Gekitai: Adversarial Search

João Sousa Miguel Rodrigues Ricardo Ferreira

April 05, 2022

Problem Formalization

State Representation

- The state of the game is represented by a **matrix of 6x6** and some additional data (eg. current turn).
- The **initial state** is represented by an **empty board**.
- Player's markers are represented by a boolean:
  - `False` for player 1,
  - `True` for player 2.
- An example of the game's board representation would be:

```
board = [[None, False, None, None, None,  None],
         [None, False, None, True, None,  None],
         [None,  None, None, None, None,  None],
         [None, False, True, None, True,  None],
         [True,  None, True, None, None, False]]
```

Objective Test

- There are 2 possible ways to win the game:
  1. If a player line up **3 pieces in a row** at the end of their turn (after pushing);
  2. If a player have all of their **8 markers in the board** (after pushing).

Operators

- The rules of the game are pretty simple, thus we've just defined a single operator.

`move(curent_state, position)`

- Arguments:
  1. Current State;
  2. Position - pair of coordinates.
- Preconditions:
  1. `state.board[i][j] == None`
- Effects:
  1. `state.board[i][j] = state.player`
  2. The neighbour markers might:
     - 2.1 Be pushed away from the new marker by one space if that same spot is empty;
     - 2.2 Stay in the same place if they can't be moved, i.e. there's another marker in the destination space;
     - 2.3 Be returned to the player if they fall out of the board after being pushed.
  3. `state.player = not state.player`
- Cost:
  - 1, all the moves have the same cost, possibly we want a board evaluation that takes the minimum number of moves possible.