# Lab 4: Game Theory and Multiagent Systems

1 & 2 )

The first twenty lines initialize the game board, display it, and set either the player or cpu as the turn player at random. The main game loop then begins. This loop essentially processes all the events within the game. The following occurs within the game loop:

* First, check for a user initiated quit
* Next, determine if the current event is just a mouse move, in which case, update the position of the game piece and move on
* Next, we process a user move as follows:
  + Determine if move is valid
  + If so, place the game piece and update the board
  + Next, check if game has been won
  + If game isn’t over, update the turn count and turn player and move on
* After the user turn, assuming the game has not been won, the CPU takes its turn as follows:
  + CPU calls the minimax method with the current game board and a depth of 5 to find an optimal location for a move. The minimax behaves as follows:
    - First, check to see if we have reached the desired depth or if the game board is in a terminal state
    - If the board is in a terminal state, determine draw or winner and move on. If the game isn’t over and we have reached the peek depth, move on with the current selection
    - Assuming we haven’t reached the peek depth, we determine whether this iteration is a max or a min, and perform all of the possible moves, recursively calling minimax with a lesser depth and the opposite player, always returning the optimal move found so far
  + Once the minimax completes, the optimal move as determined by the algorithm is played, and then the game either ends or control is passed back to the human player

3 ) This is a classic example of a zero-sum game, as there can only be one winner and each move either advantages or disadvantages one of the two agents playing.

4 ) As currently constructed, we are recursively calling the function on every combination of moves branched off from the current game board. That means we will frequently be searching possible scenarios that do not have the potential of being our optimal solution.

5 ) SEE CODE: <https://github.com/dkStephanos/AI_Labs/tree/main/Lab%204>

6 ) The reason this moves so much faster is we essentially cut off any potential path that isn’t appealing. This is essentially the equivalent of not turning down a dead end while exploring a neighborhood to find a certain house. We aren’t necessarily searching any faster, we are just spending less time wandering down paths that can’t benefit us.