Arcade Learning Environment - Maze Games
CS 748 Project Report

Harish Koilada
rimcoharish@cse.iitb.ac.in

Sai Krishna Deepak Maram
mskdeepak@cse.iitb.ac.in

ABSTRACT

In this article we show that for games that involve navigating a maze, simple rule-based agents that make use of some domain dependent features perform better than the DQN agent that makes use of deep reinforcement learning [8]. We show the results for three such games Ms. Pac-man, Bank Heist and Amidar. Further we show that the scores achieved can be improved by tuning the various parameters used in the algorithms by using techniques such as CMA-ES [3].

1. INTRODUCTION

Arcade Learning Environment(ALE) [1] provides a framework for training, testing and comparing various domain-independent agents with general competency. ALE makes use of the Atari 2600 game environments as such algorithms need to be able to perform across various domains. Atari 2600 is a second generation game console for which over 500 different games have been developed. These games span various domains and suit perfectly well for the purpose of ALE. ALE provides an interface for an agent to play any arbitrary Atari 2600 games and is built on top of Stella, an open-source Atari 2600 emulator. Each state consists of a single game screen (frame): a 2D array of 7-bit pixels, 160 pixels wide by 210 pixels high. The action space consists of the 18 discrete actions. When running in real-time, the simulator generates 60 frames per second.

For the experiments that were conducted as part of the project the RL Glue interface is used. RL-Glue (Reinforcement Learning Glue) [10] provides a standard interface that allows us to connect reinforcement learning agents, environments, and experiment programs together with the flexibility of writing them in different languages.

2. EXISTING RESEARCH

The benchmark for ALE has been set by Bellemare et al. [1] for Reinforcement Learning and Planning. In the Model-free Reinforcement Learning setting the traditional Sarsa(λ) is used with linear function approximation, replacing traces and ϵ-greedy exploration. Various features extractors such as Basic, BASS, DISCO, LSH and RAM are used to train model. The search methods used in model-based planning are Breadth-first Search and UCT: Upper Confidence Bounds Applied to Trees. The paper also provides an evaluation metric to follow for comparing agents with each other. Finally a detailed table of results are presented which can be used for comparing the performance of a new agent with respect to the above method.

Following up on the previous result, Mnih et al. [8] uses deep learning with Q-learning (Deep Q-Network DQN) to improve the performance of the games. The network architecture uses Convolutional Neural Network to process the game screen followed by two fully connected layers. The results presented in this paper show that the performance of such network exceeds human expert level in many Atari games which can be seen in Figure 1. H van Hasselt et al. [11] uses an improvement of the above method using double Q-learning. The Guo et al. [2] paper uses offline Monte-Carlo Tree Search planning along with deep learning and claims that the agent performs better than the DQN agent.

The Neuroevolution approach Hyper-NEAT has been used in Hausknecht et al. (2012) [4] and the HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player has been introduced. Later on this approach has been compared with other Neuroevolution approaches such as fixed neural networks (Conventional Neuro-evolution), Co-variance Matrix Adaptation Evolution Strategy (CMA-ES), evolution of network topology and weights (NEAT) in Hausknecht et al. (2014) [5]. Some of the games such as Bowling, Kung Fu Master, and Video Pinball have beaten the human high scores using the HyperNEAT-GGP agent.

Since a lot of research exists for the Ms. Pac-man game, a competition took place at Conference on Computational Intelligence and Games, CIG 2011 in which various agents were developed to simulate the behavior of both the Ms. Pac-man and the ghosts. The first place in this competition was bagged by an agent implementing UCT for Ms. Pac-man [6]. At the second place, we have a rule-based Ms. Pac-man agent called ICE Pambush 5 [7].

3. PROBLEM STATEMENT

From the results in Figure 1 it is evident that for games such as Ms. Pac-man, Bank Heist, Amidar, maze games in general, the performance of DQN agent is below human-level. In these set of games the agent has to perform a set of tasks by navigating through a maze and try to maximize the game score. The agent needs to identify the set of actions that are useful in a situation and act accordingly to gain the maximum reward.

The aim of this project is to build rule based agents that use some game dependent features and compare it with the DQN agent. In the next section we discuss the game-play of the three games Ms. Pac-man, Bank Heist and Amidar which were chosen to improve the scores in comparison with the DQN agent.

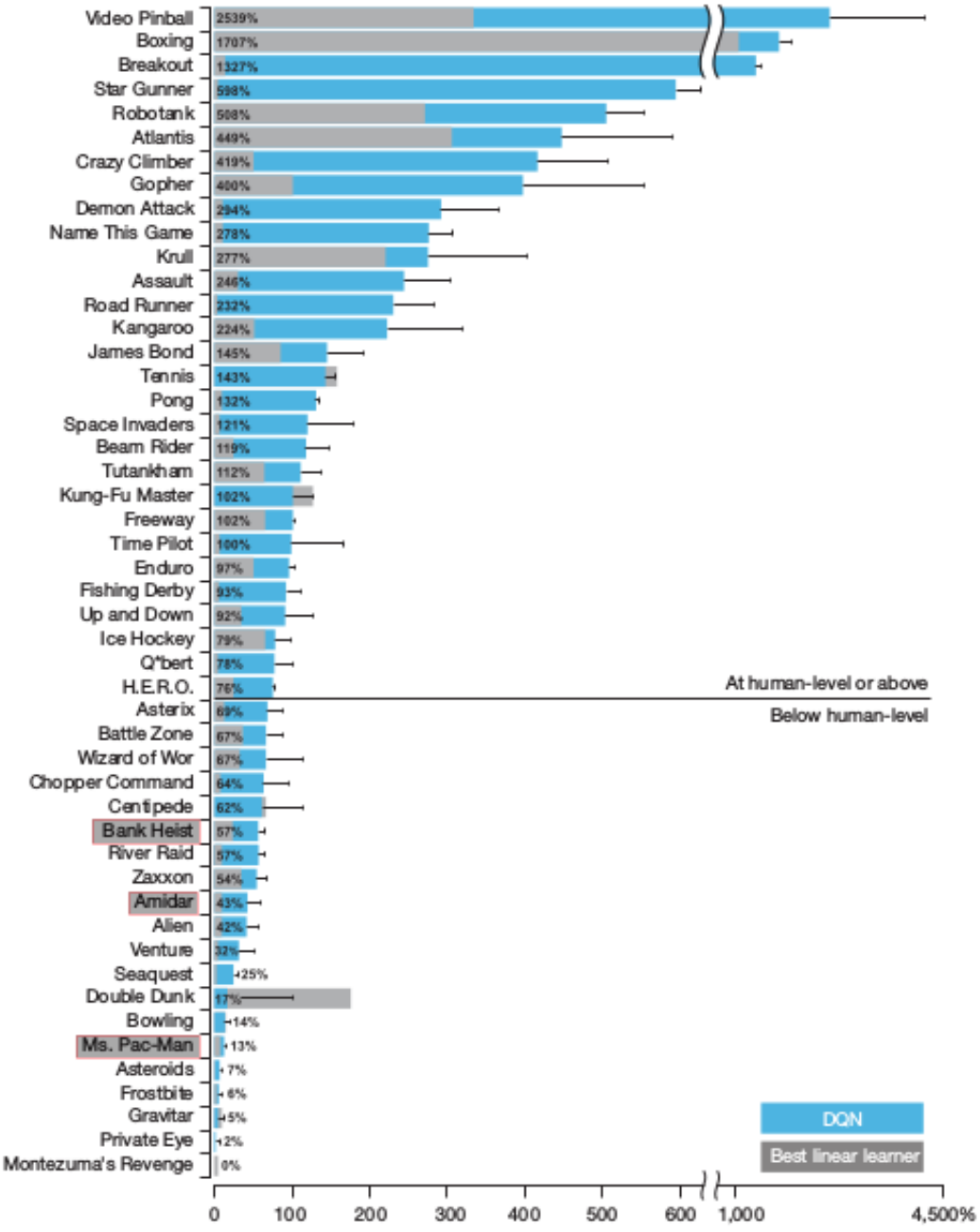

Figure 1: DQN results

4. ALGORITHM

4.1 Ms. Pac-man

The game Ms. Pac-man has 4 ghosts and Ms. Pac-man in a maze setting (Figure 2). The player earns points by eating pellets and avoiding ghosts. Contact with any of the ghost leads to the death of Ms. Pac-man. Two different types of pellets exists: Normal pellet and Power pellet. Eating the power pellet causes the ghosts to turn blue with fright for a specific amount of time during which Ms. Pac-man is able to eat the ghosts and score bonus points. Bonus fruits also appear in the game which when eaten increase the score of the game. The reward for eating a Normal Pellet is 10 and for Power Pellet the reward is 50. In the powered mode, eating ghosts consecutively yields reward of 200, 400, 800 and 1600 respectively. The detailed gameplay can be found at the Ms. Pac-man AtariAge Manual.

The algorithm 1 employed to play the game is a simple rule based agent (Escape Agent) that tries to escape from the ghosts at every step in normal mode and tries to eat the ghosts in powered mode. At each position, Ms. Pac-man calculates all the possible moves it can take among UP, DOWN, LEFT and RIGHT. It calculates the cost of each possible next location and chooses to go to the location with

Figure 2: Ms. Pac-man game screen

Algorithm 1 Escape Agent

```
function EscapeAgent(location, gameMode)
    dirArray ← possibleMotion(location)        ▷ possibleMotion function returns the all the locations Ms. Pac-man can reach by taking an action
    Initialize bestCost and bestDir appropriately
    for each dir in dirArray do
        nextLoc ← getNextLoc(location, dir)    ▷ getNextLoc function returns the next location of Ms. Pac-man given the current location and the direction of motion
        if gameMode is Normal then
            dirCost ← cost_normal(nextLoc)
        else if gameMode is Powered then
            dirCost ← cost_powered(nextLoc)
        if dirCost < bestCost then
            bestCost ← dirCost
            bestDir ← dir
    return bestDir
```

least cost. In normal game mode, cost at a location is given by the following equation:

$$cost_{normal}(location) = \sum_{i=1}^{4} \frac{\alpha}{dist(ghost_i, location)} + \sum_{i=1}^{P} \frac{\beta}{dist(pellet_i, location)} + \sum_{i=1}^{4} \frac{\gamma * \mathbb{1}_{corner}(corner_i, location)}{dist(corner_i, location} + \sum_{i=1}^{4} \frac{\delta}{dist(tele_i, location)} \quad (1)$$

where *location* denotes the location of Ms. Pac-man, P is the total number of pellets left in the game. $ghost_i$ is the location of ith ghost in the game screen, $pellet_i$ is the location of ith pellet. If the cost function included just the ghosts and pellets, Ms. Pac-man will be stuck in the corner of the maze most of the time. This is due to the fact that the corner points become local minima, w.r.t. the cost function above, over a significant period of time and reaching these points make Ms. Pac-man stuck at the position. Hence the corner costs are introduced to solve this problem. The maze is divided into 4 quadrants with one corner in each of them. $\mathbb{1}_{corner}(loc1, loc2)$ is an indicator function that gets activated if both the locations belong to the same quadrant. It is also possible for Ms. Pac-man to oscillate between two connected teleporation points (warp tunnels) as the tunnel end points are located on the edges of the maze. Teleporation costs have been added to avoid this. The $dist(loc1, loc2)$ function gives the estimated distance between the two locations.

In the powered mode, cost at a location is changed to allow Ms. Pac-man to move towards the ghosts as opposed to moving away from them. The cost function is given by

$$cost_{powered}(location) = \sum_{i=1}^{N} \frac{\lambda}{dist(edible\_ghost_i, location)} \quad (2)$$

where N is the number of edible ghosts present in the game at that instant. Powered mode does not need corner and teleportation costs as Ms. Pac-man moves towards the edible ghosts which do not lead to a point with local minima over a period of time as the ghosts are constantly in motion and eating an edible ghost make it go back to the normal mode. The parameters α, β, γ, δ and λ need to be tuned to get better performance.

Improvements Possible

When the ghosts are in fright mode, consecutive eating of the ghosts lead to exponential rewards. Targeting individual ghosts will yield better reward than the current cost policy. Bonus fruits are not targeted in our current algorithm.

Figure 3: Bank Heist game screen

4.2 Bank Heist

The user operates a Getaway Car and the objective of the game is to rob as many banks as possible before getting out of the city. The gas tank at the top left of the screen shows the amount of fuel remaining. After each robbery, one Cop Car appears at the location of the bank and will start to chase you. For example, in the figure 3, we can see that two banks are already robbed and a cop car(blue) is also present. Three of the passageways located at the edges of the screen, are "wrap-around" passageways. For example, if you exit through the lower left passageway, your car will reappear in the lower right passageway and vice versa. The upper right passageway leads to the next town. Robbing the first bank gives a score of 10, while the second one gives 20 and the third one gives 30. In each city, the Getaway Car starts at the upper left passageway. A complete explanation of this game can be found at the AtariAge Manual.

The algorithm employed is a simple target-based agent that tries to reach assigned targets if no cop car is near it. We have a parameter λ which denotes the minimum permitted distance between our car and the nearest cop car. If any cop car is at a distance less than λ, we go into the Escape mode. In the Escape mode, we try to escape from the cop cars by moving in a direction which maximizes the distance between the Getaway car and cop cars. We return to the target-based mode when this distance reaches above λ. All the distances are calculated using Breadth first Search in order to be accurate. In each city there are four targets - 3 banks and an exit point. The targets are ordered according to the distance from the exit point(Exit point being the last target). Hence, the bank farthest from the exit point is assigned as the first target for the Getaway Car. This strategy helps to escape quickly and safely(hopefully) from the city after we rob the third bank.

Processing the image to get the locations of cars was not easy. Finding the right movement to change the direction of Getaway car at a junction point is difficult because our agent seems to have inertia with which it becomes tougher to do this. Even without the inertia, it is difficult to understand this right position because at times the car turns at locations before the actual junction point. Because of these reasons, we find occasional hiccups in the game.

Improvements possible

- We do not use the amount of fuel left(shown in the top bar) in our algorithm.
- Note that in the current algorithm, our car leaves the city only after robbing all the banks. This strategy is not effective because it might be necessary to leave the city especially in situations where we are low on fuel.

4.3 Amidar

The game Amidar gives the user control of a gorilla in a maze setting (Figure 4) and is chased by 5 warriors. The gorilla can paint the maze and score points. The objective of the game is to make the gorilla paint as much as possible by avoiding the warriors. The maze is a set of rectangular boxes that are made up of edges. An edge in the maze is considered to be a horizontal or vertical line in which only the starting and end points are the corners. The interior of a box is empty which can be colored by the gorilla. From Figure 4 we can see that vertical edges are of equal lengths where as horizontal edges length can vary. An edge is considered to be painted if the gorilla moves from one corner of the edge to the other and when the gorilla reaches the end corner, the the edge turns blue. Painting a vertical edge yields a reward of 1 and reward for painting a horizontal edge can vary from 1 (shortest edge) to 6 (longest edge). Painting all the edges of a box turns the interior of the box blue and the box is considered to be painted. This yields a reward of 50. Painting all 4 corner boxes turns the warriors into chickens which can be caught by the gorilla and score 100 points. The next level of the game replaces gorilla with a paint roller and warriors with pigs. These two maze settings alternate as the game progresses. The detailed gameplay for can be found at Amidar AtariAge manual.

Even though the game is almost similar to that of Ms. Pac-man Escape agent cannot be used to play the game as such an agent cannot complete painting many boxes. Another reason that the Escape agent does not work for the game is that the speed of warriors is greater than the speed of the gorilla. When Escape agent is used the gorilla spends most of it's time on the border of the maze, as the cost is least on these points, which is patrolled by a warrior. 4 out of the 5 warriors when moving turn at the next corner they

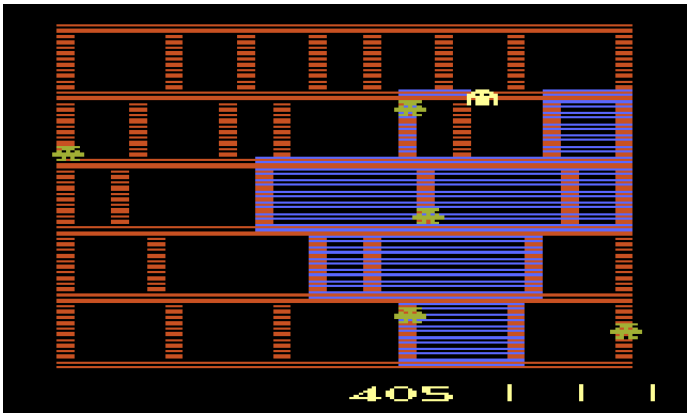

Figure 4: Amidar game screen

Algorithm 2 Target Based Agent (Amidar)

targetStack stores the targets gorilla needs to visit

function TARGETAGENT(gorillaLoc, warriorLocations)
  if dist between nearest warrior and gorilla $< \alpha$ **then**
    return Escape(gorillaLoc, warriorLocations)
  else
    return Target(gorillaLoc)

function ESCAPE(gorillaLoc, warriorLocations)
  Empty *targetStack*
  $dirArray \leftarrow possibleMotion(gorillaLoc)$ ▷ *possibleMotion* function returns the all the locations gorilla can reach by taking an action
  Initialize *bestCost* and *bestDir* appropriately
  for each *dir* **in** *dirArray* **do**
    $nextLoc \leftarrow getNextLoc(gorillaLoc, dir)$ ▷ *getNextLoc* function returns the next location of the gorilla given the current location and the direction of motion
    $dirCost \leftarrow cost(nextLoc, warriorLocations)$
    if $dirCost < bestCost$ **then**
      $bestCost \leftarrow dirCost$
      $bestDir \leftarrow dir$
  return *bestDir*

function TARGET(gorillaLoc)
  if gorilla reached *targeStack.top*() **then**
    targetStack.pop()
  if *targetStack* is empty **then**
    $edge \leftarrow nearestUnpaintedEdge(gorillaLoc)$ ▷ *nearestUnpaintedEdge* returns the corners of the unpainted edge nearest to the gorilla
    $targetBox \leftarrow unpaintedbox(edge)$ ▷ *unpaintedSquare* function returns all four corners of a box in the maze that the edge is part of
    Add the *targetBox* corners to the *targetStack*
  $targetLoc \leftarrow targetStack.top()$
  $dirArray \leftarrow possibleMotion(gorillaLoc)$
  return *dir* from *dirArray* that takes gorilla nearest to *targetLoc*

encounter. The remaining warrior is known as the tracer which patrols the border of the maze only. This warrior catches the gorilla within a few steps. The new algorithm that is employed to play the game is a target-based agent 2 that finds the nearest unpainted edge and tries to paint a box that contains the edge. When the gorilla is in the vicinity of a warrior, the agent shifts to an escape mode similar to that used in Ms. Pac-man game and tries to evade the warriors. This is determined by a parameter α which indicates the minimum safe distance from the nearest warrior when the gorilla is painting boxes. In the escape mode only ghosts that are in the vicinity of the gorilla are considered to take an action. The cost function used in escape mode is the following:

$$\begin{aligned} cost(loc, warriorLocs) = &\sum_{i=1}^{4} \frac{\gamma * \mathbb{1}_{warrior}(warrior_i, loc)}{dist1(warrior_i, loc)} \\ &+ \frac{\gamma * \mathbb{1}_{warrior}(tracer, loc)}{dist2(tracer, loc)} \\ &+ \sum_{i=1}^{4} \frac{\delta * \mathbb{1}_{corner}(corner_i, loc)}{dist(corner_i, loc)} \end{aligned} \tag{3}$$

where *loc* denotes the location of gorilla and *warriorLocs* contains the locations of the warriors with the tracer location identified. The function dist1 uses the Euclidean measure of distance as when followed by a warrior that is not a tracer, the gorilla when crossing a corner shouldn't change it's direction as the warrior will definitely change the direction of motion. Where as in the case of tracer using dist1 will make the warrior chase the gorilla until being caught as the tracer doesn't turn at the corner and the tracer moves faster than the gorilla. Using Manhattan distance - dist2 will enable the gorilla to take turns at the corners and get the warrior off it's tail. The cost of a warrior is only taken into consideration if it is in the vicinity of gorilla. This is determined by the parameter β which activates the indicator function $\mathbb{1}_{warrior}(warriorLoc, loc)$ if *warriorLoc* is located within β distance of *loc*. The corner indicator function is same as the one used in the Escape Agent for Ms. Pac-man. The parameters α, β, γ and δ need to be tuned for better performance.

Table 1: Ms. Pac-man score comparison

Algorithm	Mean	SD	Episodes	SE
DQN	2311	525	30	95.85
Escape Agent	5391.81	2043.91	100	204.39
Escape Agent tuned with CMAES	7706.38	2730.88	100	273.09

Table 2: Parameter Tuning for Ms. Pac-man Escape Agent

Algorithm	Ghost Cost(α)	Pellet Cost(β)	Corner Cost(γ)	Teleportation Cost(δ)	Edible Ghost Cost(λ)
Escape Agent	5000	-100	1000	1000	-5000
CMAES EA	5000	-534.015	1197.515	314.72	-5101.625

5. RESULTS

Standard Error of the Mean (SEM)

If X_1, X_2, ..., X_n are n independent observations from a sample with mean μ and standard deviation σ, then the variance of $T = X_1 + X_2 + ... + X_n$ is $n\sigma^2$. The standard error of the mean i.e. the standard deviation of the sample mean's estimate of the population mean is given by:

$$SE_{\bar{x}} = \sigma\left(\bar{x} = \frac{T}{n}\right) = \sqrt{\frac{n\sigma^2}{n^2}} = \frac{\sigma}{\sqrt{n}} \tag{4}$$

Comparing the means of two samples

Let $\bar{X_1}$, $\bar{X_2}$, SE_1, SE_2 be the estimates of sample mean and standard errors of two independent random samples. The distribution of The standard error of the difference between the independent means is:

$$SE(\mu_1 - \mu_2) = \sqrt{SE_1^2 + SE_2^2} \tag{5}$$

The random variable μ_1 - μ_2 is normally distributed wtih mean $\bar{X_1}$ - $\bar{X_2}$ and standard deviation $SE(\mu_1 - \mu_2)$. From this we can compute the probability that $\mu_1 > \mu_2$ by the following way:

$$\begin{aligned} P(\mu_1 - \mu_2 > 0) &= P\left(Z > \frac{0 - (\bar{X_1} - \bar{X_2})}{SE(\mu_1 - \mu_2)}\right) \\ &= P\left(Z > \frac{\bar{X_2} - \bar{X_1}}{SE(\mu_1 - \mu_2)}\right) \end{aligned} \tag{6}$$

where Z is a standard normal distribution.

5.1 Ms. Pac-man

The performance of DQN [8] and Escape Agent 1 are presented in Table 1. From Equation 6 the probability of Escpae Agent outperforming DQN agent is ≈ 1. Further when Covariance Matrix Adaptation Evolution Strategy (CMAES) [3] is applied to the tune the parameters for better performance, the performance of the agent increased by 2000. The parameters used for both the escape agents are listed in Table 2. The population size of each generation used in CMAES is 16. The evaluation of a parameter set is done by simulating 100 episodes for better estimates of the sample mean and standard deviation. As the parameter were initially tuned the CMAES algorithm converges within few iterations which can be seen in Figure 5. The algorithm used for optimization is active CMAES and parameters are optimized relative to the value of α (which is kept constant for all the evaluations).

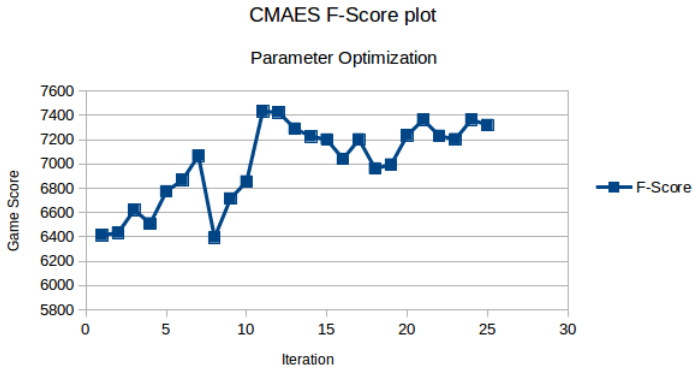

Figure 5: CMAES F-Score plot for Escape Agent

5.2 Bank Heist

The performance of Target Agent for Bank Heist and that of DQN agent are presented in Table 3. Using Equation 6 the probability of Target Agent outperforming DQN agent is ≈ 0.95922 which gives us a confidence level of 95.922%. The average score can be further improved by more sophisticated agent which includes the improvements mentioned before.

Table 3: Bank Heist score comparison

Algorithm	Mean	SD	Episodes	SE
DQN	429.7	650	30	118.67
Target Agent	636.9	83.4	100	8.34

5.3 Amidar

In this case, the performance of Target Agent is comparable with the performance of DQN agent which can be seen from Table 4. The Target Agent outperforms DQN agent with probability ≈ 0.71316 which gives us a confidence level of 71.316%. Due to high standard deviation of the score of DQN agent, the confidence level is less even though the average performance of Target Agent is better than the DQN agent.

Table 4: Amidar score comparison

Algorithm	Mean	SD	Episodes	SE
DQN	739.5	3024	30	552.10
Target Agent	1050.61	310.70	103	30.61

6. CONCLUSIONS AND FUTURE WORK

Results show that simple target based or escape agents outperform the DQN agent in all three Maze games. We have used CMAES to optimize the parameters in Algorithm 1. Similar optimization can lead to better results in Amidar where the confidence level is low compared to other games. An extension to this project could be to train a Neural Network with architecture similar to the DQN's NN. This could prove that DQN's NN is not reaching the global optimum or otherwise. To establish such a result it is also important to test on a diverse set of Atari games.

We could also use these basic algorithms to quickly train a NN using Supervised Learning methods and then use Reinforcement Learning techniques as been done in recent work such as AlphaGo [9].

7. REFERENCES

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

[2] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346, 2014.

[3] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[4] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. Hyperneat-ggp: A hyperneat-based atari general game player. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 217–224. ACM, 2012.

[5] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(4):355–366, 2014.

[6] N. Ikehata and T. Ito. Monte-carlo tree search in ms. pac-man. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 39–46. IEEE, 2011.

[7] T. Miyama, A. Yamada, Y. Okunishi, T. Ashida, and R. Thawonmas. Ice pambush 5. *publicado em*, 2011.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[10] B. Tanner and A. White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.

[11] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.