



Date handed out: 27th of April, 9 AM

Date submission due: 17th of May, 11 PM

The overall goal of this assignment is to enable you practice how to use search algorithms we learnt in the class, all the pseudo-code will be provided to you and you must follow it for your implementation.

Important: All the material included in this assignment could be included in future quizzes, and exams. I expect of both team members to understand each other's tasks and the total code despite the work load distribution.

Play Against The Computer: Poisonous Chocolate Bar Game

A) Context - Two players (player 1, player 2) have a bar of chocolate divided into $m \times n$ squares. The square in the top left corner is known to be poisonous. **The players take turns choosing one of the remaining squares of chocolate and eating it, along with all the pieces located below and/or to the right of the chosen one.** In other words, they remove the entire rectangle of chocolate whose top left corner is the piece they chose. The player who is forced to eat the poisonous piece loses the game. In order to begin playing, the game tree needs to be generated by simulating each move a player can make once it is their turn. For example, if $m = n = 2$, with the poisonous square located in the top left (represented by a black box), the game tree will look like this:

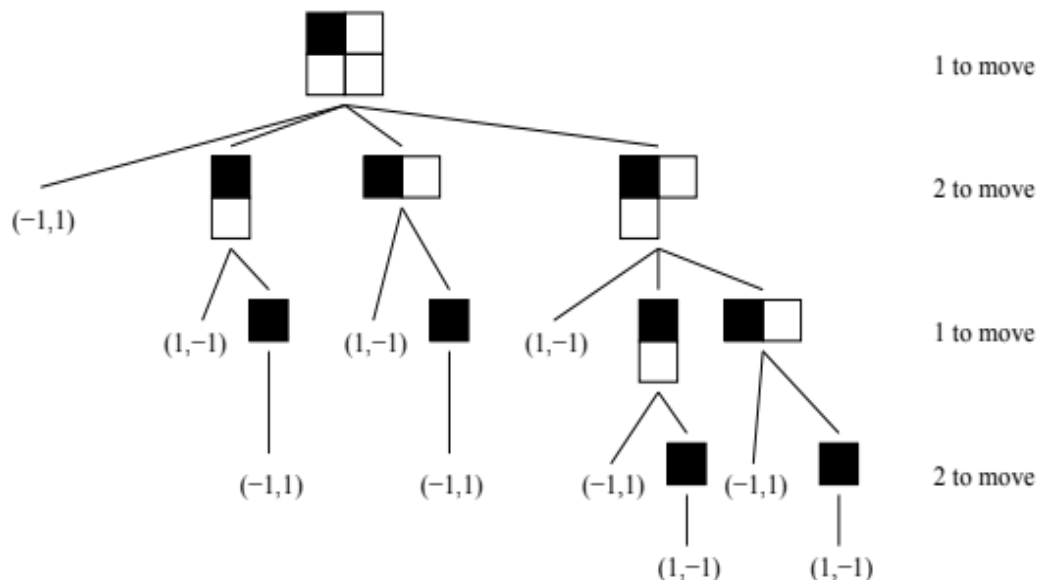


Figure 1: Game tree

In this sweet version of chomp games, the outcome is displayed at the terminal states. A result of -1 indicates a loss, while a result of 1 represents a win. This game is a two-player, zero-sum game.

There is a maximum player (**max**) and a minimum player (**min**). The maximum player is trying to maximize their profit, while the minimum player is trying to minimize it. Thus, the search tree can be seen as follows, where -1 means player 2 won and 1 means player 1 won:

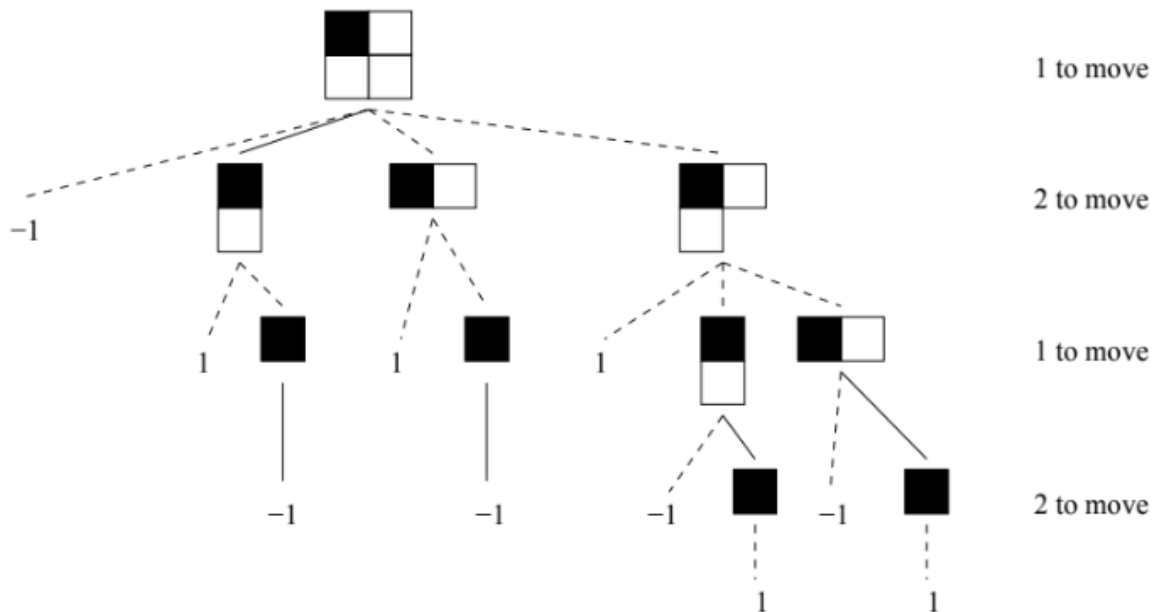


Figure 2: Finalized Game Tree

B) Tasks:

You have to create a program that simulate the poisonous chocolate bar game with the defined rules in the context. The tasks in details:

1. The game must have two modes: AI vs. Human and Human vs. Human.
2. In AI vs. Human mode, allow the human player to choose to start first or second.
3. You need to provide a proper representation of the game, including the size of the chocolate bar ($m \times n$) and the location of the poisonous square (usually located at the top left). The size of the chocolate squares may vary as well as the poisonous square location.
4. Code your assignment in such a way as to be able to show every move being made in both of the game modes.
5. Implement the Minimax Search Algorithm. We highly recommend testing your code with smaller board sizes as the AI player may take a long time to make decisions if the board is big.
6. Provide a way to measure and record the Node Count of the computer player's search. The Node Count is the number of nodes visited by the Minimax algorithm

Note: Don't spend too much time on the graphics and UI development. A command line representation is enough so long as it is understandable.

C) Some useful links:

[1] The Game of Chomp https://wiki.math.wisc.edu/images/Chomp_Sol.pdf

[2] Tricks in the game: <https://sjcinspire1.files.wordpress.com/2020/07/poisonouschocolate-answers.pdf>

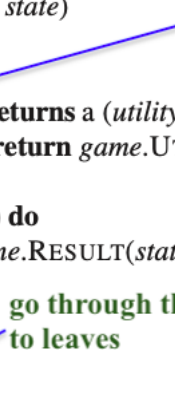
[3] You can play similar games via <https://cariboutests.com/games/chomp.php>

D) The Pseudo code:

```
function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← -∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move

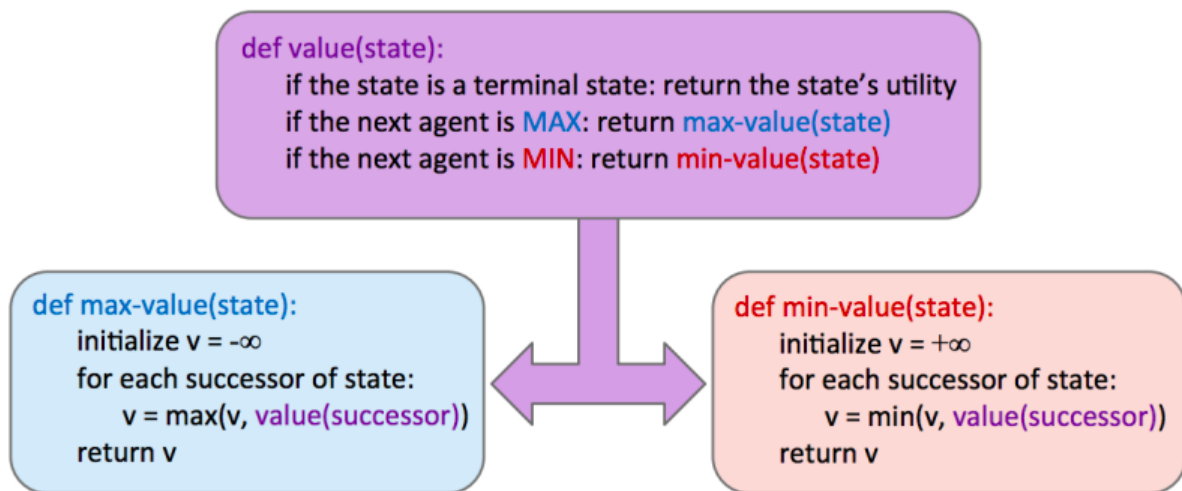
function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```



We can summarize the way minimax assigns values to states as follows:

$$\begin{aligned}\forall \text{agent-controlled states, } V(s) &= \max_{s' \in \text{successors}(s)} V(s') \\ \forall \text{opponent-controlled states, } V(s) &= \min_{s' \in \text{successors}(s)} V(s') \\ \forall \text{terminal states, } V(s) &= \text{known}\end{aligned}$$

This can be illustrated in the game tree shown in figure 2, as the states in blue are the point in which max can make a decision, the ones in red are for its opponent and the others are terminal states. In implementation, minimax behaves similarly to depth-first search, computing values of nodes in the same order as DFS would, starting with the left most terminal node and iteratively working its way rightwards. More precisely, it performs a postorder traversal of the game tree. The resulting pseudocode for minimax is both elegant and intuitively simple, and is presented below:



Programming Language Specifications

Your program can be written in C or Python.

Grading:

Grading	Points
Readme.txt (Your code should include a readme.txt that has a short description of your program and explains how to compile and run your code, please include your name, surname and id at the top)	10 (you cannot get this if you do not submit a code)
Representing the game board to players and the poisonous square to the user on the screen along the game.	15
Human vs Human mode	15
Human vs AI mode	20
Counting the number of visited nodes	25
Displaying the result of the game by the end	5
Readable code (well commented)	10