Indian Institute of Technology, Kanpur

Game AI Project Report

TEAM MEMBERS:
Shubham Sourav & Ayush Tharwani

MENTOR:
Abhishek Yadav

July 9, 2018

Contents

1 Aim Of The Project **2**

2 Overall Timeline **3**

3 Fundamental Algorithm **4**
3.1 Minimax Algorithm 4
3.2 Alpha-Beta Pruning 5

4 Tic-Tac-Toe **6**
4.1 How to play ? 6
4.2 Scoring 6
4.3 Implementation 6
4.4 Link of the code 6

5 Connect-4 **7**
5.1 How to play ? 7
5.2 Scoring 7
5.3 Implementation 7
5.3.1 Minimax 7
5.3.2 Alpha-Beta 8
5.3.3 Others 8
5.4 Link of the code 9

6 Make Way **10**
6.1 How to play ? 10
6.2 Scoring 10
6.3 Implementation 11
6.4 Link of the game 13

7 Future **14**
7.1 Connect-4 14
7.2 Make Way 14

8 Other important links **15**

9 Team work division **15**

1 Aim Of The Project

The main aim of the project is to introduce to various algorithms and using them make AIs for different games.

The main algorithms used are **Minimax** and **Alpha-Beta Pruning**.

The games on which these algorithms will be tested are **Tic-Tac-Toe**, **Connect-4** and **Make Way**.

2 Overall Timeline

May 18	Learned Minimax Algorithm.
May 20	Learned Alpha-Beta Pruning.
May 21	Started Tic-Tac-Toe.
May 26	Finished Tic-Tac-Toe.
May 27	Started Connect-4.
June 2	Completed first proto-type of Connect-4.
June 3	Worked on improving Connect-4.
June 5	Completed Connect-4.
June 6	Started learning unity.
June 10	Started MakeWay.
June 20	Finished making the two-player mode of the game.
June 21	MID-TERM EVALUATION.
June 22	Started making AI for MakeWay.
June 30	Finished making the one-player mode for MakeWay.
July 1	Started making scenes for MakeWay and connect them amog themselves.
July 9	Finished MakeWay.
July 10	END-TERM EVALUATION.

3 Fundamental Algorithm

3.1 Minimax Algorithm

Minimax is a **decision-making algorithm**, typically used in a turn-based, two player games. The goal of the algorithm is to find the optimal next move.

In the algorithm, one player is called the **maximizer**, and the other player is a **minimizer**. If we assign an evaluation score to the game board, one player tries to choose a game state with the maximum score, while the other chooses a state with the minimum score.

In other words, the maximizer works to get the highest score, while the minimizer tries get the lowest score by trying to counter moves.

Our goal is to find the best move for the player. To do so, we can just choose the node with best evaluation score. To make the process smarter, we can also look ahead and evaluate potential opponent's moves.

For each move, we can look ahead as many moves as our computing power allows. The algorithm assumes that the opponent is playing optimally.

Technically, we start with the **root node** and choose the best possible node. We evaluate nodes based on their evaluation scores. In most of the cases, evaluation function can assign scores to only result nodes (leaves). Therefore, we recursively reach leaves with scores and back propagate the scores.

Maximizer starts with the root node and chooses the move with the maximum score. Unfortunately, only leaves have evaluation scores with them, and hence the algorithm has to reach leaf nodes recursively.When it is the minimizer's turn the nodes with minimum scores will get selected. It keeps picking the best nodes similarly, till it reaches the root node.

Now, let's formally define steps of the algorithm:

1. Construct the complete game tree.
2. Evaluate scores for leaves using the evaluation function
3. Back-up scores from leaves to root, considering the player type:
    - For max player, select the child with the maximum score
    - For min player, select the child with the minimum score
4. At the root node, choose the node with max value and perform the corresponding move

3.2 Alpha-Beta Pruning

Alpha-beta pruning returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.

Alpha: It is the best choice so far for the player MAX. We want to get the highest possible value here.
Beta: It is the best choice so far for MIN, and it has to be the lowest possible value.

Now, let's formally define steps of the algorithm :

1. Set the value of Alpha at the initial node to -Limit and Beta to +Limit. Because initially these are the max values that Alpha or Beta could possibly obtain.
2. Search down the tree to the given depth.
3. Once reaching the bottom, calculate the evaluation for this node.(i.e. it's utility)
4. Backtrack, propagating values and paths according to the following:
    - If the move being backtracked would be made by the opponent:
        - If the current score is less than the score stored at the parent, replace the score at the parent with this and store the path from the bottom and the value of Beta in the parent. If the score at the parent is now less than Alpha stored at that parent, ignore any further children of this parent and backtrack the parent's value of Alpha and Beta up the tree.
        - If the score at the parent is greater than Alpha, set the Alpha value of the parent to this score and proceed with the next child, sending Alpha and Beta down. If no children exist, propagate Alpha and Beta up the tree and propagate the value of Alpha up as the min score.
    - If the move being backtracked would be made by the computer:
        - If the current score is more than the score stored at the parent, replace the score at the parent with this and store the path from the bottom and the value of Alpha in the parent.
        - If the score at the parent is now more than Beta stored at that parent, ignore any further children of this parent and backtrack the parent's value of Alpha and Beta up the tree.
        - If the score at the parent is less than Beta, set the Beta value of the parent to this score and proceed with the next child, sending Alpha and Beta down. If no children exist, propagate Alpha and Beta up the tree and propagate the value of Beta up as the max score.
5. When the search is complete, the Alpha value at the top node gives the minimum score that the player is guaranteed to attain if using the path at the top node.

4 Tic-Tac-Toe

4.1 How to play ?

Tic-tac-toe is a simple, two-player game in which players alternatively places X and O in the matrix until either one player has three in a row, horizontally, vertically, or diagonally; or all squares are filled.

If at any moment one player had a n in a row, horizontally, vertically, or diagonally than that player is Winner whereas if all the n*n squares are filled and no one player had a n in a row then the Game would be declared as a Draw.

4.2 Scoring

There are three types of outcome :-
Win : 20
Lose : -20
Draw : 0
But in case of bigger boards computer will not be able to find any of the above .

4.3 Implementation

We have provided each move played in the game a certain score based on certain conventions. The more the score is negative, it favors computer's victory and works other way around for player. So our algorithm tries to evaluate minimum score move followed by subsequent max score move until the end of game. In this way, it decides a move to be played at that certain state of the game.

We have improvised our evaluation by including depth in scoring conventions. So a victory with less moves will be favored more.

4.4 Link of the code

Tic-Tac-Toe code in github

NOTE :

- The tic-tac-toe made by us is of size n*n so one can play the game in different board sizes.
- Instead of X and O we have used 1 and 0.
- Code has been written in Python language.

5 Connect-4

5.1 How to play ?

Connect-4 is a two-player connection game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs.

5.2 Scoring

- So,first of all for direct win I used **1000 points** and similariliy for direct lose I used **-1000 points** and 0 for draw.
- Now,a new case arises ,when computer is not able to decide among any of the above 3 cases. So,in case what to do ?
- So, at all such stages I used to calculate how many active 3 in a row are present at that point of time for both computer and player. Then I multiplied both that number by 10 and then calculated the new score by using the formula

$$score = computer - 2 * player$$

- I used to twice the player count because the main aim of our game is not to win but not to lose.

By using the above scoring method ,we basically motivate the computer to make 3 in a row and then use them to make 4 in a row.

5.3 Implementation

5.3.1 Minimax

So, we created three functions for this purpose :-

- A **BestMove** function which puts the disc one by one in each column and calculated the feasibility of that chance.
- A **minimax** function with two parts max and min which took alternative turns to make there chance.
- A **score** function which returned 1000 if computer wins, -1000 if player wins and 0 otherwise.

In order to improvise the minimax algorithm we also made use of the depth functionality i.e. if the computer wins by putting discs in two column one in 2 chances and other in 3 chances then by using depth functionality we can prefer the one in which computer wins in 2 steps (same goes for losing cause) .

Depth keeps on increasing keeping time constraints in mind.

5.3.2 Alpha-Beta

Well Alpha-Beta in itself is very fast but still we can enhance it's speed by searching first those columns where the probability of winning is more.

In case of connect-4 this mainly happens in the inner columns than the outer columns.

So, in the minimax function where we check by putting discs from column 1 to 7 we can instead put from 4 then 3,5 then 2,6 then 1,7.

This almost doubles up the speed.

5.3.3 Others

In order to improve the accuracy of the game in the beginning or at times when computer is not able to find any winning move i.e there is a tie in the scores for two columns than instead of putting the disc in 1st column or randomly in any column put it in the inner column i.e. 4th column as there is more probability to win as we can make many winning moves using them. In order to use randomness we can choose randomly among 3rd and 5th column or 2nd and 6th column in case of tie among them.

In order to avoid piling up of discs in the middle column we can use the fact that it is beneficial to put discs in odd rows if given first chance and otherwise if given second chance. This is done because there are 6 rows i.e. even so if all the columns except one are filled up then one will put the disc in the odd row and so if we have more discs in odd row then our winning chances enhances as we can easily make horizontal line of 4 disc by enforcing it as opponent has no other move but to put there.

Regarding its implementation, I added 0.5 to the score of that column if the discs would have been filled in odd row or even row depending whether we make first move or the second.

0.5 has been used because if any other whole number would have been chosen than it might have changed the priority of the moves which originally had different scores.Note that it is an improvement for case when highest scores have multiple occurrences.

5.4 Link of the code

Connect-4 code in github

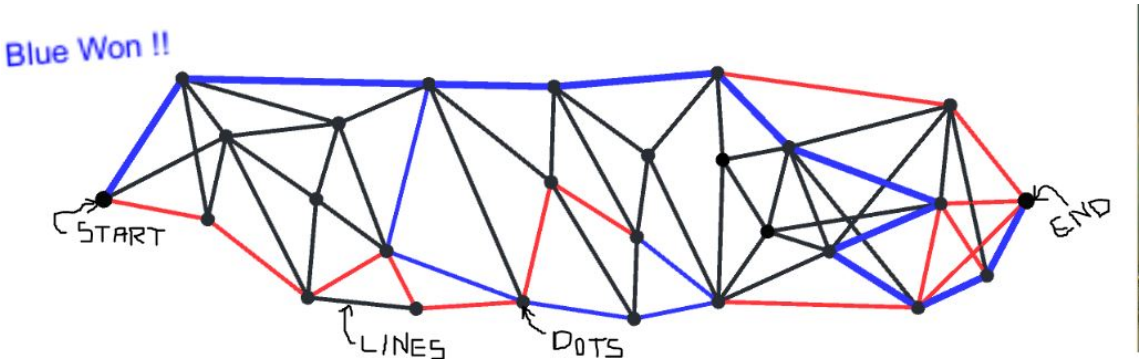
NOTE :

- To represent the pieces we have used O and X . Note that X gets the first chance to make its move.
- We have added four different levels in the game . Different level refers to different depth i.e 2,4,6,8 up-to which computer anticipates its next moves
- Code has been written in C Programming language.

6 Make Way

6.1 How to play ?

Make Way is a two-player-connection game. There are many lines between small dots using which you have to connect two bigger dots . The player which connects first wins the game. Each player gets to choose one line at a time.



6.2 Scoring

- There are three types of situation which can arise during the gameplay
  - Win / Lose : 1000000/-1000000
  - Enforce other/yourself not to win: 100000/-100000
  - Draw : 0
- Now what to do when the computer is not able to get any of the before-mentioned cases. We need to drive the computer towards victory. So,the first thing which comes to mind is to take away all the bigger lines (in terms of length in x-direction) .
  - So,I took the sum of all the length of computer-selected lines in the x-direction and subtracted it with that of the player.
  - But there is a problem with this technique i.e. the computer doesn't make moves in continuation and does it randomly making it very difficult to win.
- Next motive was to make the computer do its move in continuation . Note that we have the index of all dots in ascending order i.e. start has 0 index and then increasing till the end.
  - So, we took the difference of the index between the starting and ending dots of all the series of line in continuation and raised it to the power of a suitable number which in our case comes out to be 1.3 .

  - By adding this our computer was able to make moves in continuation but it could continue in the opposite direction which happened quite usually. So,our next aim is to avoid this type of negative continuation.

- So, we calculated start_pos i.e. upto how long the starting dot has been connected and similarily end_pos .
  - Then, we doubled the scores of all the moves in which the indexes are in-between start_pos and end_pos.
  - By doing this we where able to make the computer move in positive direction but by playing many times we realized that instead of using start_pos and end_pos we should use start_pos-2 and end_pos+2 as sometimes it is necessary to move backward.
  - All these was good, but we needed to make computer realize that start and end dots are too important and that they must be secured first so we incresed the score of all those lines which where connected to either start and end dots by raising them to the power of 1.4 instead of 1.3 .
  - Also we ensured that till the end of five chances at least one connection each from start and end dots would be made.
- Now to calculate the final score of the move which was not among the decisive one we used the formula

$$score = 10 * computer - player$$

  We gave such less weight-age to player because this game is very difficult to win by blocking the opponent as the grid is too big. So, instead of defensive play computer has to play aggressively.
- Now what happens if computer has been enforced not to win by blocking its way. We must change our strategy as there is no use of making long chains as instead we have to block the player as far as it is possible.For this we need to change our strategy. For such cases we changed the formula to

$$score = -5 * player$$

6.3 Implementation

The dots were made with the use of spheres and the lines are made with the use of the cylinders originating from the center of the sphere.

PROBLEM The first task was to make the game plan random.

SOLUTION So, we made many points(80) and then chose 25 distinct points from them randomly.

PROBLEM But if it is random then it may happen that all points are close by or in just a half of the screen.

SOLUTION So,we divided the game area into 4 subarea and chose certain amount of points from each subarea.

PROBLEM But it may happen that two points which are very close to each other are selected.

SOLUTION To have sufficient distance between two points we put the constraint that the difference in x any y coordinate of any two points should be atleast 2.

PROBLEM How to join the points randomly

SOLUTION We chose a random number between 2 and 4 and made that number of connection from that particular point to the subsequent points.In this way even if the same points are choosen but the connections among them will be different

PROBLEM Now it was happening that some lines were very close to each other so we need to avoid them.

SOLUTION For this we discarded all the bigger lines where the angle was less than 14 degrees or if the line was bigger than a given length(so that the game don't become very easy and one-sided).Now, we need to take care of the fact that there should be at least two connections from start and end dots.

PROBLEM How to know if there is a line between two points and get access to that line.

SOLUTION We have created a 2D array in which a data element 1 suggests there is a connection between those two points.

Parallely we created another 2D array which stored the link to the line gameobject if corresponding element in previous array is 1.

PROBLEM But using array to store the connections is very much space consuming as many array element are empty.

SOLUTION So,we used lists as in it an element can be added at any time i.e. size not has to be given beforehand.

PROBLEM Now how to know which player gets the first chance.

SOLUTION So, we included a toss scene where a player is randomly decided to make the first move.

PROBLEM How to pass value of a variable to different script in a different scene as variable are destroyed after the scene changes.

SOLUTION So,to do this we used static variables as they are not destroyed even on scene changes.

The scoring method has already been discussed.

We used minimax with alpha-beta pruning for this purpose.We also checked depth of the move so that the computer makes the move in which result occurs fast.

PROBLEM What about DEPTH ??

SOLUTION DEPTH used for this purpose was only 2 as the grid was too big(27 dots and nearly 70-80 lines) thus making it very difficult to search all the possibilities . Also we cannot skip any area of the grid as all are very important.

6.4 Link of the game

MakeWay code and game in github

NOTE :

- This game contains both one player as well as two player mode.
- The whole game has been built on the unity platform.
- All the programming work has been done on C#.

7 Future

7.1 Connect-4

Our game is almost perfect as it wins in most of the cases . But still then it is not perfect as depth is not the whole play area. So, to make it perfect we can try the following things :-

- We will introduce transcription table i.e. as a game board is evaluated its value is stored so when we arrive at such a situation we can directly refer to that value instead of carrying the whole lot of calculations once again.
- Store the whole game board in the form of bit-boards as by it the memory space will decrease and our transcription table can store more and more values.

7.2 Make Way

- As for now the computer is able to win 75% of the games. We need to improve this accuracy.
- We need to add some more facilities to the game before it is ready to be launched on any platform such as the facility to save the game in the current state.

8 Other important links

- Video of the Gameplay
- Final Project Presentation

9 Team work division

- TIC-TAC-TOE : AYUSH THARWANI
- CONNECT-4 : SHUBHAM SOURAV
- MAKE WAY : SHUBHAM SOURAV & AYUSH THARWANI