Programming Homework, CSE 440
Spring 2019, Michigan State University,

Designing Game AIs Using MinMax in Python

Sari Saba-Sadiya
CSE 440, Fall 2019

Michigan State University
sadiyasa@msu.edu

ABSTRACT. In any two player game of perfect information it is theoretically possible to identify any winning strategies for rational players. However, the search space of most nontrivial games quickly becomes unmanageable for commercial hardware. In this project we will build a MinMax AI that can play multiple games, additionally we will explore the impact of an $\alpha - \beta$ pruning optimization on the performance of the algorithm.

KEYWORDS: MinMax, Alpha-Beta Pruning, Gale Game.

Prelude

This document will be censored for academic honesty concerns, an uncensored version will be made available upon request after all projects are submitted. A censored text looks like this: [redacted].

The author is grateful to the many programmers who compiled their knowledge and made their great resources available on the web, the interested reader is encouraged to check out:

- [redacted].
- [redacted] blog post at [redacted] on some theory behind MinMax.

Finally, the Wikipedia pages for MinMax, Alpha-Beta Pruning, and the Gale-Game (Chomp) were are very helpful.

1. Introduction

In the code file for this project you will find three classes, the first two are implementations of the Tic-Tac-Toe game and the Gale-Game (chomp). In this project you will implement a naive minmax, as well as a minmax with $\alpha - \beta$ pruning optimization and test it using these two games. Each game class has the following fields:

- `ROWS` and `COLS`: The number of rows and columns for the board of this game, default is 3 for both games.
- `board`: a numpy array representing the current state of the game.
- `player`: Either 1 or -1, represents which player is the one currently playing. 1 is for the first player and -1 represents that it is the second player's turn to make a move.
- `numMoves`: The number of moves made to get from the opening position to the current board.

In general you only need to worry about the `board` field, as your minmax implementation needs to save and return a list of boards that reflect a perfect game between the two players. You are also able to call the following functions:

- `make_copy`: returns a copy of the game object.
- `move(ii,jj)`: the player who's turn it is will check cell ii,jj
- `children`: returns a list of all game objects that result from 1 move
- `result`: returns the result, always between [-1,1]. A negative result indicates a player 2 win, 0 indicates a tie.
- `final_move`: return true if the current game is at a final state.

1.1. Gale-Game (Chomp)

This game was invented by the mathematician David Gale[1]. The game is usually formulated in terms of a chocolate bar were each of two players tries to avoid eating the last square. The players in turn choose one block and ”eat it” (remove from the board), together with those that are below it and to its right. The top left block is ”poisoned” and the player who eats this loses.

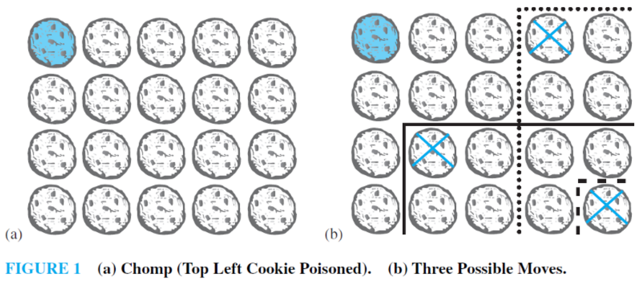
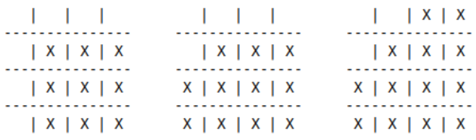

(a)
(b)
FIGURE 1 (a) Chomp (Top Left Cookie Poisoned). (b) Three Possible Moves.

FIG. 1: *The blue cookie is poisoned, and whoever eats it looses. on the right are three possible moves by the players.*

As can be seen in figure 2, at each turn a player selects cookie i, j and all of the cookies to it’s right, as well as those below them are removed. Figure **??** shows the first three moves between two players, note that player 1 win’s this game as no matter which move player 2 makes player 1 will be able to leave only the ”poisoned” cookie, forcing a win.

Interestingly, there is no known strategy for any $n \times m$ Gale-Game. David Gale himself offers a 200$ for anyone that is able to present a complete analysis of a 3D game of Chomp. If you are still confused there is a good wikipedia article and a fun online interactive implementation you can play against.



```
  |   |   |         |   |   |         |   | X | X
---------------  ---------------  ---------------
  | X | X | X       | X | X | X       | X | X | X
---------------  ---------------  ---------------
  | X | X | X     X | X | X | X     X | X | X | X
---------------  ---------------  ---------------
  | X | X | X     X | X | X | X     X | X | X | X
```

FIG. 2: *An ’X’ denotes a cookie that was taken.*

2. Min Max

The minmax class has two methods you should complete, one that uses a naive minmax algorithm, and another that implies an $\alpha - \beta$ pruning optimization. Both methods will return the reward achieved from a perfect game, the reward of a board is given by `game.result()` (see previous section) and is equal to $\frac{1}{NUM\_MOVES}$ (with negative sign if player 2 - the min player - won, and zero if the game resulted in a tie) note that you will also need to check if the board is in a final state (using `game.is_final()`) before looking at the result.

The two methods will also need to return a list of the boards (a list of numpy arrays) that result from optimal play from both players. Given this list the method `show_game()` will print out the game in more readable format.

Interestingly, it was mathematically proven that $\alpha - \beta$ pruning is a spatially optimal optimization, meaning that it is impossible to evaluate less boards and guarantee an optimal solution [2, 3]. Using the global variable `NUM_CALLS` it is possible to check how many boards were evaluated. In my own implementation a naive minmax evaluated 3500 boards while an $\alpha - \beta$ pruning was able to reduce the number to only [redacted].

It is strongly suggested that you first code the naive minmax method, the $\alpha - \beta$ pruning is basically the same code with only a minor changes.

REFERENCES

[1] D. Gale, "A curious nim-type game," *The American Mathematical Monthly*, vol. 81, no. 8, pp. 876–879, 1974.

[2] J. Pearl, "Asymptotic properties of minimax trees and game-searching procedures," *Artificial Intelligence*, vol. 14, no. 2, pp. 113 – 138, 1980.

[3] J. Pearl, "The solution for the branching factor of the alpha-beta pruning algorithm and its optimality," *Commun. ACM*, vol. 25, pp. 559–564, Aug. 1982.