University of Essex

School of Computer Science and Electronic Engineering

Computer Science Capstone Project

<u>Topic: AI Solver for Puzzle Games</u>

Pratham Mittal

Supervisor: Professor Luca Citi

Second Assesor: Dr Ana Matran-Fernandez

Registration Number: 1905188

Word Count: 6887

Acknowledgments

First and foremost, I want to express my gratitude to my supervisor, Professor Luca Citi, for his unwavering support during my undergraduate capstone research project. Whenever I needed his advice along the process, he was only an email away. I needed his guidance during the implementation of algorithms and research work. I can never imagine having a better mentor for this project.

Lastly, I would like to thank my friends for their continuous support, providing with much the much needed motivation in this journey.

Abstarct

Artificial Inntelligence is a technology that is helping in the rapid tranformation of every industry and buisness sector. This innovative technology has also helped in making advancements in the computer games industry. And as game developers are looking more more new innovative ways to make drive game developments, AI is taking the lead.

This project is based on analyzing and developing AI agents for Puzzle Games. I have chosen Tetris and 2048. Tetris and 2048 are two off the most well reknowned and respected puzzle games in the industry and I have always found them challenging. 2048 is a single player sliding block puzzle game with the objective to slide and merge the numbered tiles on the board and create a tile with number 2048. Tetris is a puzzle game where a player has to complete lines by moving different shaped blocks which are decending down the game board.

One of the most intresting things that I found during developing this project was that tetris has many variants and each variant have different set of rules which are followed by the game.

The project is based in Java and uses multiple machine learning algorithms. For 2048 I experimented with Minimax algorithm, expectimax algorithm, alpha beta pruning and Monte Carlo algorithm to find the most efficient path to victory. Furthering my research I build tetris based in Java and developed a genitic algorithm which checks for the best move in each situation based on heuristics provided and chooses the move with the most accurate hueristic values and the highest score after all the moves have been exhausted.

The goal of the project was to build an algorithm which can solve the puzzles efficiently with the highest possible score. This is starting point in how AI is helping with the day to day challenges faced in the industries such as autonomous driving.

Tables of Content

1. Introduction

The video games industry is one of the largest and popular industry and has been growing rapidly specially in the 21st century. This industry is has its roots in techniques used in computer science. Game playing has recently become a popular part of human life. Games have different level of difficulty that improves the experience of the player and challenges them to improve their skills in different ways. For e.g. in Tetris, the strategy is to plan and place every block in a perfect position. Players makes an error when they choose a not so efficient move that leads to lower score. Since the game has different difficulty levels player should also be good with the controls. Sometimes the player can make wrong move due to time pressure. This is specially true for Tetris. This is what makes puzzle games so interesting and set them apart from other games, they require a good strategy and a good adaptiveness. Game playing is often also viewed as a indicator of intelligence which makes it a ideal testbed for testing different algorithms.

My primary motivation for this project was to learn different Artificial intelligence techniques and how artificial intelligence is being used to solve day to day problems. And since development of AI techniques for computer games impacts several other industries as well for e.g. autonomous behavior of characters can be used in development of autonomous cars and is already being used as a the base for its development, it is the ideal first step to take in order to get a deeper understanding about this innovative technology. Also puzzle games can mimic the circumstances of real world challenge where the line between known and unknown is very hazy and is hard to distinguish, improving the ability of a artificial intelligent agent to solve a puzzle game, the boundary of solvability of those challenges becomes apparent.

Tetris is game which has been popular since last three decades. There have been several algorithms which people have experimented with and have given good results. The game is played on a two dimensional grid which gradually fills up. Tetris is game where people need to place different shaped blocks which are descending onto the board to complete a line and score points. These different shaped blocks are known as tetrominoes and the player has control on how these tetrominoes falls, they can rotate the blocks and move them right and left. When an entire row is completed the whole row is deleted. Over the years there have been several different versions of Tetris with different rules. For this project I have tried to stick to standard Tetris rules as much as possible. I am using a Tetris grid of size 15 cells wide and 23 cells tall. All the tetrominoes (Tetris blocks) start at the top. There are 7 different tetriminoe shape : “I”,”0”,”J”,”L”,”S”,”Z” AND “T”. I have used the standard Tetris rotation system to make the rotations and the game picks the shapes randomly [1]. The player can also see the next upcoming block on the game board which helps him to think about the next move while placing the already spawned tetriminoe. If a tetriminoe touches the top of the grid the game gets over. As we discussed above Tetris rules are relatively simple. The goal of our AI is to clear as many lines as possible and get the best possible score before the game ends.. For this I developed an algorithm which takes into account four heuristics: height, complete lines, holes and bumpiness. We will discuss about the algorithm in more detail further in document.

2048 is a single player sliding block puzzle game with the objective to slide and merge the numbered tiles on the board and create a tile with number 2048.It is played on a 4x4 grid on which random numbered tiles are generated usually with value 2 or 4 after every move is made. It is an addictive game since it is hard to win despite the simple rules. This was one of the reasons I chose to develop an algorithm to solve this game. Many methods and approaches have been tried by computer scientists to solve this game. In my experiment I tried experimenting with three algorithms at different stages of the project development. I used alpha beta pruning which is an extension of minimax search algorithm . It is a decision making algorithm which is designed to find optimal move for the player to help reach the goal state. Alpha beta pruning tries to decrease the number of nodes that are used for evaluation by minimax search tree. It is used to solve adversarial games. 2048 is not technically adversarial , but for the algorithm I have assumed that the computer is completely adversarial [3]. With Alpha Beta pruning the game assumes that the opponent player in this case the computer will be able to minimize the output of the maximizer player. Another algorithm that I have used is expectimax algorithm and it is also a variation of minimax algorithm. Minimax algorithm assumes that the minimizer will play optimally expectimax doesn't. For expectimax algorithm the player does not chooses the maximum value out of the minimum but it choses the maximum of the expected scores[cite].The last algorithm that I experimented with and have chosen as the final algorithm that I have implemented in my final code is monte Carlo search tree. It tries to create large number of simulations to try and achieve the goal. I decided to use this algorithm because in 2048 after every move a random tile is generated at a random location and it is impossible to predict where the tile be spawned. So monte Carlo search tree searches through every possible move in the tree and finds the best move to be made in every situation. Most notably Monte Carlo tree search algorithm has been used to solve AlphaGo.

Tetris-2048 self-explanatory by the name is a hybrid of Tetris and 2048. I made it as a variant of Tetris. This was a game that was completely developed form scratch for this project, from deciding the game mechanics till the game rules, everything was done from beginning. It uses the idea of Tetris with a twist of 2048. Instead of plain blocks, I numbered the blocks of tetrominoes to function as 2048 tiles. Developing an algorithm for this game was an extreme challenge due to the complexity of game board and mechanics and also due the fact that no research has been done in the past to solve this game which would have guided me in a direction. Due to complexity and lack of time towards the end I abandoned that game agent and switched to focusing on Tetris.

2. Aim

Puzzle games have always been an interesting and addictive genre of video games. They become popular by the help of games like Tetris and Candy Crush Saga. The challenging aspect of these games got attention of several computer scientists to develop Artificial Agents to solve the games. In this project I develop AI agents for two of the most popular puzzle games, Tetris and 2048 with the help of some Artificial Intelligent algorithms.

3. Objectives

- Study about different Artificial Intelligent algorithms used to solve games.
- Develop 2048 game.
- Explore algorithms generally used for development of 2048 AI agent.
- Implement the Artificial Intelligent Algorithm in the game.
- Develop a hybrid of 2048 and Tetris
- Experiment with different algorithms to play the game
- Develop Tetris game.
- Develop an efficient algorithm to play the Game
- Implement the algorithm in the game interface.

4. Literature Review

4.1 Artificial Intelligence in Puzzle Games

In 1769, Wolfgang von Kempelen first demonstrated, the Turk, his chess playing automation to the world(Carrol,1975). It was the first machine that created the illusion of having intelligence [1]. For years computer scientists have been developing algorithms to solve two player games (i.e., chess), where the outcome of the game can be predicted easily and efficiently using Artificial intelligence techniques and statics [13]. Puzzle games can also be single player based. The challenge with this is how to use the existing algorithms to solve single player games with the same accuracy.

Throughout the years many scientists have tried and apply algorithms are generally applied to two player games in single player puzzle games. For e.g. Alpha-beta-pruning an algorithm which is based on minimax search is widely used to make Artificial agents for chess and tic-tac-toe, but computer scientists have tried implement this algorithm in single player puzzle game by trying to see the game as a two player game where one player is computer and the other is the player. Many researchers and scientist also developed methods which can be easily applied to puzzles like A* search algorithm is used to solve the infamous sliding puzzle problem.

4.2 2048

In recent years games like 2048 have become very popular since it can be played on both web and mobile. Gabriele Cirulli [14] the author of 2048 said that the players played for an aggregate of 3000 hours weeks after the game was released. The game is hard to win even though it has fairly simple mechanics and rules. It has 4X4 board with numbered tiles inside it. The numbers are a power of two. Initially there are only a couple of tiles and the tiles can be moved around and merged together by using arrow keys. For e.g. if you press right button, all of the tiles on the game board will go towards right. If the numbers of the tiles right next to each other matches they merge and the resulting tile is the sum of the two tiles for e.g. if 4 and 4 are merged together the resulting tile is numbered as 8. The aim of the game is two get a tile 2048 by merging the tiles in the way described above. This attracted many computer scientists to develop AI agents to solve the game. In[4] the authors also thought that the game is a interesting test bed for studying AI methods. Many researchers suggested alpha beta pruning, expectimax search algorithms to solve the game and monte Carlo search tree [5].

2048 is a game that can be viewed a two player game since the job of the computer is to spawn new tiles at position which will minimize the players chance to reach the goal. So we can describe the two players as the computer and the player. This observation allows us to use alpha beta pruning and expectimax search algorithms by assuming the computer is the minimizing player and human is the maximizing player.

2048 has very large state space which can result into expensive computational cost while trying to exhaust possible the game moves. That is why I chose Monte Carlo tree search algorithm to be implemented in my final product, Firstly it is general purpose solver which will give an output without needing a game specific input. Secondly Monte Carlo is heuristic algorithm that focuses on finding the best possible move. And it does not traverse the entire search tree while looking for the best move, it chooses the branch which has a better potential. Hence even in the case of 2048 which has large state space, algorithm has a deeper search depth. Thirdly MCTS runs simulations to evaluate which node will provide a better result.

4.3 <u>Tetris</u>

Tetris has been around for more than two decades and requires precise decision making skills to master the game hence making it difficult game to master. Due to this reason Tetris is considered as a benchmark for research in artificial intelligence. Tetris is estimated to have 2^{200} states[17]. Due to the large scale the most efficient approach h to develop algorithms to solve tetris is to learn a policy with the help of features that describes the real time board state.

Tetris is difficult to study due to number of reasons. Firstly it is a highly complex game and over the years there have been many different variants of the game. Secondly it can take a long time to complete the game depending the individual skill set. Due to this scientist go for small game board generally smaller than 20 x 10, but to make the project interesting I am using a game board of size 23 x 15. First notable attempts of developing an AI agent to solve the game came in 1996. [6] used Tetris as test bed for dynamic program based on heuristics. They used to number of holes and height of highest current column on the game board. [8] managed to clear 50 lines by using reinforcement learning and gaussian kernel. Since then there have been significant progress and better algorithms have been developed. In 2009, Boumaza introduced the covariance matrix adaptation evolution strategy (CMA-ES) an algorithm to solve Tetris. His algorithm managed to clear 35,000,000 lines.

4.4 <u>Algorithms</u>

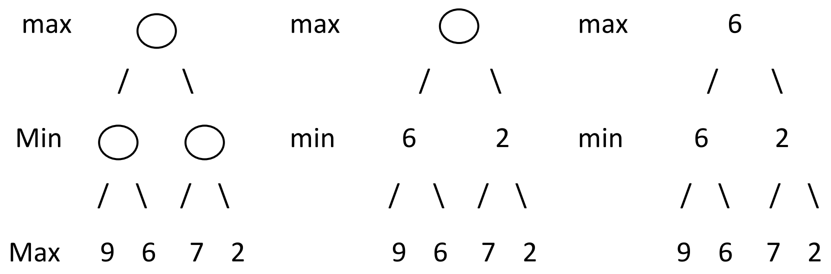
Any artificial intelligent system needs a set of instruction which are used for the task to be carried out in an efficient manner. In artificial Intelligence an algorithm gives instruction in which help the computer to learn to operate on its own [12]. These set of instructions that helps the computer to complete the task are knows as algorithms. This section talks about the algorithms that have been used in the project.

4.4.1 Minimax Search Algorithm

As mentioned above minimax search algorithm is designed for playing adversarial games. In other words games that are played by two players. It is generally used in decision making and game theory. The idea behind the algorithm is that one player i.e. the opponent, always tries to minimize the chances of other player to win, whereas the player will always try to maximize his probability to win. The algorithm assumes that both players play optimally.

The goal of the algorithm is to find the best possible move, and to achieve this the algorithm develops a tree to look ahead as many moves as possible after every turn taken and chooses the move with the best result.

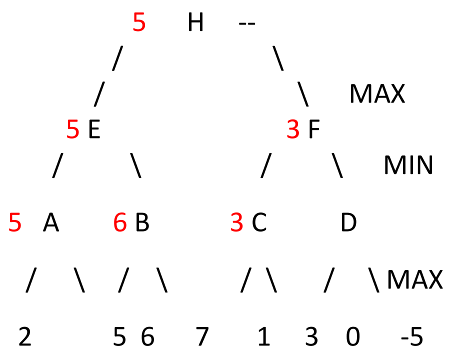
In the tree the algorithm start at the root node and chooses the next best node. To choose the next best node a evaluation function is used which assign values of the nodes. Since the first player will always try to maximize their chance to win, the maximizer always start at the root node and chooses the node with maximum value assigned by the eval function. Then the minimizer chooses the minimum value assigned by the eval function as it wants to minimize the chance of the opponent winning. The maximizer always chooses the larger vale and the minimizer will always choose the smaller value. The diagrams below illustrate the functioning of the minimax algorithm.



4.4.2 <u>**Alpha Beta Pruning**</u>

Alpha beta pruning is an optimized version of minimax algorithm. The difference between the two is that Minimax search algorithm explores every single path whereas alpha beta pruning does not explore every single node. This helps in reduction of computer processing time and processor power required by the machine to efficiently run the algorithm.

Alpha beta pruning cuts off branches in the search tree which are not required to be searched as it has already found a better move to be made. In the algorithm alpha is maximum value that the maximizing player has and beta is the minimum value the minimizing player has at that specific state of the game board. The example below explains the working of the algorithm in more detail.

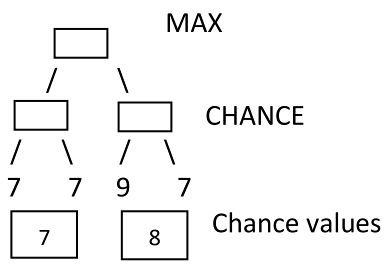


- The call start from H. The values of alpha is -infinite and beta is +infinite at this point and these values are passed down. At H maximizer choose maximum value E and F. So H calls E.
- At E the minimizing player needs to choose the minimum value out of A and B. Hence it calls out A.
- At A, the value is max. So it looks at the leaf node and tries to decide if it should consider the right node or not. The value of leaf node is 2. It checks if the beta<=alpha, in this case it is false and hence it looks at the right node where the value is 5, since 5 is greater than 3, the value at A is now 5.
- A passes the values to E. Now at E since it is minimum beta = min(+inf, 5) which is 5. Now minimizer has a guaranteed value of 5 or less. E now calls B to see if it has a value lower than 5.
- B gets a value of 6, since 6 is greater than 5, E being the minimizer chooses 5.
- Note: Since 7 was never computed it saved the computational power and time. This is the difference between alpha beta pruning and minimax search. In minimax it would have ran through every node irrespective of the result after each node.
- Similarly the second sub tree of C,D,F is calculated and the value passed comes to be 3. Now F = 3
- Now H seeks the maximum value of F and E which is 5. Hence the optimal value that maximizer can get is 5.

4.4.3 Expectimax

Expectimax search algorithm is similar to minimax search algorithm. The difference between the two algorithms is that minimax search algorithm assumes that the minimizing player plays the game optimally whereas expectimax does not. Expectimax algorithm is use in cases or games where the result of the move is not only dependent on player skill level but also the probability of player choosing wrong. This second consideration of probability of player being wrong is what that separates expectimax and minimax algorithm. Thanks to this probability in the game tree for expectimax algorithm we also add a row of chance nodes, so unlike minimax where there are only maximizing and minimizing nodes, here we have max, min and chance nodes. This changes the weights of nodes from min or max to expected value nodes or expectimax values. The expectimax values are derived from the help of child nodes.

Example game tree below helps us understand this better.



As it can be seen by the tree above chance values are calculated by taking the average of the two child nodes, this gives us the expected value. In our case the value that gets passed on to maximizer is 8.

4.4.4 Monte Carlo Tree Search

MCTS is a heuristic driven search algorithm which focuses on giving the most promising moves. It combines the classic tree search method which is also used in Minimax, and reinforcement learning which is a machine learning method. Monte Carlo Tree search comes into play when there are a large number of potential actions that are required to be made at every single stage of the game tree. MCTS algorithm is based on multiple number playouts i.e. at each stage it plays the game until there is a result by selecting moves at random. Then the algorithm weighs the nodes according to result of each playout, so that the nodes with the higher weigh are chosen during the playout at the next stage. As the game progresses the algorithm will have to chose the alternative nodes as well instead of the current strategy being used. Here comes in play the "exploration-exploitation" nature of the monte Carlo tree search algorithm. The algorithm exploits the strategy and makes alternative moves in order to find a better move. The exploratory nature of algorithm helps in exploring the unknown parts of the game tree which can lead to more efficient way to the goal state. This ensure that the algorithm is not missing out on any possible more optimal ways. Exploitation focuses more on getting the most out a single path. This balances out the tree as exploration increases the breadth of the tree, exploitation increases the depth. [15] Levente Kocsis and Csaba Szepesvári introduced the UCB (*Upper confidence Bound)* formula which helps in balancing out the tree breadth and depth which is increased by the exploitation and exploratory nature of the algorithm. [18] The UCB value of a node is given by the formula:

$$UCB = Vi + C \,.\, \sqrt{\ln \frac{N}{ni}}$$

Where,

- Vi is the estimate value of the node/

- N is number of times parent node has been visited.
- Ni is number of times node has been visited.
- C is exploration constant

The process of MCTS is easy to understand, according to the results of the playouts it gives weighs to the nodes and then node by node it builds a tree. The steps of building this tree are selection, expansion, simulation and back-propagation.

Selection : Selection is the process where the nodes are selected form the root node. This step uses an evaluation function to select good nodes which will give the best overall result at each stage. UCB formula is used to traverse through the tree in the selection process. The child node that has the highest UCB value is selected.

Expansion : If the leaf node is unable to reach a decision (win or lose), another child node is created.

Simulation : It performs the playout. It plays one entire game by choosing random moves until the game has ended with decision from the child node C created after expansion. This will assign the weight to the child node.

Backpropagation : It updates the current sequence with the result after simulation by backpropagating from the child node C to the root node R.

5. Technical Overview

5.1. 2048

In all I experimented with 4 different algorithms to develop the 2048 AI agent. The algorithms I used are expectimax algorithm, alpha-beta pruning algorithm, minimax algorithm and Monte Carlo Tree Search Algorithm. In the final product I have only implemented the Monte Carlo Tree search Algorithm and the code file for Minimax algorithm and not the rest since they were implemented in different versions of 2048 game that I developed. In this section we will discuss in detail how these algorithms have been implemented in our code.

5.1.1. <u>Monte Carlo Tree Search</u>

As Monte Carlo Tree Search algorithm is based on number of playouts or simulations or roll plays i.e. the game plays a simulation with equal amount of iterations after every move. In our particular algorithm we are iterating 100 times after every move made by the agent. Also every time the agents makes a move from an unexplored game state it explores a new move that was not previously explored.

For playing 2048 game using monte carlo tree search agent we make a simulation of the game after every move to choose the best move, this simulation is played until the games has not ended, The game ends if all the tiles are filled). For each simulation we choose random moves until the game has achieved a result. For all the moves (up, down, Right and left) we make a copy of the board that runs through the depth of the game. In our case we have selected the depth of 200. If the move does not have an effect on the game state or it does not increase the score we cancel out that move. If the move helps in increasing the score we run the chosen number of simulations by making random moves.

Since 2048 has very large state space we decided that every single one of the tiles present on 0248 game boards (16 tile) can either take a value of zero (empty tile) or any power of two (2,4,8…2048). Usually while doing monte Carlo Tree search we take the action that has the best score or the move that maximized the UCB (upper confidence bound) value, which defined by the formula $UCB = Vi + C\,.\sqrt{\ln N/ni}$, where Vi is the estimate value of the node, N is number of times parent node has been visited, Ni is number of times node has been visited and

C is exploration value that scales the value of the unexplored actions. For 2048, since we generate a new tile after every move and the value of the tiles increases as we make more moves as the tiles are merged together we are unlikely to visit any previously explored node. The fact that 2048 has a very large space state, approximately upper bound of 18^{16} [16], also helps in understanding that it is highly unlikely to revisit explored nodes.

To evaluate our artificial intelligent agent we consider three features of the game. First on is the largest sum of tiles on the board when the game ends, second is the largest tile value at the end of the game and the score, third is the highest score. Most player just consider the score to be the main evaluation parameter, whereas I made an observation where I noticed that the higher the sum of board is at the end of the game the better are the chances to win if the game hasn't been over. Also another parameter which is better than score is the value of the tile at the end of the game, the larger it is the higher are the chances of winning if game wouldn.t have ended. These observations were made after realizing that score increases just by merging the tiles, so it really doesn't completely depend on the highest value of tile, but the number of tiles merged. For a good AI agent we need to consider all three parameters and the agent should achieve all of them. Below is the pseudo code for the Monte Carlo Search Algorithm and the function to run random simulations that we are using.

```
Copy board
While a move can still be made
Do choose random move and update board
Make move
return
```

```
while possiblemoves  //checks with the help of function
best move = 0;
 bestscore = 0;
        for( move) do
        makes 4 possible moves (up,down,right,left)
        for( var i  iterate number of plays)
        score = score + random(game score) //score of random move
        if score > best score
                best score = score
                best move = move
make best move
```

5.1.2 Minimax

Minimax search algorithm is a decision making rule in artificial intelligence. The task of this algorithm is to reduce the possibility of a loosing or not achieving the goal in the worst case scenario. As discussed in above sections, this rule is generally used for two player games. But to apply this in 2048 we are assuming it has two players, one is computer and other is the player. Computer tries to minimize the chances of player winning hence acting as a minimizer and player tries to maximize his probability to win. Minimax algorithm uses a game tree to decide the best move for the player who is currently playing. But in many cases due the large depth of the tree the base node or the leaf node is never reached, this doesn't allow us to get the value of gain (value of gain is the value that indicates the guaranteed payoff the opponent will have based on each players specific strategy). So in many programs depth limiter is set. We have set the depth limiter at 100.

Below is the pseudo code of minimax algorithm being used:

```
Func eval
If no nodes exist do
        Update board for free space
if depth = 0 do
        return the min value of node //heuristic value
else if player turn
        max val = -infinity; //minimum
        for  each node child
                value = gamestate.eval()
                if value>max
                max= v
        else
                max val = +infinity //maximum
                for each node child
                value = gamestate.eval()
                if value<min
                min = v
return value
```

5.1.3 Alpha Beta Pruning

As discussed in the sections above Alpha Beta Pruning is optimized minimax searched. It save computational power and time. It does so by reducing the number of nodes visited in the search tree by not searching the next node if one possible move have been found which is a better choice.

I experimented with this in other variants of the game and not the final product, so I wasn't able to compare the results derived from this algorithm with others.

Pseudo code of the algorithm:

```
Func alphabetapruning(alpha,beta,depth,node)
if depth = 0
return heuristic score
else if player turn
    for  each child node //right and left
    curscore = alphabetapruning(node,  depth -1, alpha, beta)
    alpha = curscore;
        if beta <=alpha
        break; //beta cutoff
    return alpha
else
    for  each child node //right and left
        curscore = alphabetapruning(node,  depth -1, alpha, beta)
        beta= curscore;
        if beta <=alpha
        break; //alpha cutoff
    return beta
```

```
Heuristic Score = actual score + log(actualscore) – clustering score

//clustering score is the value given to the state based on how many empty tiles are
there, lower the score the better.
```

When player one moves, player tries to maximize his chances of winning by increasing the value of gain, player does so by evaluating all the moves. The value of alpha (lower bound) gets updated every time player 1 moves, this happens until the value of alpha is not greater than the value of beta (upper bound). After this agent stops as there are better values than the one played by opponent. This where beta cuts off.

5.1.4 Expectimax Search Algorithm

As mentioned in the above sections expectimax is a variation of minimax search algorithm and it is based on the assumption that the player will might not play optimally. Unlike in minimax search algorithm where we consider the value of gain this creates a new value that we need to consider called expectimax value or expected value of the gain. So while calculating the gain we add weights to child nodes by the taking in the average of the two child nodes.

Again this algorithm was also used in previous version of the game developed and was not used in the final game because I wasn't able to make the algorithm solve the game, hence I was not able to compare the results of this algorithm with Monte Carlo or Minimax Search.

5.2 Tetris

The game has been developed from scratch using java swing components. The idea of the algorithm has been referred from Yiyuan Lee. He mentions this in his blog Tetris AI – The (Near) Perfect Bot.

For Tetris the goal of the artificial intelligent agent is to maximize the score by picking out the best move for each of the possible tetriminoe shape at that particular board state. To do so the primary target of the agent is to clear as many lines as possible as to increase the score and the duration it plays for. To achieve this target the agent has been programmed to pick the best possible moves i.e. what place to place the tetriminoe and how much they should be rotated to get the most out of the move. To help the agent it is allowed to see one upcoming tetriminoe (the next piece), so that it can decide where to place the current shape to get the most out of second shape as well. The agent gives weights to each of the possible move, for calculating weights it considers both the current and the next tetriminoe and then selects the move which has the highest weight.

The weights are given on basis of heuristics, we consider four heuristics in our approach. We take into account the height of the board, number of completed lines, number of holes in the game board and the bumpiness of the board.

The height of the board indicates the height of the highest block on the grid, or the grid height. The lower the height is the better it is, as this will allow us to have more free space and we will be able to allow more blocks to fall down the top, also this will give us the margin to correct errors. To calculate the height of the grid we calculate the distance between bottom of the game board and the highest tile in each column.

A hole is the empty space between the filled out grid. We want to minimize the umber of holes on the game board because to clear a hole we will need to clear all the lines which are above the hole. This make clearing the hole really hard. The number of holes is calculated by iterating over the grid and checking if there is a empty space between the filled height of the grid.

```
public int chhedcalc(Board b)// function to calculate number
of holes
{
    int ched = zero; //initializng holes
    {
        int x = zero;
        while (x < b.getW()) {//while within the map
         {
                int y = zero;
                  //it gets the heights within the map
                while (y < tempH.get(x)) {
                    //if map has empty spaces
                    //checks for any gaps in the filled height
                      if (b.ch(x, b.getHeight() - y)) {
                      } else {
                          ched++; //if it encounters any holes
                                  //add number of holes
                      }y++;
                  } }
              x++;
          }
      }

    return ched; //returns holes
}
```

Number of completed lines is probably the most import heuristic as it is directly proportional to the aim of the agent. We want to maximize the number of lines completed. We check for number of completed lines by checking if the line is filled and if it is, add it to the tally of completed lines.

We check for if line is filled or not by checking if there is no empty space in the game grid width.

```
public boolean checkforcompleteline(int i)
{ //iterate through the width, if width is full return true.
    return IntStream.range(zero, w).noneMatch(x -> board[i][x] == null);
}
```

Bumpiness of the grid can be define as the height difference between two columns, a bump can be pictured as imagining the filled space in columns as mountains and if there is valley between two mountains. This is a highly undesirable character of the board. These valleys indicate that there are several lines that can be cleared but are not being cleared. And if the agent makes n error and covers the opening of the valley it will make one big hole in the grid, hence reducing the chances of getting high score. We calculate it by adding the absolute differences between the two columns.

```
       bump =bump + Math.abs(tempH.get(i) - tempH.get(i + one));
```

The weight of the board or the gris is calculated by combining our four heuristics. We get the values of the heuristics and define four constants a,b,c and d. We have taken the heuristic values from[7]. The function then becomes like:

A x height + b x completed lines + c x holes + d x bumpiness

The values of a,b,c and d are as -0.55, 0.82, -0.41, -0.21 respectively. I tried experimenting with different heuristic values, in my observation these heuristics worked the best as they give more weight to completing lines. Our algorithm is like greedy genetic algorithm which focuses on making the optimally decision at every move. The pseudo code for the algorithm is below:

```
Func bestmove(board b)

Target.rotation = rotation.none //initialize shape rotation to none
Target.x =0; // x poso s shape
Taget.y = 0; // y axis pos of shape

Double bhs= -0.99999 //best score
Array list height      ;
X = teriminoe.getwidth  // returns the length of the shape based on rotation
While rotated tetriminoe can fit in map //if needs to be rotated
          Int checkheight = 0;
          While tetriminoe is in map
                    If x > 0
                    Checkhieght  = board.getheight() – height.get(0) – y;
                    Else
                    Checkheight = board.getheight() – height.get(x) – y;
                    If (checkheight = 0)
                    {continue; }
                    If ( checks if no collisions)
                              Score = Gets grid score;
                              If( score > bhs)
                                        Updates the tetriminoe length, height , rotation;
                                        Bhs = score;
Rotate shape as required;

return target //the move
```

The algorithm here decides where to place the tetriminoe and whether to rotate the block or not by taking into account the width and height of shape and then checking the height of the board and compare it with the shape to place it in the right position. Then it checks if the shape is in bounds and updates the board score.

6. Results

6.1 2048

For 2048 I made 3 different variants of games, in the beginning two were played on command line and used the alpha beta pruning and expectimax algorithm. I couldn't successfully implement expectimax to a level where it will work for the game, so I there I majorly focused on running alpha-beta pruning algorithm.

I tested alpha beta pruning by running the algorithm multiple number of times in order to try and get the accuracy of winning the game. In the beginning I tried to get the accuracy by running test 10 games. Every time the agent played 10 games it gave different accuracy. On an average it had accuracy of 84 percent. There were cases where it won all the 10 games and the next time 10 games were played it had only won 6. To get a better estimate of the performance of the algorithm I changed the number of test games to be run form 10 to 100. The accuracy of the algorithm was 87 percent.

For minimax search I ran the game multiple number of times and the results weren't the best.

Simulations	**(25) (100)**	**(50) (100)**	**(25) (200)**	**(50) (200)**
2048	No	no	no	no
Highest tile	512	512	256	512
Sum of board	968	920	568	936
Score	7104	5992	3224	7200

For Monte Carlo Tree search algorithm I tested the algorithm by running different number of simulations and tested them against the highest tile, whether or not it achieved 2048 tile and the score.

Simulations Depth	**(25) (100)**	**(50) (100)**	**(50) (200)**	**(25) (200)**
2048	yes	yes	yes	yes
Highest tile	1024	1024	1024	512
Sum of board	3622	3414	4102	3044
Score	32540	36380	36296	27216

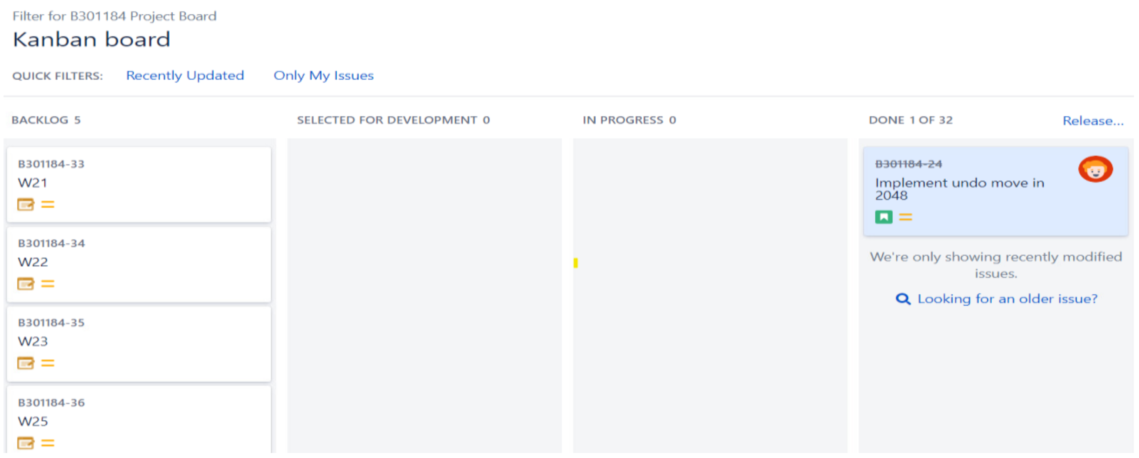
6.2 Tetris

Tetris AI was tested by updating the heuristic values. The first values used were from [6]. Then later I experimented with new heuristic values by assigning different values to change the weight that the heuristics provided. In the end I chose to have a higher weight on completed lines and height as they are the two factors that were changing results in a significant way.

7. Project Planning

The project managed by following agile methodology and updating git repository. Weekly meetings were held with my supervisor where I gave him my weekly progress. Regular updates were made to Jira. To keep the track of the issues, I used Kanban board.

The tasks were marked completed as soon as they were accomplished. If a task could not be completed it was marked not done. The evidence of the completion of issue was submitted where ever possible as well as comments.


Filter for B301184 Project Board

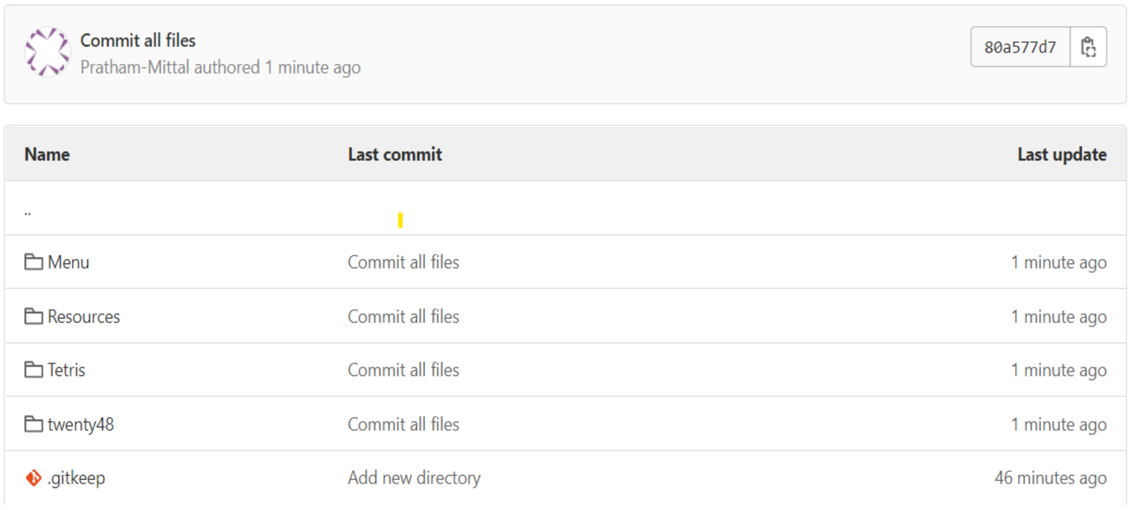
Kanban board

QUICK FILTERS: Recently Updated Only My Issues

BACKLOG 5	SELECTED FOR DEVELOPMENT 0	IN PROGRESS 0	DONE 1 OF 32
B301184-33 W21			~~B301184-24~~ Implement undo move in 2048
B301184-34 W22			We're only showing recently modified issues.
B301184-35 W23			Looking for an older issue?
B301184-36 W25			

Release...

I updated my GitLab after every version of the project was done. I committed my code at the end of each version. I updated the gitlab repository after every new version of the games were created.


Commit all files
Pratham-Mittal authored 1 minute ago
80a577d7

Name	Last commit	Last update
..		
Menu	Commit all files	1 minute ago
Resources	Commit all files	1 minute ago
Tetris	Commit all files	1 minute ago
twenty48	Commit all files	1 minute ago
.gitkeep	Add new directory	46 minutes ago

8. Conclusion

During the course of this project I learnt about different artificial intelligent algorithms. I also learnt about how to develop a game while developing 2048 and Tetris. I created AI agents to solve 2048 and tetris using different AI algorithms. For 2048 we used Monte Carlo Tree Search and to compare its efficiency we took into account the score of the game, the sum of tiles at end of the game and the highest tile after 2048 is achieved. Our AI agent achieves 2048 every time. From our results we found out that sum of board and the highest score are the most valuable features to create a good AI. I also found out that higher number of simulations resulted into better score. One disadvantage of monte carlo tree search is a expensive algorithm because it keeps on exploring other nodes to exploit and find a better strategy than the current one.

The minimax agent wasn't the most accurate as it couldn't reach 2048 tile even with higher number of depth and simulations. For our alpha beta pruning agent had an accuracy of 87 %. From the above results we can conclude that Monte Carlo Tree Search is the most optimal algorithm for 2048 AI agent.

In this project we developed a teris AI based on the idea of Yiyuan Lee genetic algorithm. While trying to improve the heuristics we observed that adding higher weight to number completed lines and higher weight to reduce the number of holes made the algorithm more efficient.

9. References

1. Louis Victor Allis, "Searching for Solutions in Games and Artificial Intellgence".
2. Tetris Wiki. 2022. *SRS*. [online] Available at: <https://tetris.fandom.com/wiki/SRS> [Accessed 29 April 2022].
3. sandipanweb. 2022. *Using Minimax with Alpha-Beta Pruning and Heuristic Evaluation Functions to Solve the 2048 game with a Computer: a python implementation*. [online] Available at: <https://sandipanweb.wordpress.com/2017/03/06/using-minimax-with-alpha-beta-pruning-and-heuristic-evaluation-to-solve-2048-game-with-computer/> [Accessed 29 April 2022].
4. *M. Szubert and W. Jaśkowski, "Temporal difference learning of n-tuple networks for the game 2048," in 2014 IEEE Conference on Computational Intelligence and Games (CIG), August 2014, pp. 1–8*
5. Kiarostami et al, "On using Monte-Carlo Tree Search to Solve Puzzles"
6. *Tsitsiklis & Van Roy (1996*)
7. Lee, Y., n.d. *Tetris AI – The (Near) Perfect Bot*. [online] Code My Road. Available at: <https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/> [Accessed 29 April 2022].
8. *Ramon & Driessens (2004)*
9. *Ashwin Ram, Santiago Ontanon, Manish Mehta, "Artificial Intelligence for Adaptive Computer Games"*
10. *Simon Algorta and Ozgur Simsek, "The game of Tetris in Machine Learning"*
11. *Ahmad Zaky, "Minimax and Expectimax Algorithm to Solve 2048"*

12. 2022. [online] Available at: <https://rockcontent.com /blog/artificial-intelligence algorithm/> [Accessed 29 April 2022].

13. SciTechDaily. 2022. *Gaming the Known and Unknown via Puzzle Solving With an Artificial Intelligence Agent*. [online] Available at: <https://scitechdaily.com/gaming-the-known-and-unknown-via-puzzle-solving-with-an-artificial-intelligence-agent/> [Accessed 29 April 2022].

14. M. Overlan. 2048 AI. [Online]. Available: http://ov3y.github.io/2048-AI/],

15. GeeksforGeeks. 2022. *ML | Monte Carlo Tree Search (MCTS) - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/> [Accessed 29 April 2022].

16. 2048?, H., 2022. *How many possible board states in 2048?*. [online] Mathematics Stack Exchange. Available at: <https://math.stackexchange.com/questions/920884/how-many-possible-board-states-in-2048> [Accessed 29 April 2022].

17. Tetris Wiki. 2022. *List of curiosities*. [online] Available at: <https://tetris.fandom.com/wiki/List_of_curiosities> [Accessed 29 April 2022].

18. Cs.swarthmore.edu. 2022. *Monte Carlo Tree Search - About*. [online] Available at: <https://www.cs.swarthmore.edu/~mitchell/classes/cs63/f20/reading/mcts.html> [Accessed 29 April 2022].