**Lab 2 Report: Hassan Naveed and Abdul Qadeer Rehan**

**Abstract**: Our goal in this lab was to make an AI that plays chess. The AI makes use of H-MiniMax to determine the next move. Alpha-Beta pruning is used to reduce computational time and improve efficiency. After testing the AI over different board configurations, we have found that the AI makes only valid and makes “smart” moves to increase chances of winning i.e. to capture the king.

**Introduction:** Chess is a 2-player game on an 8x8 board, with 16 pieces (1 king, 1 Queen, 2 Bishops, 2 Knights, 2 Rooks, and 8 Pawns). Each kind of piece has a different set of rules governing the moves it can make. The goal of the game is to capture the opponent’s king before you own king gets captured. There are 20possible moves on average for the average position from where the winning state could be many moves deep. Having a program smart enough to compete and win against humans is truly a feat. Moreover, Good performance in chess is viewed as an indicator of intellect. It is worthwhile to develop an AI for this as it gives more competition to humans enabling us to practice and improve. This makes it a good experimental exercise to learn important AI concepts.

**Formulation:** We represented the problem as an adversarial search problem. Given the current computational power in common laptops and personal computers being able look at the terminal states for each move quickly enough to play in real time seems impossible. The developed AI uses H-minimax with alpha beta pruning to choose the move most likely to lead to the terminal state the fastest. Following describes the setup of the search problem.

* **Initial State:** The initial state is an 8x8 board with pieces arranged in some valid fashion. The king of both players must be on the board, otherwise the game never starts.
* **Cut-off Test:** A heuristic function is used in H-minimax to evaluate the position of the player with for possible sequence of moves for all pieces of a player. Here the cutoff test is when a depth of 4 has been reached. This translates to 2 moves for each player. (Using a higher depth significantly increased runtime).
* **Transition Model:** One chosen piece is moved to a valid position. All other pieces remain unchanged.
* **Evaluation Function:** The evaluation function here calculates the standing of the player using Renfield values assigned to each piece. A very large number was assigned for a king, since the game terminates without the king. (Pawn = 1, Knight = 3, Bishop = 3, Rook = 5, Queen = 9, King = 100000). There is a small penalty for depth as we value a quicker capture more than one which takes a greater number of moves. The evaluation function is:

(sum of players Renfield values) - (sum of opponents Renfield values) – depth\*0.01

* **Exploration policies:** Our implementation brings forward two different explorations.
  + *Exploration Policy 1:* The first exploration policy chooses a random piece which the player can move to explore possible moves which are also explored at random. By randomizing the piece to explore we avoid getting stuck in moving the piece back and forth.
  + *Exploration Policy 2:* The second exploration policy makes use of the fact that different kinds of pieces have varying levels of importance. The importance was assigned according to the Renfield values where the King has the highest importance while pawns has least importance. To explore children states of the most important piece, this implementation chooses the closest straight-line distance of the moved piece from its original position at the current state. This is beneficial since more important pieces are likely to give better moves and allows greater pruning.

**Experiment and Analysis:**

|  |  |  |
| --- | --- | --- |
| Initial State | Number of Nodes explored Exploration Policy 1 (first move) | Number of Nodes explored  Exploration Policy 2 (first move) |
| A | 28776 | 14590 |
| B | 2465 | 4183 |
| C | 4590 | 11415 |

The number of nodes explored for both exploration policies do not provide with any conclusive information regarding which policy is better. It can be seen that for initial state A exploration policy 2 explores almost half the number of nodes explored by exploration policy 1. On the other hand, for initial states B and C exploration policy 1 explores roughly half the number of nodes explored by exploration policy 2. Therefore, we can conclude that in 2 out of 3 initial states exploration policy 1 outperforms exploration policy 2 but this is not conclusive as we see that it is dependent on initial state and in certain conditions might take longer than exploration policy 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial State | Exploration Policy 1 (first move) | | Exploration Policy 2 (first move) | |
| Optimal Action | Action Score | Optimal Action | Action Score |
| A | Q (g3 -> g8) captures q | 7.96 | Q (g3 -> g8) captures q | 7.96 |
| B | B (b8 -> a6) | 13.96 | B (b8 -> a6) | 13.96 |
| C | P (3e -> f4) captures b | 4.96 | P (3e -> f4) captures b | 4.96 |

Here we see that for all three initial states both exploration policies make the same move resulting in the same action score. While both exploration seem to find the same optimal actions, when playing the game we see that for exploration policy 1 when a pawn is moved by us the AI chooses to a move pawn, but in the case for exploration policy 2 when a paced is moved by us the AI choose to move a knight. This is because exploration policy 2 gives priority to moving the higher importance pieces first, while exploration policy 1 is more likely to explore a pawn as it half of the pieces which can be explored are pawns.

**Conclusion:**

We conclude that through a Cheese AI can be developed using adversarial search problem implemented through alpha-beta pruning. We found that the exploration policy could certainly improve the efficiency of our AI by reducing the time required to search. While we do see that the AI is able to make smart decision, there is room to increase optimality. A run with depth limit 5 instead of 4 was conducted. While it took roughly 2-3 minutes to make a move, it found a far more optimal move for initial state A. Instead of sacrificing our queen the AI decided to sacrifice a rook by moving it from r1 to r8, before finally calling a check mate by moving our queen from g3 to g7. An extension to this lab would be to implement iterative deepening where the search algorithm will run for a given amount of time, and for as many possible nodes as the algorithm has searched will return the best move possible. While in some cases this might perform worse than the current depth cut-off, the extension would guarantee completion of a move in a guaranteed time hence. The suggested evaluation function is quite simple, we can enhance the evaluation function by considering additional features such as good pawn structure and king safety.

Roles of members:

Abdul Qadeer Rehan:

* High-level implementation of the solution
* Developed coding skeleton
* Ran experimentation and testing
* Wrote the Lab Report

Hassan Naveed:

* Wrote code for detailed implementation of the solution from the coding skeleton for both AI and user input.
* Assisted writing and discussed the Lab Report