Gekitai: Adversarial Search

João Sousa Miguel Rodrigues Ricardo Ferreira

April 26, 2022

Problem Formalization

State Representation

- The game state represents a specific state of the game.
- It holds information about the board, the current player and the number of markers each player has left to play.
  - Board's size and the number of markers can be customized too.
- The initial state is represented by an empty board (a matrix full of 0's) and each player marker is represented by a number - either 1 or 2.

Objective Test

- In Gekitai, there are 2 possible ways to win the game:
    1. A player lines up 3 pieces in a row at the end of their turn (after pushing).
    2. A player places all of their markers in the board (after pushing).

Operators

`move(game, position)`

- Preconditions:
  1. `game.board[position] == 0`
- Effects:
  1. `game.board[position] = game.current_player`
  2. The neighbour markers might:
     2.1 Be pushed away by 1 space from the new marker if the destination is empty
     2.2 Otherwise they stay in the same place
     2.3 Fall out of the board and be returned to their respective player
  3. `swap(game.previous_player, game.current_player)`

Game Implementation

Libraries Used

The project uses python[1] inside a conda env. Both built-in and external libraries were used.

- Numpy
  - Fast array manipulation proved to be crucial for the intensive computations made on the board's game (in particular with minimax).
- SciPy
  - The main propose of this library was the use of `convolve2d` - a powerful routine used in the implemented evaluation functions.
- PyGame
  - Used in for the graphical interface and for hadndle input events from the user.

[1] The setup guide can be found in the README file.

Adversarial Search

Algorithms Implemented

For this game we found appropritate to implement the following algorithms:

- Minimax with alpha-beta cuts (together with several evaluation functions)
- Monte Carlo Search Tree

minimax(game, ev, depth, is_max, alpha, beta)

- The minimax algorithm was the one that generated better moves overall.
- However, it takes a significantly long time when the depth value increases, due to the its exponential time complexity[2] - O(b^depth)
  - The value b represents the branching factor, which on average for a board of 6 by 6 and 8 markers for each player is 30!
- The prunning of the tree, with alpha-beta cuts, helps reducing the time it takes to generate the move, yet far from optimal since it is very difficult to order the nodes of the tree in a consistent way, e.g. many moves have the same evaluation values.

[2] Time taken to generate a move

Evaluation functions

- Evaluation functions are the key for the success of the minimax.
- Since this a zero-sum based game, positive values shows that player 1 is in front whereas negative values show the opposite.
- In the project we developed 3 different functions. Since we have 2 different ways of winning, 2 of those functions focus on 1 of the criterion over the other.
  - `markers_evaluator(game)`
    - Benefits the player with more markers placed in the board.
    - Uses `f(m,p) = -m / (m-p)` where `m` is the inital number of markers and `p` is the number of markers already placed[3].
  - `combination_evaluator(game)`
    - Benefit the player that is close to win by having 2 markers together.
    - It uses `convolve2d` provided by the SciPy library.
  - `mix_evaluator(game)`
    - Combines, as the name suggests, both functions described above.

[3] View the graph

`mcts(game, iterations, ci)`

- The Monte Carlo Tree Search algorithm, generates worst moves when compared against minimax, since it does not perform a full search.
- Increasing the number of iterations, the allows MCTS to produce better moves at the expense of taking more time[4]. However, the time increase does not have the same impact as changing the `depth` in minimax.
- One big advantage of MCTS is the fact that an evaluation function is not required.
- In our implementation, we select our nodes based on the UCB1 formula, hence the `ci` parameter. It is also important to point out how we deal with backpropagation after a simulation:
  - `If win Then reward = 1`
  - `If lose Then reward = -1`

[4]Time taken to generate a move

Extras

Features

Below are some of the features implemented:

- Customizable board sizes and number of markers at the beggining of each game.
- Various game mode, i.e. Human vs. Human, Human vs. PC and PC vs. PC.
- Algorithm fine tunning.
- Possibility of requesting an hint based in minimax or MCTS.

References

- Gekitai Rules
- IA Course's Moodle
- Artificial Intelligence - A Modern Approach (3rd Edition) by Stuart Russel & Peter Norvig
- Minimax
- Minimax by Sebastian Lague
- MCTS
- MCTS by Jonh Levine