REPORT ON GAME AI PROJECTS

Rahimbeyli Sattar

University of of Bonn

ABSTRACT

Game development has always been a challenging field. Nowdays addition of the AI to this process makes it more breathtaking and simple. This field is a new and there are a lot of improvements and inventions can be performed. In this report we test basic techniques and continue complicating them, which will require more time and information.

***Index Terms*—** Breadth-first search,path planning, Dijkstra's algorithm, A* algorithm, self-organizing maps, Bayesian imitation learning.

1. INTRODUCTION

The three projects realized in the scope of Game AI and the following tasks were implemented:
• Project 1 : simple strategies for turn-based games
The first project is about turn-based games like tic tac toe, connect four and simple strategies for them: probablistic and heuristic and introduction to the arcade game breakout.
• Project 2 : game trees and path planning
Next step was to improve our turn-based games by finding optimal ways. Minmax algorithm helped us to find the minmax value of a given tree, Connect Four was improved by using depth-restricted search attachment to the minmax algorithm, Breakout's improved controller was also one part of this project and finally path planning .
• Project 3 : behaviour programming
Final task covers such fields as fuzzy-logic, Self organising maps and Bayesian imitation learning conducted the final line of this project.

2. SIMPLE STRATEGIES FOR TURN-BASED GAMES

Initially we were asked to implement 2 different games in order to realise further probabilistic and heuristic strategies for them. We can split this into 2 steps.
1. Gather statistics
2. Use statistics for probabilistic and heuristic strategies.

Thanks to XYZ agency for funding.

2.1. Tic tac toe game

Tic tac toe [1] is a turn-based board game. Essence of the game is to connect vertically, horizontally or diagonally 3 cells on 3x3 board. We represent the board as a 2-dimensional array where the values are initially assigned to zero. Each **Board[i,j]** equals to a cell.There are 3 terminal conditions:
1. **Player 1 wins**
2. **Player 2 wins**
3. **Draw**
And according to them, cells can have 3 values:
1 $\Rightarrow$ player 1
-1 $\Rightarrow$ player 2
0 $\Rightarrow$ nobody

Statistics is gathered after 1000 games in which both players uses random strategy as it shown on Figure 2.1:

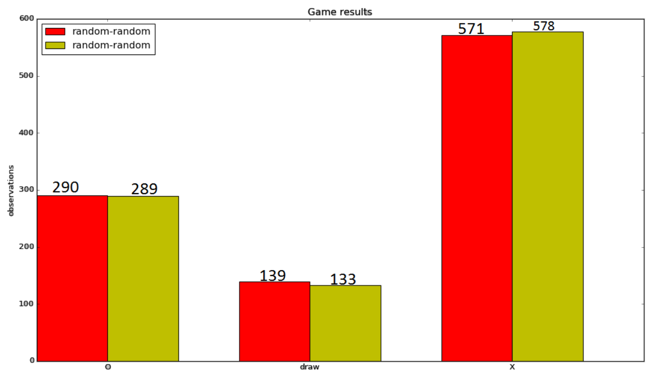

Figure 2.1.Random player - Random player

2.1.1. Probabilistic strategies

Probabilistic strategy is strategy based on statistics, realisation of this strategy requires backup data which is in our case is an array which contains the number of captured cells. Incrementation takes place only when Player X gets victory.

$$A = \begin{bmatrix} a_{11} & a_{12} & .. & a_{1n} \\ a_{21} & a_{22} & .. & a_{2n} \\ .. & .. & .. & .. \\ a_{n1} & a_{n2} & .. & a_{nn} \end{bmatrix} \Rightarrow a_{i,j} = c_{i,j}/N$$

where N = number of games, $c_{i,j}$- our incremented array. When the probabilstic array is ready we use the following formula to make a decision:

$$NextMove(i,j) = \arg\max_{x,y} a(x,y)$$

in other words the layer which has the highest incremention will most likely be the next move.

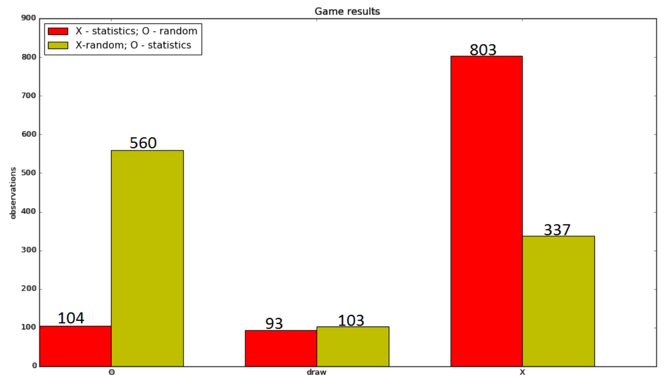

Figure 2.2.1.Probability player - Random player

Algorithm 1 Heuristic approach

```
1:  procedure HEURISTICMOVE(player,grid)
2:      ▷ We are assuming that layers is the set of 3 elements, connected diagonally, horizontally or vertically
3:      ▷ Cell is the neighbor of connected layers
4:      ▷ Attack mode
5:      for all rows, columns and diagonals in grid do
6:          if $grid[layers] == player$ then
7:              if $grid[cell] == 0$ then
8:                  $grid[cell] == player$
9:                  return
10:     ▷ Defence mode
11:     for all rows, columns and diagonals in grid do
12:         if $grid[layers] == -player$ then
13:             if $grid[cell] == 0$ then
14:                 $grid[cell] == player$
15:                 return
16:     if neutral then
17:         $grid[randomLayer] == player$
18:         return
```

2.1.2. *Heuristic strategies*

Heuristic strategy is a set of rules based on different conditions. It shows better performance because it includes into itself every condition which can happen during the game. Algorithm 1 shows this strategy and results of this approach can be seen on Figure 2.2.2 :

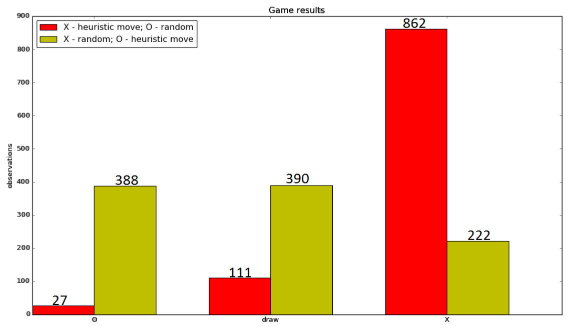

Figure 2.2.2.Heuristic player - Random player

2.2. Connect four

Connect four [2] is the extended version of Tic Tac Toe, with some changes:
1. Board size 6x7.
2. Instead of 3 layers, termination state is reached at 4 layers.

2.2.1. *Heuristic strategy*

The *" **Good Move** "* is the move which considers 1 step forward movement. In other words, it checks whether you are going to loose or win in the next move. Algorithm 1 shows this in detail. But we can go actually deeper, instead of warning or hinting about the terminal move, algorithm can be extended by predicting the layers of size 2 which can grow into 3, for both players. These hints are called *" **Good Move** "*, because you are warned before and you can control the game. Random player has no chance.

Defence shows the defence mode:

To defence put your coin to layer 5

Attack Shows the attack mode:

To attack put your coin to layer 2

Statistics after 1000 games with heuristic strategy shown on Figure 2.2.3 Results

3. GAME TREES AND PATH PLANNING

This project contains 4 tasks which involves more deeper strategies of improvement our 2 board games, one arcade game and path finding. At first we have to compute game tree and look for the properties of it. It is needed to find the upper

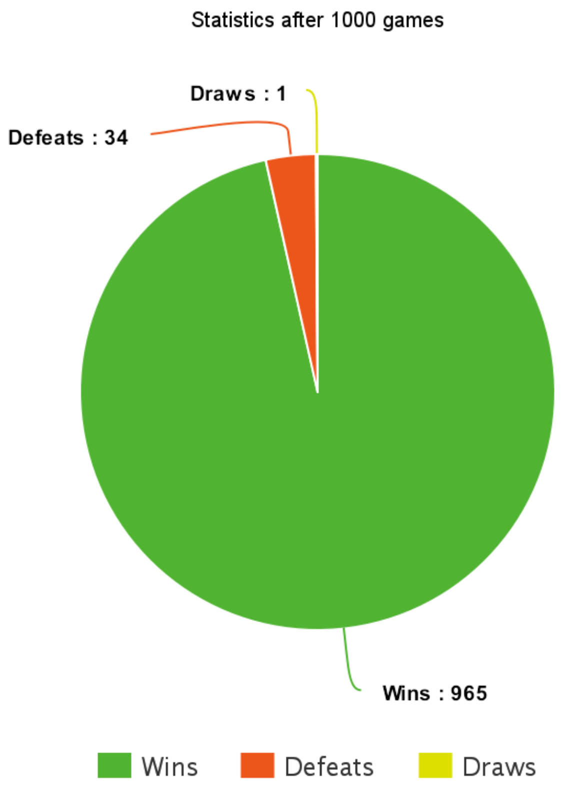

Figure 2.2.3.Results

bound of the nodes for tic tac toe. When the game tree is settled next is to use minmax search algorithm for finding the minmax value of a given tree. Minmax algorithm with depth restricted search is also used to improve the performance of Connect Four, next issue is to implement a contoller for an arcade game Breakout and finally we are facing path finding problem and its appropriate algorithms are Dijkstra's algorithm and A* algorithm.

3.1. Tic Tac Toe GAME TREE

Our goal to calculate the upper bound of nodes for Tic Tac Toe game. Initially we have 9 possible moves and with every level the number of possible movements is decreasing by 1. If we use naive approach, the number of nodes would be 9! = 362880 , it contradicts with situations where game reached terminal state, but the movement is still continues as well and the fact that the number of all possible game states, which is in our case equal to $3^9 = 19683$. To calculate the upper bound we start with the empty board and continue until we don't reach termination criteria, in our case 3 elements in row. Result of this algorithm is

$Number\ of\ nodes = 269173$
$Victories = 55872$
$Draws = 95166$

$Branching\ factor = 1.728149 = \frac{nodes}{nonterminalnodes}$

Next aim is to calculate the minmax value for a given node **n** on the figure 3.1.1, using formula[3] shown on graph 3.1.2

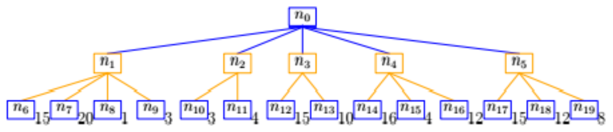

Figure 3.1.1 : Tree

$$mmv(n) = \begin{cases} u(n) & \text{if } n \text{ is a terminal} \\ \max\limits_{s \in Succ(n)} mmv(s) & \text{if } n \text{ is a } MAX \text{ node} \\ \min\limits_{s \in Succ(n)} mmv(s) & \text{if } n \text{ is a } MIN \text{ node} \end{cases}$$

Figure 3.1.2 : Minmax solution

After performing the calculation we achived the following results: n_1 = 1 , n_2 = 3 , n_3 = 10 , n_4 = 4 and n_0 = 10.

3.2. Connect Four Minmax algorithm

In order to improve the game , we are bringing the Minmax algorithm[3]. It is given that, one player will use minmax strategy while the other will make random moves. Taking into account the size of the board and property of the algorithm to search till the end of the search tree we are adjusting the search by restricting the depth in order to improve the performance. In order to define good move, we need to develop an evaluation function, this function simply returns positive value for the victory and the negative value of the lost. But we expanded this description by assigning values even to those levels when player can increase his layers in row. To clarify the lines given above, just think that initially we have 1 element in a row. If we attach to him second one we get $\Rightarrow$ +10, if we attach 3 element to the line with 2 elements $\Rightarrow$ +40 and finally if we attach the final elements to the line of containing 3 elements $\Rightarrow$ +150. This strategy also gathers information about the opponent's layers and uses the same evaluation with minus sign. After running 1000 games with depth 1 we get the following graph 3.2.1 , with depth =2 shown on figure 3.2.2

As it can be seen from the pie, that depth =2 gives a total victory, while depth=1 can still lose.

3.3. Breakout Controller

Breakout [4] is an arcade game, in which the aim is to hit all bricks with the ball and try to parry the ball with the pad-

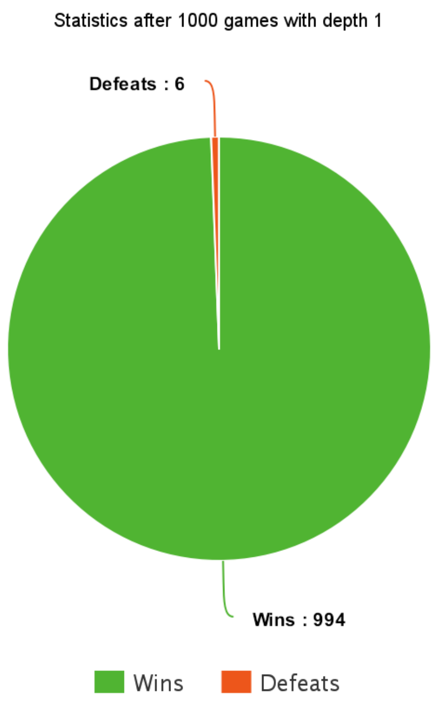

Figure 3.2.1 : Depth=1, Number of games =1000

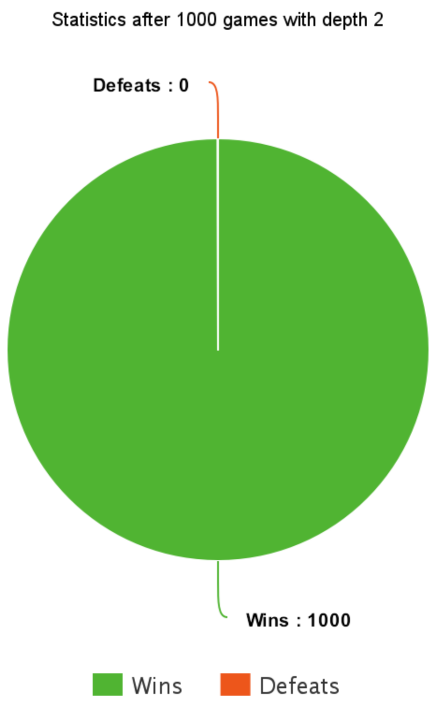

Figure 3.2.2 : Depth=2, Number of games =1000

dle.In order to improve this game, we were asked to create a controller. Main idea of the controller is to try to bring the paddle into the area in which the ball leaves the board. In order to improve this, we came to an idea to dublicate the part's of the game such as:bricks and ball. When the game starts both balls moves with same trajectory, hits the same brick and goes back, the main difference is that dublicated ball has 4 times higher speed than original ball. When the copy of the ball leaves the area, paddle starts moving towards to that area. When the original ball reaches that area, paddle is already there. This is a determenistic approach. Graphical representation of this idea is shown on Graph 3.3.1 :

3.4. Finding the path

Finally we were given a task to find an optimal path for a given dataset. This task is done by implementing 2 algorithms Dijkstra's and A* algorithms. Our dataset consists of 1 and 0's . Where 1 represents the wall or end of the path and 0 is path. Optimal path refers not only finding the shortest path but also looking for the cheapest one and trying to avoid of examining the whole area . Each movement from node to node has its own value and sum of the all nodes together has to be the cheapest. As an example, it is cheap to walk through the grass than a lava, even though the distance to next node is faster if we take the lava path. That's why greedy approach is not suitable for this kind of tasks. Output of this algorithms gives us the path which is optimal and cheap.

Our initial task is to transform the dataset into the map and try to implement the given algorithms.Based on our dataset we achieve the following labyrinth shown on 3.4.0

3.4.1. A algorithm*

A* [5] algorithm is one of the best search algorithms that lookss for an optimal path. The main idea of this algorithm that in addition the calculation of the cost of movement from one node to another, it also uses heuristic approach by giving estimation of the distance from current node to the termination node. This step makes it to generate more natural graph. Output of this algorithm can be seen on Figure 3.4.1

3.4.2. Dijkstra's algorithm

Dijkstra's [5] algorithm is one of the useful and important algorithms available for generating the optimal solutions for class of finding path problems. This algorithm is searching for the neighbors of the starting node, and then looks for the neighbors of the nodes it has found. While performing this operation it assigns or updates distance due to the cost and closeness. This algorithm uses priority queue. After applying the following algorithm , we achieve a graph showing the optimal but unfortunately human-unfriendly graph 3.4.2

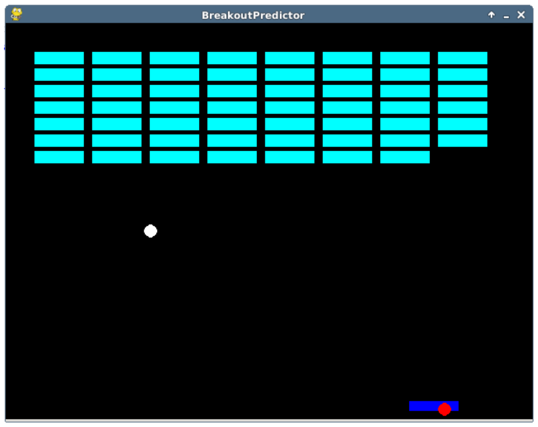

Figure 3.3.1 : Breakout's Controller

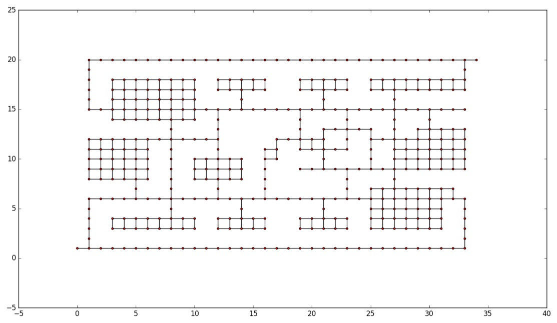

Figure 3.4.0 : Path

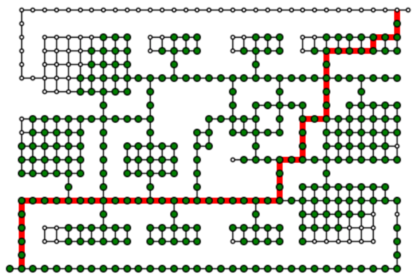

Figure 3.4.1 : A* algorithm

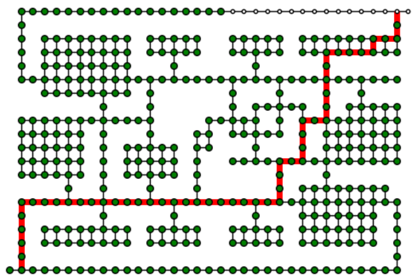

Figure 3.4.2 : Dijkstra's algorithm

4. BEHAVIOUR PROGRAMMING

Behavioural programming is a technique in software development which represents a certain scenario which system follows or not follows under certain conditions. As a primitive example can be a game in which depending on the choice of the room, the certain scenario will be used. Example : counter strike.
In this project we extended the size of the board for connect four game using the same technique to figure out the statistics, breakout will be improved by adding fuzzy logic into controller, self organizing maps represents player movements and as a conclusion we are implementing Bayesian imitation learning.

4.1. Extended Connect Four

We are asked to bring statistics for extended Connect Four board game. By terms of extended we refer that the size of the board is increased to 19x19. The statistics for this version doesn't differ from the previous one. The only significant difference is time. Running time for this extension is much more longer than the previous due to enlargement of the board. Figures 4.1.1. and 4.1.2. shows the statistics:

4.2. Fuzzy controller

In order to improve breakout game, we are adding fuzzy logic into our controller. Fuzzy logic[6] can present the degrees of truth and false. Example of fuzzy logic can serve the statement "Student wrote the test nicely" can be 100 true if he made no mistakes, 70 true if he made 3-4 mistakes, 50 true if he had half of the test wrongly and 0 if all the answers were wrong. To be precise fuzzy logic stands for the if else cases, where condition defines the probability of the statement. Let's apply this technique to breakout's controller. After improving breakout, by implementing the controller. Our aim were to add fuzzy logic into, by assigning set of rules. Particularly the velocity of the paddle is constant and in this task we are going to adjust it. Taking previous idea into account, especially the ball movement, we are dividing the area in which

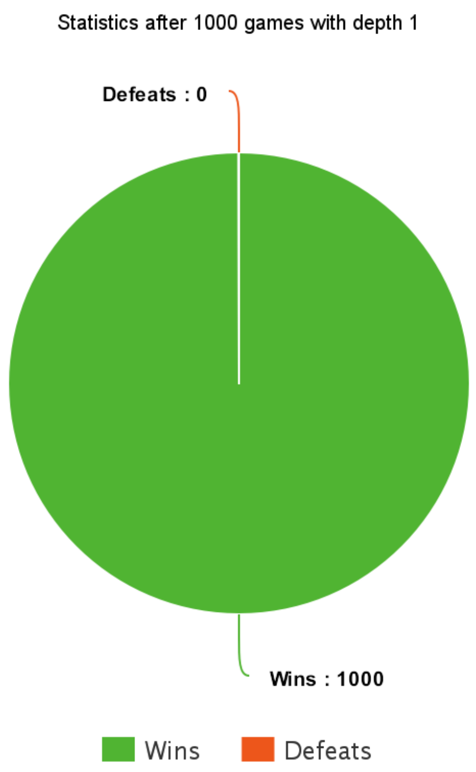

Figure 4.1.1. Results

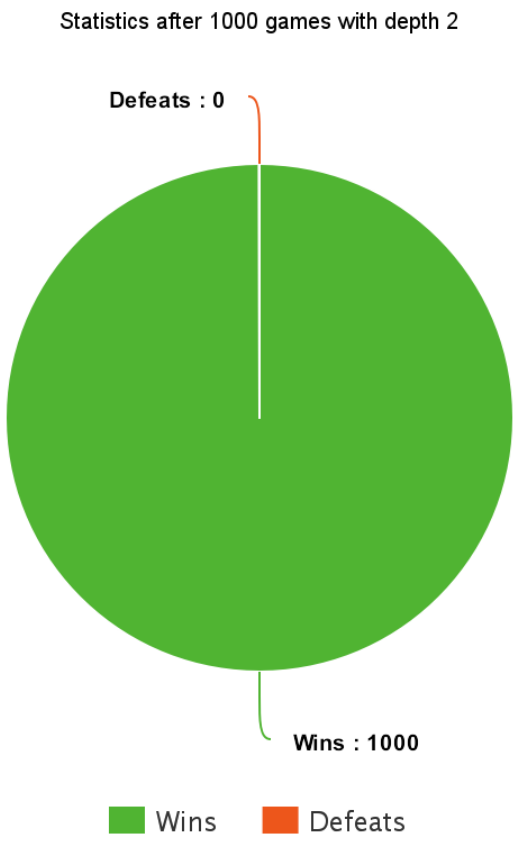

Figure 4.1.2. Results

the copy ball will leave the board and the paddle into 4 slots. This part can be observed on a Figure 4.2.1

Depending on the location of slots, the velocity will vary. Let's take into account the worst case and provide a formulas for its velocity:
Slot 1 = 5 units/framecall
Slot 2 = 10 units/framecall
Slot 3 = 12.5 units/framecall
Slot 4 = 15 units/framecall

As you can see from the Figure 4.2.1 given above, there are zones which are intersected to both slots. Which velocity should be taken into account in those zones? This is the place where fuzzy logic works. To make the definition simpler, there is a need to provide a picture 4.2.2.

Let's assume that the red line is the place where the copy ball lands, we are drawing the straight line from there untill it crosses both slots, in our case it first crosses the first slot, pointed as blue, and next crosses the second slot, pointed as yellow. Then we are splitting those slots into 2 parts and finding its area. The founded are will be the assigned as a velocity of the paddle. In this example the formula for finding the velocity is :

$$V = \frac{5 * Area(1) + 10 * Area(2)}{Area(1) + Area(2)}$$

where 5 and 10 are the velocity of Slot 1 and 2 accordingly.

4.3. Self organizing maps

Self-Organizing map [7] is one of the most popular neural network model. It based on unsupervised learning. SOM reduces the dimensions and displays similar objects. The most common application where SOM used is clusterisation.
We are using this technique here to visualise the input or to represent the player movements. Structure of the SOM:
1)An area V of the N-dimensional input space.
2)A distance metric in input space
3) Number of P patterns, vectors X from V, training set.
4)Number of K neurons, each one with a center C_k
Each SOM neuron has a center vector C_k defined in the N-dimensional input space. It is used to compare with the input

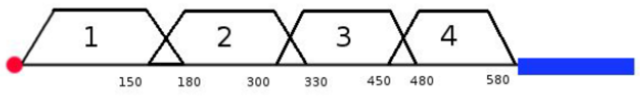

Figure 4.2.1 : Splitting

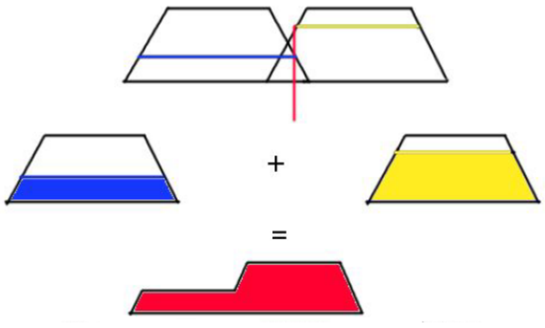

Figure 4.2.2 : Fuzzy logic

vector X. This neuron doesn't have weight, it has centers.
5) A grid structure G for the K neurons defining a neighborhood among them
6)A distance metric dist() defined on the grid G.

$$r_k = ||C_k - X||_2$$

7)A neighborhood function h(dist), adjusting the amount of learning

$$h(dist(i,j)) = \frac{||i-j||}{2 * sigma}$$

This functions shrinks with time.
8)A labeling procedure that can assign a meaning for every reference vector or center C from V
Functionality:
SOM's implementation are shown on Algorithm 2:

Results are given on Figure 4.3.1 and Figure 4.3.2. :

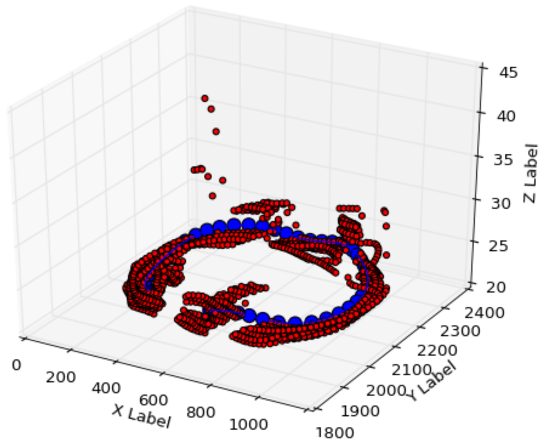

Figure 4.3.1 SOM of the Figure O

Algorithm 2 Self organizing maps

```
1: procedure SOM(data,prototypes)
2:     ▷ Choosing randomly the point from the input space and comparing it with all neurons we have in SOM.
3:     $randomValue = data[Random]$
4:     ▷ Assigning random value to the neuron centers
5:     $prototypes = data[Random]$
6:     ▷ Using distance metric function for finding the values.
7:     $Dist[index] = dist(randomValue, prototype)$
8:     ▷ After finding all the values we are looking for lowest value in our array. The neuron with the lowest value in distance is interpreted as a winner.
9:     $prot[winner] = low(Dist[indexOfLowest])$
10:    ▷ We update the winner neuron. We adjust the neuron using the following function.
11:    $prototypes[winner] = prototypes[winner] + \triangle prototypes[winner]$
12:    $\triangle prototypes[winner] = \triangle C_i = learningRate * (X - C_i)$
13:    ▷ We update its neighbors. We are using the neighborhood function to adjust the values of the neighbors of the winner neuron.
14:    $prototypes[neighbor] = prototypes[neighbor] + \triangle prototypes[neighbor]$
15:    $\triangle prototypes[neighbor] = \triangle C_j = learningRate(t) * h(dist(i,j),t) * (X - C_j)$
```

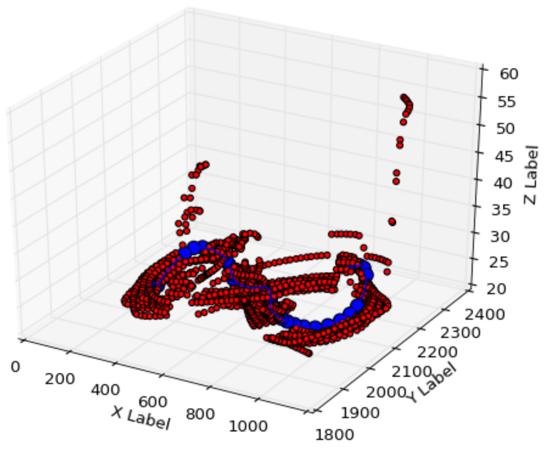

Figure 4.3.2. SOM of the Figure 8

4.4. Bayesian imitation learning

Imitation learning [8] is a learning in which teacher imitates some action and wants the system to repeat it. This approach is contained in this task. Solution here is splitted into 4 steps:
1. Using SOM approach to find prototypes for the dataset, containing player's movements.
2. Calculate activities, activities are movement from one location into another:

$$a_t = x_{t+1} - x_t$$

3.Using SOM approach to find prototypes for the dataset, containing player's activities.
4.Find the joint probability:
Joint probability is represented as an array, whose size equals to number of clusters, in our case 50x50. To be precise with calculation, we are going through the entire data, location and activity, and define to which prototypes does it belongs. As an example location at position 3 belongs to prototype 4, and activity at that position belongs to prototype 7. This means that Joint array at the position [4][7] increments by 1.
After adjusting joint array, we are building a graph, based on this array. This graph represents the movement of the player:
For the figure O, movement of the player is on Figure 4.4.1:

For the figure 8, movement of the player is on Figure 4.4.2:

Generally, this task imitates the movement of the player whose activity differs with the location and then it will lead him to some point. This is done in order that the repeating one will take the same path being at that location. Activity selection is in hands of joint probability.

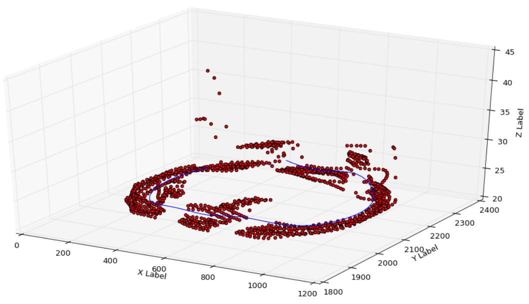

Figure 4.4.1:Path of the player on O-shaped figure

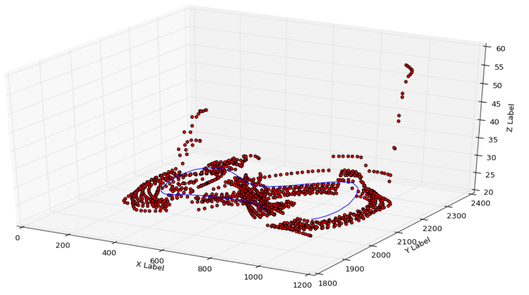

Figure 4.4.2:Path of the player on 8-shaped figure

5. CONCLUSION

Let's give an overall overview about the topics. In this project we were faced to create tic tac toe, connect four, breakout and path finding games. Applying probabilistic approach to turn-based game, particularly tic tac toe, increased the winning probability of our player. Heuristic strategies were recognized as a best approach for these games. Those approaches beats the probabilistic approach as well. In other words the possibility to win the game with the same configuration using heuristic strategy is higher than using the probabilistic approach. Random player has no chance to win. These statements stands true for both Tic Tac Toe and Connect Four. One more fact is these 2 games has simple configuration, that's why heuristic approach suits them. If we consider complex games the calculation becomes impossible.
Breakout has its own methods of improvement. The main role in this game plays the paddle and our aim was to improve it. As a simple approach we improved the controller by duplicating everything and moving the paddle into the direction in which the duplicate of the ball falls. But because of the constant speed of the paddle this approach wasn't so effective. Next step was to remove this drawback by fuzzy-logic approach. Simple splitting of the area and assigning the velocity value appropriate to the ball fallen area solved the problem.
Path finding game is solved by A* and Dijkstra's algorithms. Both of them are appropriate for this kind of tasks. The main point about these algorithms are that they support heuristic

approach. In other words they look into N steps front in order to find the optimal path. If we compare them against each other A* has more human-readable form of output, while some steps of Dijkstra's algorithm has no logic.
Self organizing maps and bayesian learning can belong to same subset, clusterisation of the data and then the use of the clusterisation to imitate the movement of the player in order to find out the path of the player in Qauke III Arena. Initially we had a set of data and our aim was to cluster them, but with addition the Bayesian imitation learning our goal was to find out which acitivity is more preferred to the default location, in other words which direction will the player preffer to move being at location A.
The overviewed topics are not new, but there is a lot of work needed to be done in order to improve them, as we said mostly strategies which asked are based on heuristic approach. But what to do with configurations which has a lot of number of states and actions? Path planning on the other side is also takes this approach , but there are a lot of configurations in which these algorithms are not effective.
Despite of the above paragraph, all of them are working well in different scientifical fields and showing good results.

6. REFERENCES

[1]Christian Bauckhage, **Rheinisch-Westflische Technische Hochschule Aachen**, "B-IT Game AI lecture slides" , *Article*, sites.google.com/site/bitgameai/home/lecture-notes **Lecture 4**

[2] Jing Li and Melissa Gymrek, **Massachusetts Institute of Technology**, "The Mathematics of Toys and Games" , *Article*, www.web.mit.edu/sp.268/www/2010/connectFourSlides.pdf

[3]Christian Bauckhage, **Rheinisch-Westflische Technische Hochschule Aachen**, "B-IT Game AI lecture slides" , *Article*, sites.google.com/site/bitgameai/home/lecture-notes, **Lecture 5**

[4]Mark Nelson , "Breaking Down Breakout: System And Level Design For Breakout-style Games", *Article*,
www.gamasutra.com/view/feature/1630/
breakingdownbreakoutsystemand.php?print=1

[5]Harika Reddy , "PATH FINDING - Dijkstras and A* Algorithms", *Article*, cs.indstate.edu/hgopireddy/algor.pdf

[6]Franck Dernoncourt, **Massachusetts Institute of Technology**, "Introduction to fuzzy logic" , *Article*,
www.francky.me/doc/course/fuzzy-logic.pdf

[7]Nils Goerke, **The University of Bonn**, "Self organizing maps", *Article*, www.ais.uni-bonn.de/WS1516/4204-L-NN.html

[8]Bob Price and Craig Boutilier , "A Bayesian Approach to Imitation in Reinforcement Learning", *Article*,
cs.toronto.edu/ cebly/Papers/BayesImitation.pdf