GAME AI REPORT

Ahmad Alzeitoun - MR.Nr 2413759

University of Bonn, Germany

ABSTRACT

In this report, we discuss in details some methods for Game AI that are widely used. First, we provide a brief demonstration about these methods. Some probabilistic and heuristic agents were implemented to play TicTacToe and Connect Four games. Then we explain strategies for turn based games and present methods for path planning and behavioural programming.

Index Terms— Game AI, Turn-based games, Path planning, Behavioural programming, Heuristic strategy, Probabilistic Strategy, Connect four, Breakout, Bayesian imitation, Dijkstras algorithm, A algorithm, K- Means, TNN, RNN.

1. INTRODUCTION

In this report we introduce many scientific aspects with considering the subject area from GAME AI such as some techniques regarding strategies for TicTacToe game, then we show the implementations of functions of Probabilistic strategy and Heuristic strategy. Also, we learn the game mechanics for Connect Four on a 6 X 7 board,breakout game and take a closer look at TicTacToe game tree. Furthermore, we learn combinatorial arguments in order to compute an upper bound of TicTacToe game tree size. In next section, we explain the Min-max and Min-max search for connect four. The path planning is demonstrated while implementing Dijkstras algorithm and A algorithm to plan a path. Also, we discuss a fuzzy controller for the breakout game behaviors during it's implementation. Finnally we go through Bayesian imitation learning and self organizing maps to represent player movements.

2. SIMPLE STRATEGIES FOR TURN-BASED GAMES

In this section, we used three strategies to implement an intelligent agent capable of playing TicTacToe game which are Probabilistic, Heuristic and Neural Network strategies.

2.1. Probabilistic Strategy

In this strategy, we used two random agents to play 1000 game and store the winning agent positions after each tournament. Using these stored positions, we implemented a function to calculate the position with the highest probability to win (the best move to play), Then used this function to move player X against random player O. Fig(1) is a generated heat map that shows the winning probability for each position on TicTacToe board. All the probabilities are normalized to the range [0,1] using min-max normalization. Notice that the position with the highest probability to win is at the center of the board. This strategy was very simple and did not give us satisfactory results.

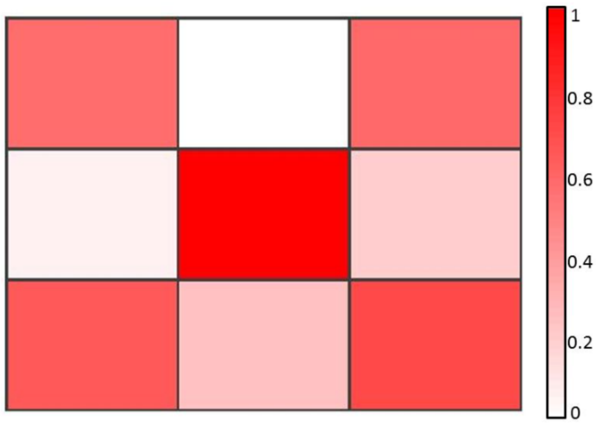


Fig. 1. The distribution of the winning probabilities on TicTacToe board

2.2. Heuristic Strategy

In this section, we used a heuristic algorithm to replace the probabilistic agent. To find the best movement the algorithm checks the following conditions.

1. Can the agent win or not?
2. Can the agent block the opponent or not?
3. Can the agent create a fork or not?
4. Can the agent block opponent fork or not?
5. Can the agent play in the centre of the board or not?
6. Can the agent play in opposite corner or not?

7. Can the agent play in a corner or not?

8. Can the agent play in any middle square of the four sides of the board or not?

Our implementation of TicTacToe Heuristic strategy (Max Result) is based on Algrithm(1) where some moves can lead to end the game in a draw. Fig(2) and (3) show the required moves to end the game and show Minmax Win-Draw Histogram.

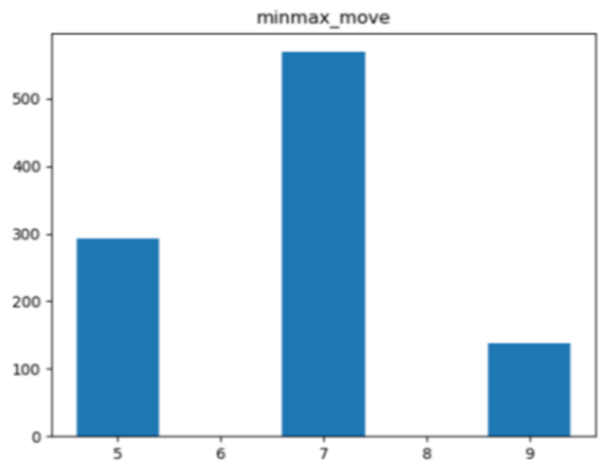


Fig. 2. Moves required to end the game

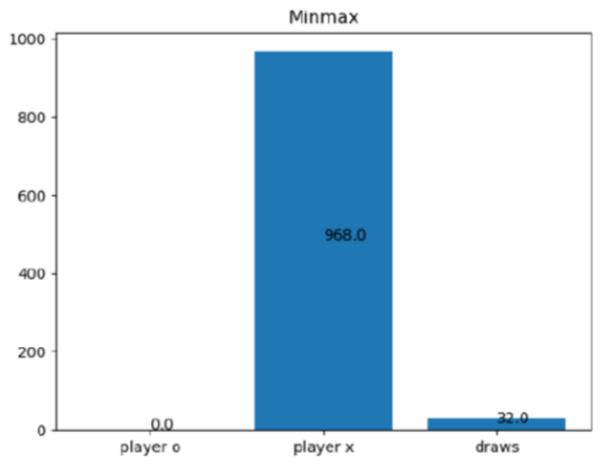


Fig. 3. Minmax Win-Draw Histogram

It is important to know that using this heuristic algorithm to play TicTacToe will make the agent X always a winner or at least the game ends with a draw.

Algorithm 1 Maxres(state, player, depth)

```
result = $\emptyset$
$\gamma$ = 0.1
depth-=1
for each possible new State with move M do
   $if$ newState is winning move $then$
      result.append(reward)
   $else$
      $if$ move still possible $then$
         result.append( maxres(newState, player 1,depth)
+ depth)
      $else$
         result.append(0)
      end if
   end if
end for
return result
```

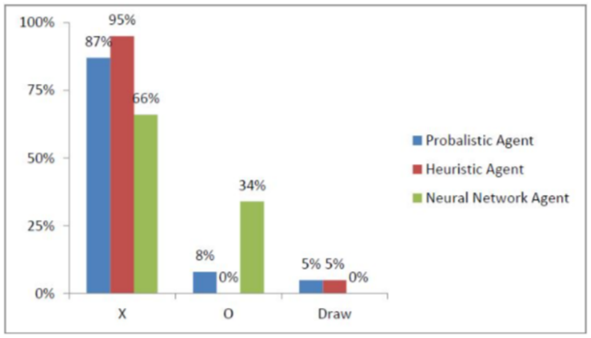


Fig. 4. Heuristic, Neural Networks and Probabilistic results when playing TicTacToe

2.3. Neural Network

Although the heuristic agent was unbeatable, we were very curious to find how neural networks could solve this game. The type of neural network we used in this part is a Multi-layer Perceptron (MLP). The Network contain 19 input neuron, half of them represent the positions from the first player point of view, while the second half represent the positions from the opponent point of view. The output layer contains 9 neurons that represent the best position to play. Fig.4 shows the network structure. The training dataset contains vectors of the players positions where we set the training data to 0 when the first player is playing and 1 when the opponent is playing,forexample,thevectorV =[0,1,0,1,,,,,]means that player O played at position (1, 1) and (1, 3), player X played at position (1, 2) and (2, 1) and the other positions on the board are empty. Using the heuristic algorithm was much better than using neural networks.

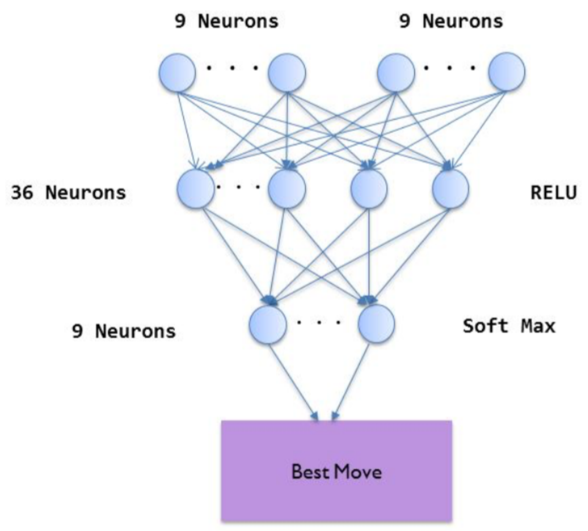


Fig. 5. Neural Network Strategy

Figure (5) shows a comparision between previous three strategies (Probabilistic, Heuristic and Neural Network), as we can see how much better the heuristic algorithm was than using neural networks and probabilistic strategies.

2.4. Connect Four

We implemented an AI agent capable of playing Connect Four by following a probabilistic strategy; this strategy also depends on storing the winning agent positions after playing 1000 tournament just like the one we mentioned in section 2.1. However, this strategy might not work with connect four because choosing the position with the highest probabil- ity can break the rules of the game. Fig.4 is a generated heat map that shows the winning probability for each position on 6X7 Connect Four board. All the probabilities are normalized to the range [0, 1] using min-max normalization. Notice that the highest probability is at position (1, 3) and according to the game rules the piece should be placed at the lowest position in the column. We can solve this issue in a naive way by implementing a function that checks if the position with the highest probability is valid, if it is not then it will search for the next valid position with the highest probability, in our example, it will be (6, 3) or (6, 5) (assuming that the game board is empty).

The win rate based on token placement is shown in Figure(6). The good statistics fields don't work well over multiple games, the random moves are not optimal as they don't lead to direct wins.

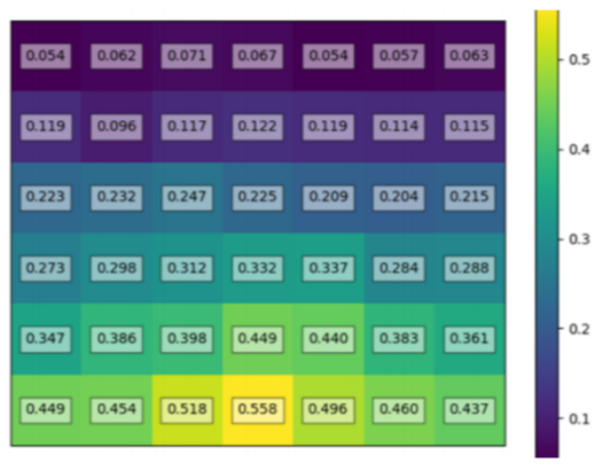


Fig. 6. Win rate based on token placement

3. GAME TREES AND PATH PLANNING

3.1. The TicTacToe Game Tree

In this section, The game tree explores all possible moves of a game scenario by starting at the initial position and making each game position a tree node. The problem with game trees is that they grow incredibly quickly. We chose the simple game of TicTacToe in order to show how big the game tree can grow. The number of game states in TicTacToe is 3^9, but the upper bound of tree size is 9! (Because there are 9 positions for the first move, eight for the second and so on). Observe that the game tree size is bigger than the game states and this is be- cause one game state can be reached in several ways, Fig.5 display an example of this idea. Fig.6 shows all TicTacToe game tree calculations. Figure(7) shows the idea of What is attracting are more detailed account of TicTacToe game tree calculations that can be learned here.

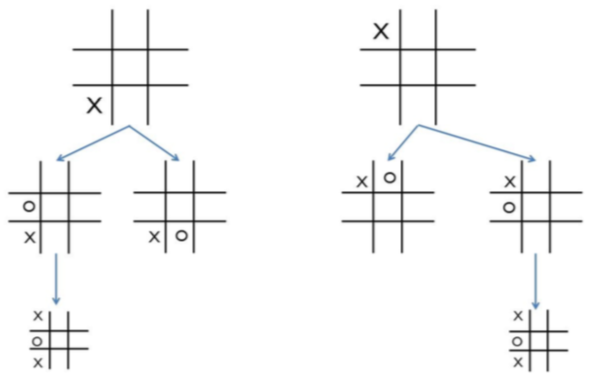


Fig. 7. TicTacToe game tree example

3.2. MinMax Computations

In this section, we demonostrate on MinMax algorithm which is a decision rule for playing against an effective opponent. Basically this algorithm depend on minimizing maximum loss, in other words a MinMax value $\boldsymbol{mmv}$ is assigned to every game state that can result from a node n and move to a state for which mmv is the largest. The distribution of the winning probabilities on Connect Four board is shown below in Fig.8.

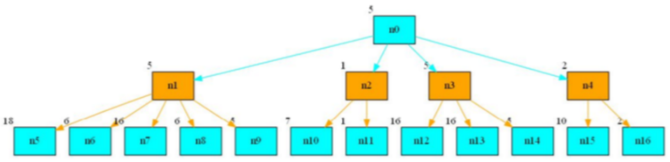


Fig. 8. The distribution of the winning probabilities on Connect Four board

At the begining we have to find a way to choose a better alternatives since we have ties, then we perform a MinMax searching from given tree, by performing $\boldsymbol{mmv}(\boldsymbol{n0})$ will result moving to state $\boldsymbol{n3}$, then for $\boldsymbol{mmv}(\boldsymbol{n3})$ from player-orange perspective will results moving to stat n13. Basically we have to find a way to choose a better alternatives because of ties. Therfore we introduce two kind of scenario, the **first scenario** we already know that the distribution of **player-orange** will pick it's move:

- It is better to break the ties by calculating the expected value of the next possible states
- For a random move

$$E(n1.children) = \frac{18+6+16+6+5}{5} = 10.2 \quad (1)$$

$$E(n3.children) = \frac{16+16+5}{3} = 12.3 \quad (2)$$

- This gives us a closer look that we have a higher chance of getting higher outcome by going to $\boldsymbol{n3}$.

The **second scenario** the *player-orange* will pick it's move as optimal as player-blue which means:

- There is no use to look at the better alternatives
- *Player-orange* will choose the minimum outcome anyway
- $\boldsymbol{n1}$ and $\boldsymbol{n3}$ both give us equal score, the next state minimum score will be the same

3.3. MinMax Search for Connect Four

We used a depth MinMax algorithm with evaluation function which assigns evaluation and below is a description of it.

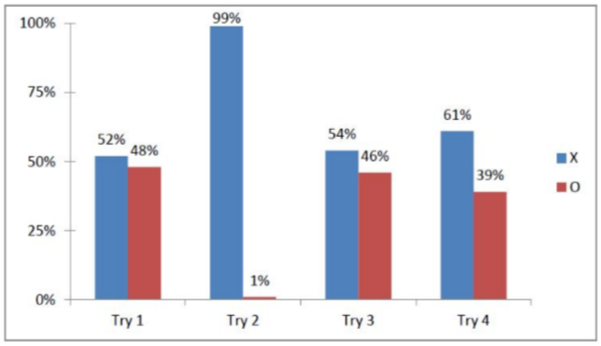
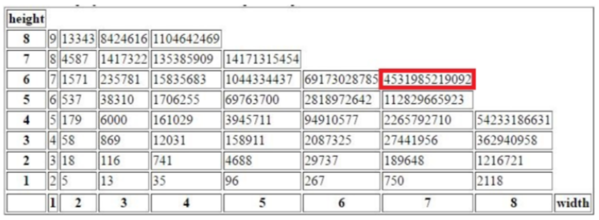


Fig. 9. The agents results when playing 6X7 Connect Four. InTry1(d1 = 1,d2 = 1),Try2(d1 = 2,d2 = 1),Try3 (d1 = 4, d2 = 3), Try 4 (d1 = 4, d2 = 4)

3.4. Connect Four using Recurrent Neural Network

As we discussed in section 3.2 the game tree can grow extremely quickly. Using MinMax algorithm to generate a game tree for all Connect Four board positions is impossible. The table in fig.10 shows the number of possible positions for 6x7 connect four board, to solve this issue, we use recurrent neural network to explain the sequence of playing between two players. First, we generate a dataset by making two players play 5000 round and track their choices, the position is positive if the player X is playing. We record the position of second player 0 as negative if it is playing, and all other positions that left will be recorded as 0.



height									
8	9	13343	8424616	1104642469					
7	8	4587	1417322	135385909	14171315454				
6	7	1571	235781	15835683	1044334437	69173028785	4531985219092		
5	6	537	38310	1706255	69763700	2818972642	112829665923		
4	5	179	6000	161029	3945711	94910577	2265792710	54233186631	
3	4	58	869	12031	158911	2087325	27441956	362940958	
2	3	18	116	741	4688	29737	189648	1216721	
1	2	5	13	35	96	267	750	2118	
	1	2	3	4	5	6	7	8	width

Fig. 10. Number of possibilities for a 6X7 Connect Four game [11]

We used RNN since it depends on time series we had to split the data to be time-based, this means that at time t, only part of the dataset is fed to the network fig.11. RNN agent results shown in Fig.12 illustrate when playing Connect Four, note that the player O is a random player. The percentage indicates how many tournaments the player won. As a conclusion, MinMax algorithm results are much better than the result of RNN which is not satisfactory.

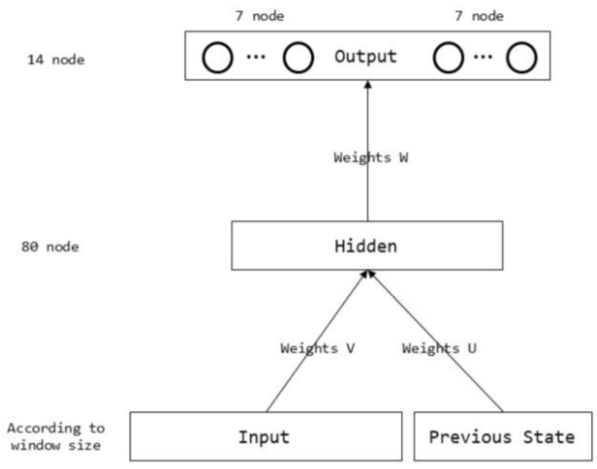


Fig. 11. RNN structure to solve Connect Four game

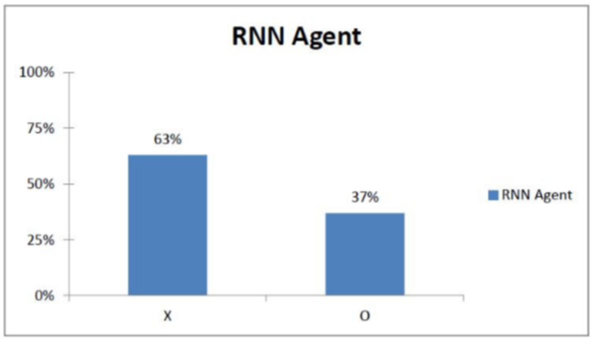


Fig. 12. RNN agent results when playing Connect Four

3.5. BreakoutController

In this section, we learn to implement a controller for the breakout game, our solution has a function that controls how the paddle has to be moved in order to hit the ball. We change the speed of the ball increases over time and observe how the paddle react to this change. Paddle Controller find out the positions x and y of the ball, then moves to the left or right. The paddle stops when it hits the ball and it moves again when the ball is coming down. changing the speed based on the distance. When including the speed acceleration we have 2 bricks otherwise 30 bricks please have a look at Fig.13.

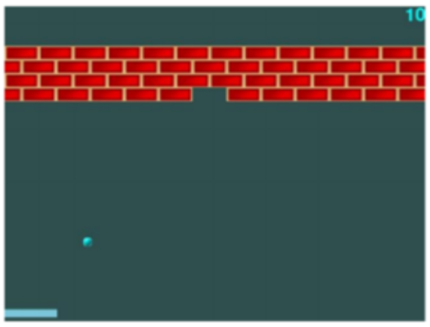


Fig. 13. Breakout-Paddle Controller

3.6. Path Planning

Path planning is a substantial part of Game AI. The problem here is to navigate the characters towards goals by using the shortest path and without hitting any obstacles. In this task, we rely on two datasets (2D game map).

We provided a grid graph has vertices only for zeros, then we used Dijkstra algorithm and A* algorithms to find out the path between the sart and the end vertices. A* is effective estimation function, it uses f [v] = g(v)+h(v) where g(v) as g(v) the past cost function and h(v) is the heuristic estimation of the distance to the goal. A* develop the next vertex according to the past cost function like Dijkstra, So A* is basically an informed variation of Dijkstra. h(v) is called admissible if it does not overestimate the costs of future paths and if h(v) is admissible then A* is optimal and complete [2]. Below in Fig.14 shows the shortest path between the starting point and the ending point.

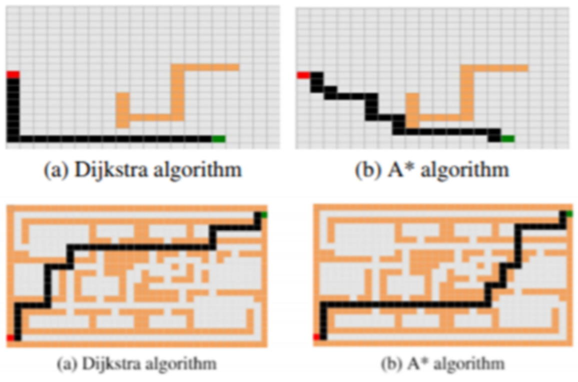


Fig. 14. The shortest path using both algorithms

4. BEHAVIOUR PROGRAMMING

4.1. Connect Four on a Large Board

In this section, we implement our code on 19X19 board and collected some statistical data to check the effect of using this size of board. For sure a bigger board's size make the game slower as a result of the big number of game states. We change the depth parameter in case of random agent Fig.15. To observe the result of two MinMax agents play together we modify the depth parameter See Fig.16.

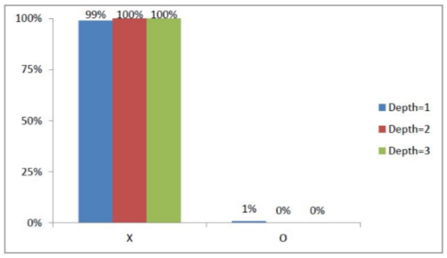


Fig. 15. Results when playing 19X19 Connect Four with different Depths

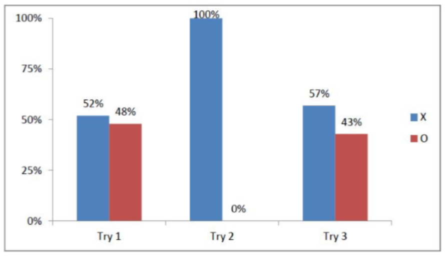


Fig. 16. Results when playing 19X19 Connect Four - two MinMax agents play together

4.2. Breakout Fuzzy Controller

Fuzzy Logic depends on the real number in the interval between 0 and 1, we demonostrate of developing the agent to play using fuzzy logic. Fuzzy system Fig.17 has two input parameters Paddle-ball and the speed of the ball, the output of this system is the speed of paddle. Therefore, the rules are as following, we define the speed [Fast, Middle-Speed, Slow, stop] and the distannce [Very close, Middle, Far, Faraway]. Fig.18Plott different cases where we can learn how membership is related to the speed and distance.

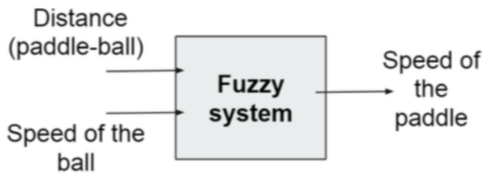


Fig. 17. Fuzzy system

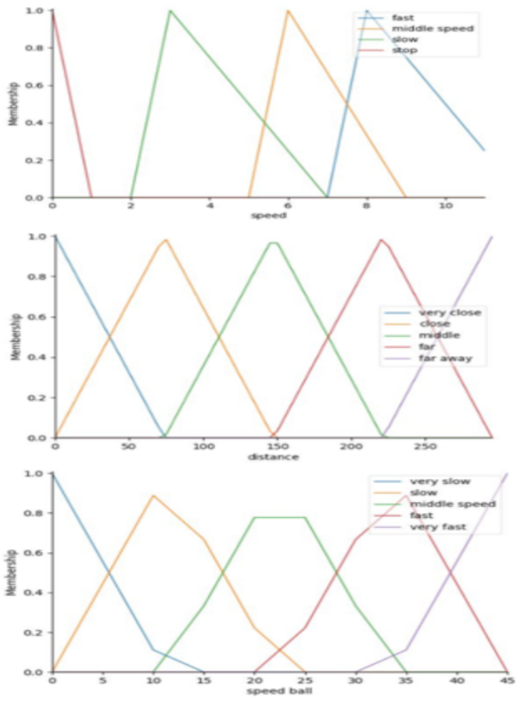


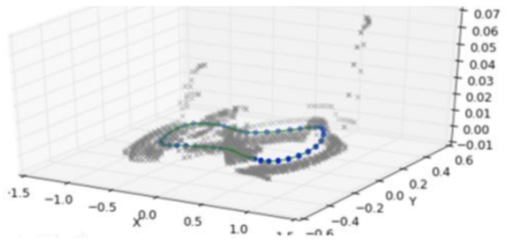
Fig. 18. relation between Membership and Speed and Distance

4.3. Self-Organizing Maps to represent player movements

In this task, we study Self-Organizing Maps (SOM) which are part of artificial neural networks that are trained to produce low-dimensional spaces from multi-dimensional data. We rely on graph theory which means that given a set of data.

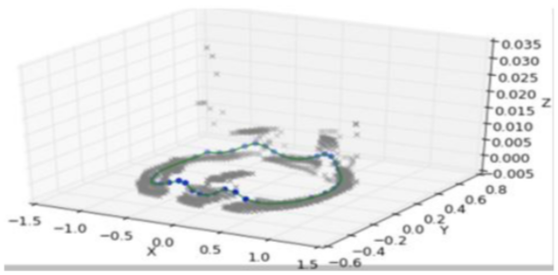
Given Dataset:$X = x_1, .., x_n \subset \mathbb{R}^m$. We fit a labeled graph G = (V, E, vloc) where: $E \subseteq VXV$ are SOK connections, $V = v_1, .., v_n$ are the SOM neurons and $V \rightarrow \mathbb{R}^m$ is the weight vector.

Our training dataset indicates a human players 3D position at time t on the Quake III map. The learning rate $\eta(t)$ and adaptation $\sigma(t)$ rate are designed to be time dependent which means that they will shrink over time. Fig.19 and 20 show our training results [9]. We used In Fig.19 more clusters K and more iterations because the player route is more complicated than the one in Fig.20.

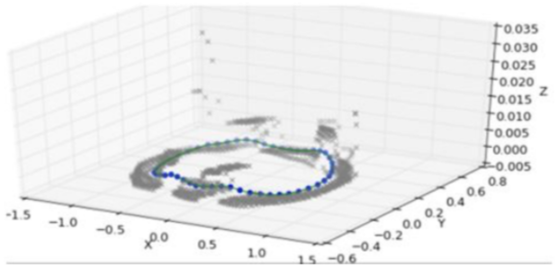


$K = 50, t_{max} = 10000$

Fig. 19. SOM training results



(a) $K = 45, t_{max} = 9000$



(b) $K = 50, t_{max} = 5000$

Fig. 20. SOM training results, for (a) and (b) we used data set1

4.3.1. Bayesian Imitation Learning

In this Task, we explain the importance of Bayesian imitation techniques for imitating successful action for a computer game characters. We implemented the al gorithm(2), we solved this issue [8].

Algorithm 2 Bayesian imitation learning algorithm

```
1:  Use SOM weights from section 4.3 as K prototypical game states.
2:  Define actions to be $\mathbf{a}_t$ = $\mathbf{x}_{t=1}$ - $\mathbf{x}_t$.
3:  Cluster action vectors into K prototypical action vectors us- ing K-Means.
4:  Assign all locations and actions to their respective state/action centers.
5:  Compute the joint probabilities p(si , rj ) of state si and ac- tion rj.
6:  $\boldsymbol{for}$ t = 0 to $\mathbf{t}_{max}$ $\boldsymbol{do}$
7:      Select a point $\mathbf{x}_t$ among the given data.
8:      Determine the closest $\mathbf{s}_i$ and select an action according
    to $\mathbf{a}_t = \text{argmax}\, p(\mathbf{r}_j|\mathbf{s}_i) = \text{argmax}\, \frac{p(\mathbf{s}_i,\mathbf{r}_j)}{p(\mathbf{s}_i)}$
9:      compute $\mathbf{x}_{t=1}$ = $\mathbf{a}_t$ + $\mathbf{x}_t$.
10:     end if
11: end for
```

Fig.21 Both graphs clarify the character imitation results for datasets [4] and [5]. The left graph plott how the trajectory results do not fit the data because of the number clustered actions is only 10, The coloured points in the middle are the actions, their color represents the assigned K-Means centroid. The purple data is the trajectory. k = 50,$\mathbf{t}_{max}$ = 10000, Action centroids = 10; Trajectory samples = 300. The right graph The connected blue dots are the SOM clusters. The coloured points in the middle are the actions, their color represents the assigned K-Means centroid. The purple data is the trajectory. k = 50,$\mathbf{t}_{max}$ = 10000, Action centroids = 50; Trajectory samples = 300.

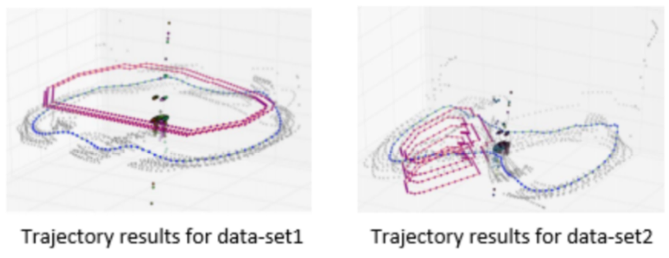


Fig. 21. Trajectory results for both datasets [4] [5]

In the middle are the actions which represented by coloured points, also represents the assigned K-Means centroid. The purple data is the trajectory. k = 50, $\mathbf{t}_{max}$ = 10000, action centroids = 50, trajectory samples = 300.

5. CONCLUSION

In this report, We have discussed different flavors of methodologies for solving multiple Game AI problems, through our implemntation we learned the probabilistic, NN and Heuristic strategies were used to build agents capable of playing turn based games such as TicTacToe and Connect Four. Additionally, we proofed that neural network is not an optimal solution in some topics from Game AI. Furthermore we demonostrated the breakout Fuzzy controller and the role of Bayesian imitation techniques for imitating successful actions for a computer game characters and reviewed Self-Organizing Maps. Finally, we have learned that chosen programming techniques has an influence for the robustness and the correctness of a program.

6. ACKNOWLEDGMENT

I am extremely thankful to **Prof. Dr. Ing. Christian Bauckhage**, without his advice, tips, guidance and support along with his positive attitude, we will might not be able to explore the field in a optimal angel [**?**].

7. REFERENCES

1 Prof.Dr-Ing.Christian Bauckhage. Behaviour programming, 2018. Lecture15-B-It Aachen Bonn.

2 Prof.Dr-Ing.Christian Bauckhage. Minmax algorithm, 2018. Lecture5-B-It Aachen Bonn.

3 John Tromp. Connect four playground, 2018, GAMEAI.

4 Prof.Dr-Ing.Christian Bauckhage. Data set training1, 2018. Projects GAMEAI.

5 Prof.Dr-Ing.Christian Bauckhage. Data set training2, 2018. Projects GAMEAI.