Assignment 2

Adversarial Search

1. Project description

Our second project will be to develop an AI agent for a turn-based adversarial game based on the Wumpus world example that was presented in class. The game will be played on a d by d grid, where d is a multiple of 3 (e.g. 3x3, 6x6, 9x9 ect.). Each cell is either empty or contains a pit. To place the pits you should select d/3-1 cells in each row except for the top and bottom row and mark them as pits.

Both players will have a set of d pieces which consist of d/3 Wumpuses, d/3 heroes and d/3 mages. The agent's pieces will start in the top row of the grid and the adversary's pieces will start in the bottom row of the grid. The pieces will start in alternating order from left to right starting with a Wumpus, a hero and a made (so if you have a 6/6 grid there will be Wumpus in cell 1,1 a hero in cell 1,2 a mage in cell 1,3, a Wumpus in cell 1,4 a hero in cell 1,5 and a mage in cell 1,6.

The game proceeds in a turn based manner starting with the adversary player. During each turn a player must select exactly one piece and move it one square up, down or diagonally. If a piece is move into a square containing one of the opponent's pieces then they do battle. If a hero battles a wumpus then the her shoots the wumpus and kills it. If a mage does battle with a hero then it uses its fire magic to destroy the hero. If a wumpus does battle with a mage then the wumpus will eat the mage. If two pieces of the same type do battle then both pieces are destroyed. If a piece moves into a cell containing a pit then it is destroyed and a player cannot move a piece into a cell that contains one of their own pieces. If the game reaches a state where one of the player's pieces are all destroyed then the other player wins. If both players ave no pieces left ten the game is a draw.

The game interface should display the board including all pieces and pits. You may display the board using a gui or text graphics (you may be able to reuse elements of your gui from project 1). The interface should also include a method for the adversary (i.e. the human player) to enter their moves.

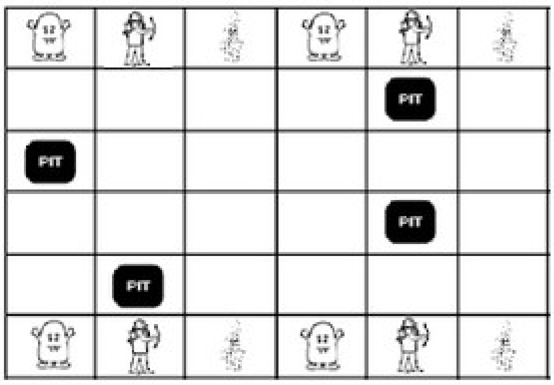

Figure 1: Example starting configuration

2. Application of heuristics to minimax

One approach to applying heuristics to minimax search is to use the heuristic to determine the order in which you search each child. At each level you can order the children of a state by their heuristic value. During max levels you can then explore the children in descending order (highest heuristic value to lowest) while during max levels you can explore the children in ascending order. The approach is shown in Algorithm 1.

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value:= −∞
        prioirty_queue:={}
        for each child of node do
            prioirty_queue.push(child,h(child))
        while child = prioirty_queue.pop() do
            value:= max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α:= max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value:= +∞
        for each child of node do
            prioirty_queue.push(child,-h(child))
        while child = prioirty_queue.pop() do
            value:= min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β:= min(β, value)
            if β ≤ α then
                break (* α cut-off *)
        return value
```

Algorithm 1: Minimax with alpha-beta pruning and heurestic ordering

3. Deliverables

1. (20 pts) Implement the interface for the game. The interface should display the board with all of the pieces, it should allow the adversary (human player) to input moves and update the state of the board based on the players move (including any captures). If should then indicate the agent's move and update the state of the board based on the agent's move.

2. (5 pts) In this problem it is clearly infeasible to search out the entire depth of the search tree and thus we will be bounding our search by the depth of d. Propose a metric for evaluating the bottom-level nodes of the tree in such a way that a higher metric value indicates a better state for the agent and a lower metric value indicates a better state for the adversary.

3. (20 pts) Implement minimax search. This algorithm should take as input a max depth and conduct the the minimax search out to that depth. When evaluating states bottom-level states during minimax you should use the metric you proposed for question 2.

4. (15 pts) Implement alpha-beta pruning for your mini-max search.

5. (10 pts) Specify a set of 5 potential heuristics that can be applied to this game and discuss the benefits of each of the heuristics. Note that the the heuristics do not need to

be admissible.

6. (30 pts) Implement the minimax algorithm with heuristics described in section 2 and apply it to this problem.