Title of project: AI for checkers

1.Group Members: Pu Zhao (pz253),Shunqi Mao (sm2784) (Ran Yao quitted, and we've only got two members remaining)

2.AI Keywords: Standard American Checkers, Minimax Algorithm with Alpha-beta Pruning

3.Application Setting: This project consists of one "main.py" file which is self-contained for running the game using whichever Python IDE you'd like. No external libraries need to be installed. Players can choose whether or not to make the first move and also choose the difficulty of the AI player.

4.Project Description

Checkers is a group of strategy board games for two players which involve diagonal moves of uniform game pieces and mandatory captures by jumping over opponent pieces. The standard version of American checkers alone is considered a complicated game with 10^20 possible legal positions in 8 × 8 board, whereas the computational complexity in higher dimensions is even more significant. According to Robson et al[1], for Generalized Checkers which are played on an N × N board, it is PSPACE-hard to determine whether a specified player has a winning strategy. And if a polynomial bound is placed on the number of moves that are allowed in between jumps (which is a reasonable generalization of the drawing rule in standard Checkers), then the problem is in PSPACE, thus it is PSPACE-complete.

In this project, we originally planned to implement several agents based on AI algorithms to solve the game of standard American checkers, which includes minimax algorithm and reinforcement learning algorithm (Q-Learning). However, due to the quitting of an original teammate and the unexpected coronavirus situation, we ended up implementing a minimax AI agent whose search depth can be manually adjusted by the player. Besides, our original plan of implementing a playable and evaluable user interface and evaluation features is successfully carried out. Players are able to use mouse and keyboard to interact with the program and choose which checker to move on the game board.

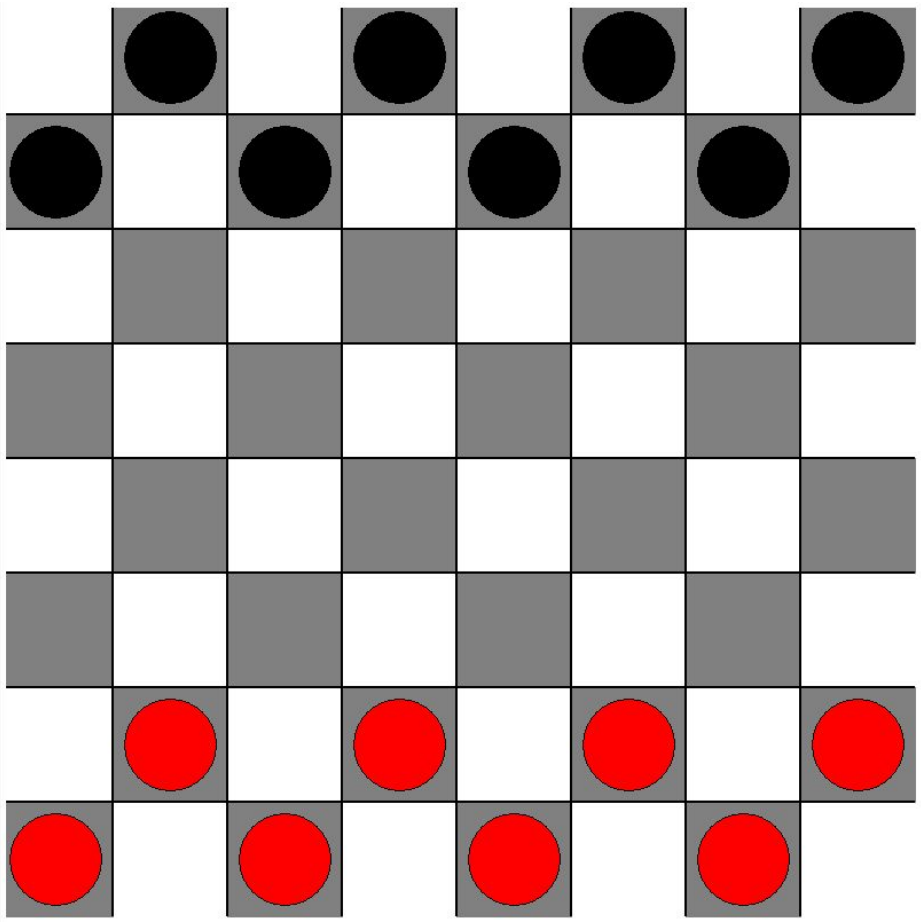

Figure 4.1: Game Board

The core aspect of this project is the implementation of minimax algorithm with alpha-beta pruning. The rules of standard American checkers allow a player to take consecutive moves if the opponent cannot make a move during his/her turn. This deviation from the minimax algorithm we learn in CS 4700 makes the implementation trickier because we have to constantly check for possible valid moves when trying to alternate between players. Another important aspect is the depth of the game tree, a.k.a. Minimax algorithm search tree. In order to create different difficulty levels for players, our approach is to let the human player choose how deep the search tree goes

before starting a game. The larger the depth, the greater the difficulty and the bigger the challenge.

Specifically speaking, there are four levels of difficulty: easy, medium, hard and infernal. In easy difficulty, the AI agent will randomly choose its next legal move no matter what the outcome might be. No algorithms are involved in this difficulty and the human player should find it relatively easy. In medium difficulty, the AI agent will use the minimax algorithm with alpha-beta pruning to search for its next move but the depth of the search tree is limited to five. Unlike regular minimax searching, we compute utility value for each node despite it being a terminal node or not. The utility value for a non-terminal node is calculated based on the number of remaining human pieces and AI pieces and the utility value for a terminal node is equal to either win reward or lose reward or draw reward. In this difficulty, human players will face a fair challenge and proficient checkers players will have a high probability of beating the AI agent. In hard and infernal difficulties, the AI agent will fairly more(with a depth of 10) or even completely explore the minimax search tree and find the optimal next move. The calculation of utility values in infernal level stays the same and there is no depth limit in this difficulty. It is extremely unlikely that a human player can beat the AI agent under this difficulty. Thus, making human players play against different difficult level AI agents is the primary way of evaluating our project.

5. Project Evaluation

The extent to which our project is successful is based on the performance of the minimax AI agent. Competence and efficiency are both important aspects of a successful game AI. In the next two subsections, we will state our evaluations of these two aspects separately.

5.1 Competence of the AI

We evaluate competence of the AI by assessing how likely it will defeat an average human player.

To start with, both our team members played against different levels of difficulty 10 times for each level and recorded the results of each game. We self-identify as average human players as we are well-acquainted with the game mechanics. To add more validity to our experiment, we anonymously invited another 14 people to play against different levels of difficulty, twice for the first three levels. Exactly 10 of them self-identify as players who are well-acquainted with the game, and we kept the data points from them. Since the skill level of us human players won't vary in a short period of time, the result of playing against AI agents can convincingly show the competence of the minimax agent.

Our game engine is pragmatic and straightforward enough to conduct these experiments. **At the end of a game, our code prints out the difference between the numbers of both players' remaining pieces and this indicates the skill parity**

between two players. Below are some visual demonstrations of the information we print out during the progress of a game:

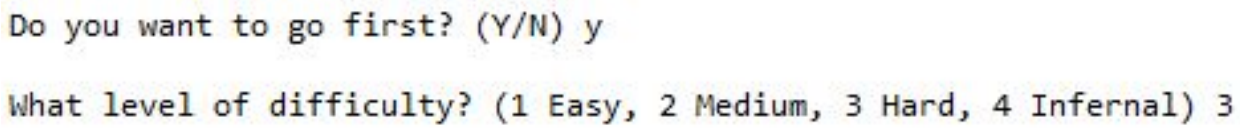

Figure 5.1.1: Beginning of our program

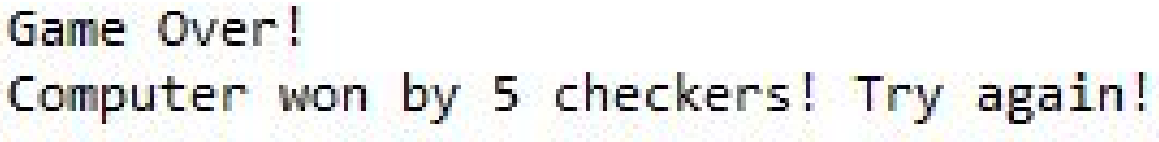

Figure 5.1.2: Information about who wins the game and by how much

For evaluation, each of our team members played against the AI agent 40 times, 10 times for every difficulty, and each of our guest players played against the AI agent 6 times, twice for the first three levels of difficulty. (Since infernal level game playing is extremely time-consuming, we did not ask them to do so.) As previously explained, the result of human playing against the AI agent can quantitatively show how competent a minimax AI agent is. Below is the table of all the statistics we obtained after playtesting:

	Easy	Medium	Hard	Infernal
Pu Zhao	Win rate: 100%(10/10) Average checkers	Win rate: 30%(3/10) Average checkers	Win rate: 0%(0/10) Average checkers	Win rate: 0%(0/10) Average checkers

	difference: +3.2	difference:- 2.5	difference: -4.1	difference: -5
Shunqi Mao	Win rate: 100%(10/10) Average checkers difference: +2.7	Win rate: 20%(2/10) Average checkers difference: -3	Win rate: 0%(0/10) Average checkers difference: -4.7	Win rate: 0%(0/10) Average checkers difference: - 5.2
10 Guest Players	Win rate: 100%(19/20) Average checkers difference: +3	Win rate: 30%(6/20) Average checkers difference: -3.3	Win rate: 5%(1/20) Average checkers difference: -4.8	N/A

Figure 5.1.3: Table about results of human playing against AI

In the table above, win rate indicates the human player's win rate and average checkers difference indicates how many checkers human players have won/lost the game (number of human player's remaining pieces minus number of AI's remaining pieces). It is not surprising that both human players are struggling against hard and infernal AI agents since our original expectation of the minimax AI agent is to significantly outperform human players. It is worth mentioning that the average checkers difference is also growing as the depth of minimax search tree increases, proving that more and more optimal moves are being made by the AI agent. In terms of AI's competence, **we conclude that the minimax algorithm is strong enough to defeat average human players.**

5.2 Efficiency of the AI

Our game is designed for running on normal personal computers, given the limited computational resources, the algorithm should be fast and efficient. We evaluate efficiency of the AI by assessing the complexity of the game search tree and the response time each move the AI used. Thanks to our implementation, we could keep tracking such information directly when each of our team members played against the AI agent.

During each move in a single game, our code prints out the number of nodes generated by minimax search and the number of pruning that happens during the search. This information helps us conduct the experiments. Below is the visual demonstrations of the information we print out:

```
Time = 0:00:03.189478
selected value 18
(1) max depth of the tree = 10
(2) total number of nodes generated = 79272
(3) number of times pruning occurred in the MAX-VALUE() = 13877
(4) number of times pruning occurred in the MIN-VALUE() = 43362
```

Figure 5.2.1: Information about minimax search tree

Our easy and medium level AI are fairly fast. But it's not the case for our "invincible" AI. Clearly, there is a significant downside of unlimited search depth for the minimax AI agent: long response time. The average response time of a single move made by infernal AI is over three minutes and more than one million nodes are being generated during the search.

```
Time = 0:01:22.955515
selected value 16
(1) max depth of the tree = 12
(2) total number of nodes generated = 2619898
(3) number of times pruning occurred in the MAX-VALUE() = 435226
(4) number of times pruning occurred in the MIN-VALUE() = 1553808
```

Figure 5.2.2: response time and nodes generated for infernal AI

We can see that even with alpha-beta pruning, there are still more than one million nodes that need to be explored, which means the computational complexity of a single AI move is nearly intractable. A response time over three minutes for a single checkers game is far beyond the acceptable range and thus relying on the minimax algorithm to find the optimal moves is definitely not a good practice. A single game with an infernal AI agent costs around ninety minutes and the entire evaluation process costs us more than twenty hours. **We conclude that searching for the whole game search tree in every single move is not an efficient way for AI running on personal computers.**

5.3 Balancing the trade-off

As analyzed above, there is a trade-off between competence and efficiency of our AI. Since exhaustive search of our infernal AI is not pragmatic, we have to reduce the depth of the search tree to a reasonable number, yet still keep the win rate of AI relatively high. Since we're researching the performance of AI against human players, the question itself is not mathematically solvable. Instead, it could possibly be done by crowdsourcing. We should change the parameter representing depth of the search tree

and publish different versions to get feedback of winning rate as well as average response time. We do not have resources for that during this project.

Fortunately, when we look back at our games against hard level AI (of a depth of 10), the winning rates are quite high (AI's winning rates are 100% for both of us), yet the average response times are 7.34s and 7.48s respectively, which are acceptable.

We also evaluate the results for depth of 8, the winning rates towards us are still 100%, and the average response times are 1.32s and 1.53s for the depth of 8.

But when we try evaluating the results for depth of 12, the average response times are around 1 minute, which is not desirable.

Thus, we propose that, to limit the depth of the search tree to a number less than 10 is a good way to balance the trade-off.

6. Acknowledgements

We would like to thank 17 friends of ours for conducting our experiments anonymously.

As an alumnus and a current student of CS4700, we would like to thank Prof. Hirsh for the enlightenment on the project.

7.References

[1] Robson, J. M. (May 1984). "N by N Checkers is EXPTIME complete". SIAM Journal on Computing. 13 (2): 252–267

[2]https://www.cs.cornell.edu/courses/cs4700/2020sp/ Our implementation of minimax algorithm comes from CS 4700 course materials

[3]https://web.archive.org/web/20190524140835/https://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html This documentation about Tkinter helps us build the GUI for our project

8.Video Presentation

The contents of two urls below are identical

GoogleDrive:https://drive.google.com/file/d/1QNUlrfrZzHWbBYu6OVxBMJwP4M-eQz1a/view?usp=sharing

Youtube:https://www.youtube.com/watch?v=FbYGGqI1GYY&t=116s