & Applic* (2018).

[7] Sturtevant, Nathan R. and Richard E. Korf. "On Pruning Techniques for Multi-Player Games." *AAAI/IAAI* (2000).

[8] Wu, Lin and Pierre Baldi. "Learning to play Go using recursive neural networks." *Neural networks: the official journal of the International Neural Network Society* 21 9 (2008): 1392-400.

[9] Zuckerman, I., Kraus, S. & Rosenschein, J.S. The adversarial activity model for bounded rational agents. *Auton Agent Multi-Agent Syst* **24,** 374–409 (2012).

[10] Rehak, M., Pechoucek, M., & Tozicka, J. (2005). Adversarial behavior in multi-agent systems. In M. Pechoucek, P. Petta, & L. Z. Varga (Eds.) *Multi-agent systems and applications IV: 4th International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2005*, number 3690 in LNCS, LNAI.

[11] McGonigal J (2011) Reality is broken: why games make us better and how they can change the world. Penguin, London.

[12] Luckhart C, Irani KB (1986) An algorithmic solution of n-person games. In: AAAI, vol 86, pp 158–162.

[13] Subelman EJ (1981) A hide-search game. J Appl Probab 18(03):628–640