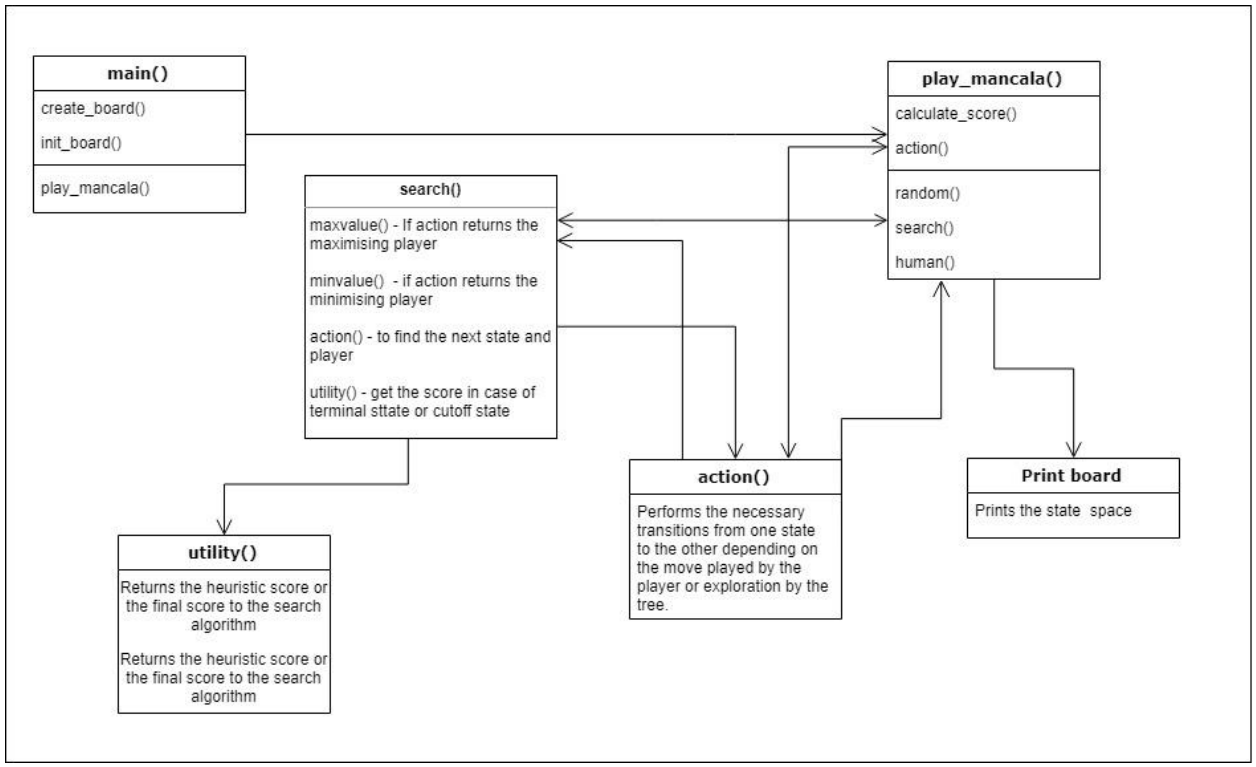
CSC 442 Artificial Intelligence

Mancala Game

Trang Nguyen, Ian Lawson, Surya Iyer

Program Design:



1. State Space

- State: We are using a 2 Dimensional array to represent every state of the game and a variable player which indicates who has to play at that time.
- Action: Each player gets to choose a number from 1 to the number of pits he has depending on whether the pit has any stones in it. Possible actions are [1, Number of pits]
- Transition model: The function action() takes care of the transition depending on the move played by the player. It also returns the new state of the game along with who has to play next. If the last stone were to fall on the store then it would return the same player, otherwise it would return the other player.

2. **Player**:
   - There are 5 possible player options the User may choose which are as follows
     - Human: This will be the user who is playing the game
     - Random: This will create a random player which will randomly choose between the possible moves
     - Minimax: This is the AI which applies the minimax algorithm and returns the most favorable move
     - Alpha Beta Pruning: This is the AI which applies the Alpha Beta pruning algorithm and returns the most favorable move
     - H - Minimax: This is the AI which applies the heuristic minimax algorithm and returns the most favorable move
     - H - Alpha Beta Pruning: This is the AI which applies the heuristic Alpha Beta Pruning algorithm and returns the most favorable move
3. **Game:**

   The game is launched by the following command:

   python program_name arg1 arg2 outputfilename

   where arg1 and arg2 are the players chosen by the user. Arg1 will play first.

   For example, to play minimax versus alpha-beta pruning, we will do:

   python mancala.py mi ab output.txt

   Initial State:

   The users will be prompted the number of pits they want on the board. After that, they will be asked to set the number of stones they want. Based on these inputs, the initial state will be set.

   For input of 6 pits and 4 stones the initial state is shown below.

```
 St   06  05  04  03  02  01  P1
-------------------------------------------------
| 00 || 04 | 04 | 04 | 04 | 04 | 04 || ra |
-------------------------------------------------
| mi || 04 | 04 | 04 | 04 | 04 | 04 || 00 |
-------------------------------------------------
 P0   01  02  03  04  05  06  St
```

After the initial state is set, the game calls play_mancala() function which first checks if the state is not the terminal state and then prompts player 1 to play the turn. Depending on the player, the game either waits for the input if it is a human the program or it will take a random value if the player is random. However, if the player is an AI, It will call the respective function passing the state of the game as the input. The function will search for the optimal path and then return the move which will reach its most favourable output. It searches for the path using the transition function action(). Once the player has chosen the move, the play_mancala() function calls the transition function action() with state and player as it's arguments. The action then makes the necessary changes in the state and finally returns the updated state and the player who has to play next. The above steps repeats until it reaches the terminal state which is when either side of the board is empty.

Note: The search algorithms are only called if the number of possible moves is more than 1.

4. **Member Contributions:**

Member	Description
Trang Nguyen	Alpha Beta Pruning
	Writeup
	Experiment
Surya Iyer	The setup of the game (human and random)
	Heuristic Minimax & Heuristic Alpha Beta Pruning
	Writeup
	Experiment
Ian Lawson	Minimax
	Writeup
	Experiment

5. Experimentation and Evaluation:

Pits	Stones	Minimax		Alpha Beta Pruning		Heuristic Minimax		Heuristic Alpha Beta Pruning		Opponent
		Final score	Time (sec)	Final score	Time (sec)	Final score	Time (sec)	Final score	Time (sec)	
2	2	4-4	2.95	5 - 3	1.51	5 - 3	1.15	5 - 3	1.51	Random
2	3	7 - 5	4.11	7 - 5	1.72	7 - 5	1.97	7 - 5	2.22	Random
3	2	8 - 4 8 - 4 8 - 4	6.17 2.19 1.63	8 - 4 8 - 4 8 - 4	3.50 1.32 2.14	9 – 3 8 - 4 8 - 4	2.61 1.53 2.54	8 - 4 8 - 4 8 - 4	1.39 2.01 1.55	Random Heuristic Minimax
3	3	12 - 6 11 - 7 10 - 8 10 - 8	30.47 36.27 20.34 21.96	12 - 6 16 - 2 10 - 8 10 - 8	23.16 15.98 15.02 16.11	16 - 2 13 - 5 10 - 8 10 - 8	2.84 2.13 2.20 2.16	14 - 4 13 - 5 10 - 8 10 - 8	2.45 2.22 1.99 4.87	Random Random Heuristic Minimax
3	4	20 - 4 13 - 11	7225 6057	16 - 8 13-11	3515.1 4249.6	17 - 7 13 - 11	5.12 3.40	19 - 5 13 -11	3.63 4.40	Random Heuristic Minimax
4	2	13-3	27576	13-3	22580	11-5	1.71	11-5	1.61	Heuristic Minimax
4	3	-	-	-	-	15-9	8.63	15-9	5.86	Heuristic Minimax
4	4	-	-	-	-	18-14	24.02	18-14	21.37	Heuristic Minimax
5	2	-	-	-	-	12 - 8	26.61	12 - 8	19.6	Heuristic Minimax
5	3	-	-	-	-	19 - 11	201	19 -11	152	Heuristic Minimax

5	4	-	-	-	-	29 - 11	475	29 -11	302	Heuristic Minimax
6	2	-	-	-	-	15 - 9	187	15 - 9	125	Heuristic Minimax
6	3	-	-	-	-	19 - 17	1210	19 -17	1109	Heuristic Minimax
6	4	-	-	-	-	31 - 17	5039	31-17	3435	Heuristic Minimax

- On multiple runs on minimax on 3 pits and 3 stone we observed that minimax seems to win all the time. This was true for alpha beta as well. The win rates is similar for alpha beta also as it is an optimized minimax. There is no improvement in the skill level for alpha beta.
- Alpha beta on most cases reduces the time taken by approximately 30% in comparison with minimax. This holds true in the case of the heuristic functions as well.
- For the case with 4 pits and 2 stones, the number of states the minimax algorithm went through is about 3,153,519,404 states whereas the alpha beta went through 2,391,987,069 states. The states-expanded-per-second with minimax was 114,357 and 105,933 for alpha beta pruning. We can assume the branching factor to be 4. If d is the depth and ϵ is the effective branching factor. Total states for a single move in minimax = $1+ 4 + 4^2+ \dots + 4^{d-1} + 4^d = (4^{d+1} - 1)/3 \approx 4^d$ .Total states for a single move in alpha Beta pruning = $1+ 4^{\epsilon} + 4^{2\epsilon}+\dots + 4^{\epsilon(d-1)} + 4^{\epsilon d} \approx 4^{\epsilon d}$

  Total states for 1st move for minimax= 3151648260

  Total states for 1st move alpha Beta = 2385720481

  $4^d = 3151648260$

  $4^{\epsilon d} = 2385720481$

  Take logarithm and divide we get

  $\epsilon = \ln(2385720481)/\ln(3151648260) = 0.987$

  We can do the same on the time taken as well on the same case.

4^d = 27576 -- Time for entire Minimax game

$4^{\epsilon d}$ = 22580 -- Time for entire Alpha Beta game

ϵ = ln(22580)/ln(27576) = 0.9804

Similarly for heuristic case

Total states for 1st move for h - minimax= 123087

Total states for 1st move h -alpha Beta = 102366

ϵ = ln(102366)/ln(123087) = 0.984

6. Discussions:

The minimax algorithm and alpha-beta pruning are not efficient enough to play with 6 pits and 4 stones in each pit. The minimax can take a lot of time for the first move. Even though the alpha beta pruning helps boost up the speed of the game, it still takes a lot of time for the first move. Hence, we were unable to run minimax on a board with 6 pits and 4 as the initial state. We then decided to do the heuristic minimax and heuristic alpha-beta pruning of depth 10 for a board of 6 pits and 4 stones. We were able to finish the game in 1.3 hours for a heuristic minimax vs a heuristic minimax. When we applied heuristic alpha beta on the same setting we could see an improvement and the time it took was slightly less than an hour.

Heuristic function used: (Difference of the number of stones in the store) + (Difference in the sum of gaps*index/3 for each player)

We were having a lot of fun when doing this project. At first, we thought minimax and alpha-beta pruning are enough for this project. However, our thought was really far from reality as mentioned above. Hence, we had other opportunities to explore the heuristic one and learn more about the algorithm. Also, working with other people is not always easy. We learnt how to collaborate and practice good coding for others to understand our code. We also learnt that discussion is necessary when doing group project. It helped us understand what everyone thought about the project at that moment and then we could come up with the best plan that might fit best for everyone. We learnt a lot about Git and also efficient coding strategies in python.