Candidate Number :
253251
19 June 2022

Intelligent Systems Techniques Report

Checkers — Human vs. AI Game

This assessment project named Checkers based on python covers how to create a human battle AI checkers game using the python module tkinter.

Code Structure

checker.py as the main program to run the game.

borad.py is the game itself, the main functionality of the program. It includes the AI which is by mini-max algorithm and some evaluation function.

pieces.py includes the class of figure, content information about position, side and graphics.

Save file saves some test files.

1 Game Internals

1.1 Gameplay

- Interactive checkers gameplay (Human user vs. Computer) **satisfied**

This is a man-machine interactive game. It is mainly a game between man and computer.

- Different levels of verifiably effective AI cleverness, adjustable by the user **satisfied**

Players can choose easy mode using white chess or difficult mode using black chess, and the black chess player has the priority to play chess.

1.2 Search algorithm

a) state representation **satisfied**

I create a class block and store the color, location, and whether the block is a king. Then I have a two-dimensional 8 by 8 fragment reference array, a black

fragment link list and a red fragment link list. Each element of the array will point to a fragment in the red or black list, or to an "empty" fragment . When moving a workpiece, it is only necessary to update the workpiece object in the list, move the reference from its current position to the new position, and set the old position as an empty workpiece. For the case of slightly increasing the update cost, you can obtain an effective method to iterate the parts on both sides and find the state of any specific point on the board for a constant time.

b) Successor function to generate AI moves **satisfied**

I used an AIMoves function to find all possible computer moves, evaluates them and selects the best ones valid, determiners if the movement from 'selected' to 'field' is valid.

c) Minimax evaluation and Alpha-Beta pruning **satisfied**

My minimax algorithm defined in this project as a recursive function in AIMoves function. I decided that 'AI' is a maximizing player and player is a minimizing one. Delete the placed part after recursively calling the minimax function.

Alpha-Beta pruning is to start from the root node and construct the pattern tree in a depth first manner. When constructing each node, the alpha and beta values of the node will be read. Alpha represents the lower bound of the best choice known when the current node is searched, and beta represents the upper bound of the worst outcome searched from the node. Since we assume that the opponent will introduce the situation into one of the worst outcomes, when the beta is less than alpha, it means that the upper limit value of the final outcome is lower than the known optimal solution from here, no matter which one is the final outcome. That is to say, it is impossible to find a better solution down here, so it will be pruned.

d) use of heuristics **satisfied**

Heuristic value = (number of AI pieces - number of Player pieces) + (kings of AI - Kings of player)*0.75. This ensures that the heuristic value is always less than the utility value, and AI players prefer the utility value because the utility value leads to deterministic results.

1.3 Validation of moves

- No invalid moves by the AI **satisfied**

Every step taken by AI is based on the current best score

.

- Check for valid user moves **satisfied**

I wrote a valid function to determines if the movement from 'selected' to 'field' is valid and a valid_moves to return a dictionary of all possible pieces moves.

- Rejection of invalid user moves, with a specific explanation given **partially satisfied**

Users can only take the route they can take, which is restricted by the rules of the game. In the GUI, you can see that the route the user takes is surrounded by red, but the route that cannot be taken cannot be clicked.

- Forced capture -an opportunity to capture an enemy piece has to be taken. If there is more than one capturing opportunity in the same turn, the user may choose which one to take. **satisfied**

I also output the probability of the player's moves on the console, and the one with the highest score will be able to capture the pawn.

1.4 Other features

- Multi-step capturing moves for the user **partially satisfied**

I also output the probability of the player's moves on the console, and the one with the highest score will be able to capture the pawn.

- Multi-step capturing moves for the AI **partially satisfied**

- King conversion at baseline (The king's row) as per the normal rules **satisfied**

- Regicide -if a normal piece manages to capture a king, it is instantly crowned king and then the current turn ends. **satisfied**

If a pawn captures a King, it will become king.

- A help facility that provides hints about available moves, given the current game state. For instance, when enabled, any squares containing movable pieces might get a different colour. More sophisticated implementations of this feature may employ the AI to make suggestions on optimal moves. **partially satisfied**

Only on the console output the probability of the player's moves, and the one with the highest score will be able to capture the pawn.

2 Human-Computer Interface

2.1 Some kind of board representation displayed on screen **satisfied**

2.2 The interface properly updates the display after completed moves (User and AI moves) **satisfied**

2.3 Fully interactive GUI that uses tkinter graphics **satisfied**

2.4 Mouse interaction focus, e.g., click to select & click to place, or drag & drop **satisfied**

2.5 GUI pauses appropriately to show the intermediate legs of multi-step moves **not satisfied**

2.6 Dedicated display of the rules (e.g., a button opening a pop-up window) **satisfied**

Player can click the Button Rule to get some information.

3 most challenging problems

3.1 I have never played checkers before this project

Checkers is very different from Chinese checkers. I've played Chinese checkers many times before, and the first time I saw an assignment about checkers I was a bit overwhelmed. I spent a lot of time learning the rules and playing against checkers in the app store many time. Although I lost many times in the fight against the bots.

3.2 The implementation of GUI is a bit difficult.

First, I took the pygame checkers course from TECH WITH TIM on YouTube. He explained the whole game very clearly, but he uses pygame, and our homework does not want us to use the pygame library. Drawing with tkinter is a bit more cumbersome than calling pygame, and I spent some time on this.

3.3 Alpha-Beta pruning is complicated

The main purpose of α-β Pruning is to optimize the MinMax algorithm. Using the α-β pruning method, when solving the MinMax algorithm for search, some nodes that are not necessary to be processed will be processed, which will result in high algorithm overhead. Compared with the MinMax algorithm, α-β Pruning adds two additional parameters, namely the lower bound α and the upper bound β, which are used to judge whether the visited node needs to be pruned. The value logic of MinMax for each node is exactly the same as the MinMax algorithm logic.

3.4 Make an effective move.

{(target position x, y), [whether there is an edible color, return color]}

If there are chess pieces in the diagonally opposite corners of the same color, break. There are chess pieces and different colors, capture them, and save the captured position,No pawns, traversal moves. If there is no next position after eating, break (such as boundary) After eating, there are chess pieces, repeat the color judgment.

3.5 Capture pieces

In the move method, swap the new position with the position of the pawn, if it is the king row, the corresponding king +1 and call the king method in the pawn. The capture method is to change his corresponding position to 0 and the number of pieces to -1. If the captured object is the king, it will directly win and trigger the king-killing. All the paintings are here, the chessboard, the pieces, the effective moves. Effective movement method, heuristic function, get all the pieces, get the state of the interface pieces (how many red and blue are in the upper half, how many are in the lower half)