AI: Checkers

Keegan Fisher ● Jackie Wong

PROJECT OVERVIEW

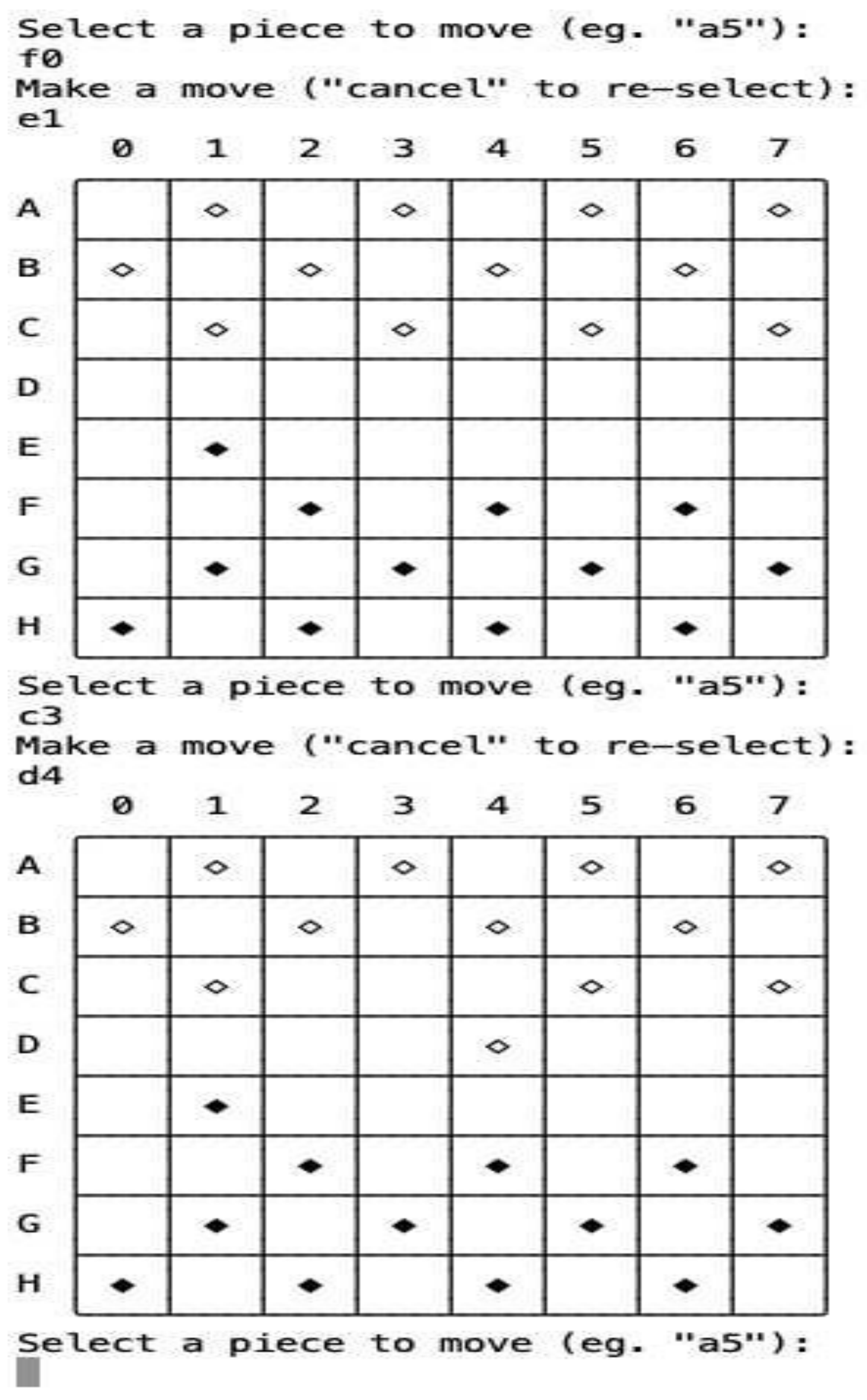
Background:

Checkers is a two player game. The players are positioned at opposite ends of the board. One player has dark colored checkers and the other player has light colored checkers. The players position their checkers as shown in the image below. The players alternate taking turns. A player is allowed to move a single checker one space diagonally from their current position. If there is a diagonal arrangement of checkers where there is one player's checker, the other player's checker, and an empty square, then the player who's checker is at the end of the diagonal "jumps" the other player's checker and moves to the empty square. The player that "jumped" the other player's checker removes their opponent's checker from the board. If a player moves their checker to the row that is at the end of the board that is opposite from them, then their opponent places one of that player's "jumped" checkers on top of that checker. The stack of checkers is called a "king". A player can move their "king" forward and backward diagonally one space from its current position. A "king" can jump its opponent's checkers. The game is played until one player has all their checkers "jumped". The winner of the game is the player who has checkers remaining on the board.

Project Goal:

The goal of the project is to develop an AI computer player capable of playing a game of checkers against a human player such that the AI player always wins. This will be accomplished through the use of a combination of AI techniques. The algorithms used will be minimax with alpha-beta pruning. The AI is to be implemented so that it will always take the most optimal move, assuming the opponent also plays optimally.

USER INTERFACE



```
Select a piece to move (eg. "a5"):
f0
Make a move ("cancel" to re-select):
e1
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
 A |   | ◇ |   | ◇ |   | ◇ |   | ◇ |
   +---+---+---+---+---+---+---+---+
 B | ◇ |   | ◇ |   | ◇ |   | ◇ |   |
   +---+---+---+---+---+---+---+---+
 C |   | ◇ |   | ◇ |   | ◇ |   | ◇ |
   +---+---+---+---+---+---+---+---+
 D |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 E |   | ◆ |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 F |   |   | ◆ |   | ◆ |   | ◆ |   |
   +---+---+---+---+---+---+---+---+
 G |   | ◆ |   | ◆ |   | ◆ |   | ◆ |
   +---+---+---+---+---+---+---+---+
 H | ◆ |   | ◆ |   | ◆ |   | ◆ |   |
   +---+---+---+---+---+---+---+---+
Select a piece to move (eg. "a5"):
c3
Make a move ("cancel" to re-select):
d4
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
 A |   | ◇ |   | ◇ |   | ◇ |   | ◇ |
   +---+---+---+---+---+---+---+---+
 B | ◇ |   | ◇ |   | ◇ |   | ◇ |   |
   +---+---+---+---+---+---+---+---+
 C |   | ◇ |   |   |   | ◇ |   | ◇ |
   +---+---+---+---+---+---+---+---+
 D |   |   |   |   | ◇ |   |   |   |
   +---+---+---+---+---+---+---+---+
 E |   | ◆ |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 F |   |   | ◆ |   | ◆ |   | ◆ |   |
   +---+---+---+---+---+---+---+---+
 G |   | ◆ |   | ◆ |   | ◆ |   | ◆ |
   +---+---+---+---+---+---+---+---+
 H | ◆ |   | ◆ |   | ◆ |   | ◆ |   |
   +---+---+---+---+---+---+---+---+
Select a piece to move (eg. "a5"):
█
```

IMPLEMENTATION

Minimax:

Minimax is a backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games in which the current-state of the board allows every possible move to be known at any time.

The algorithm consists of two "players", "minimizer" and "maximizer". Akin to their names, the maximizer will seek the maximally optimal move that can be taken from any state of the board, which the minimizer will seek the most optimal move for the opponent player.

Every board state has a value associated with it. In a given state if the maximizer has the upper-hand, the score of the board will be some positive value. If the minimizer has the upper-hand, the score of the board will be negative.

Minimax checks every single possible move, keeping track of the most optimal move (the move that will result in the highest scored board).

Implementation:

The minimax algorithm will keep track of the total turns taken and the number of pieces left on the board, upon reaching a goal-state (a board in which every one of a players pieces have been removed) the board will be assigned a value relative to these values. If the board resulted in an AI player win, the score will be positive, else it will be negative.

Alpha-Beta Pruning:

Alpha-Beta pruning is an optimization technique for minimax. It reduces the computation time by a huge factor. It cuts off branches in the game tree which need not be searched because there already exists a better move available.

Implementation:

The number of pieces and turns taken for any given board state will be used to determine whether a branch is optimal or not.