Tablut/MALI

FMAIKR - TABLUT CHALLENGE 2022/2023

Luca Morlino – luca.morlino@studio.unibo.it
Matteo Belletti – matteo.belletti5@studio.unibo.it
Ciprian Razvan Stricescu – razvancipr.stricescu@studio.unibo.it

https://github.com/CipStr/Tablut-MALI

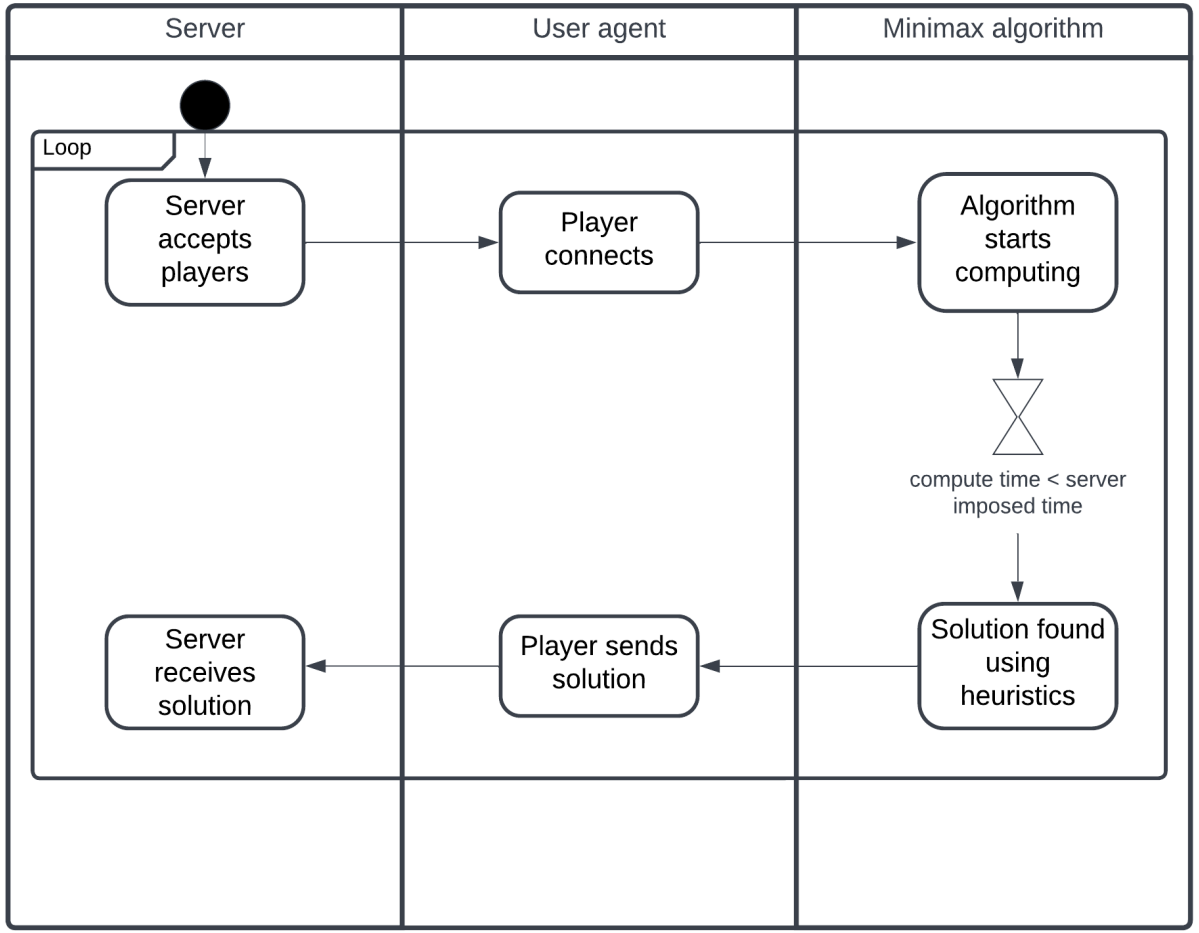
Tech stack used

- User agent is fully coded using Python.
- Python libraries used:
  - Numpy

Other built-in libraries like sys, time and json were used.

https://github.com/CipStr/Tablut-MALI

How does MALI work?



MALI is composed of two main "parts":

- User agent (class player)
- Minimax algorithm with alpha-beta cuts.

Activity diagram of the user agent.

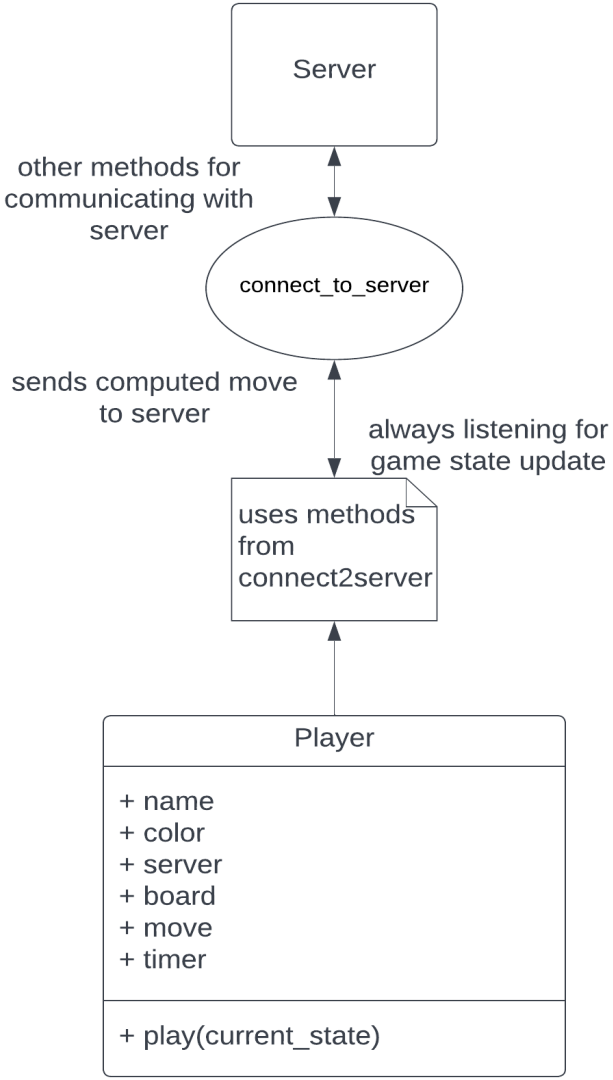
https://github.com/CipStr/Tablut-MALI

User agent

User agent is composed mainly by the class Player and manages the communication side.

The connection to the server is established by defining player name, color and server ip address and using the methods defined in connect2server.py

After receiving an update of the game state by the server the player calls on the algorithm to compute its next move.



https://github.com/CipStr/Tablut-MALI

Minimax: alpha-beta cuts

```
if player == "black":
    maxEval = -np.inf
    best_move = None
    # generate all the possible moves for the WHITE
    moves = board_class.generateMoves(board, player)
    # for each move, generate the new board and call the minimax algorithm recursively
    for move in moves:
        new_board = Board.Board()
        new_board.setBoard(board)
        new_board.setBoard(new_board.movePiece(move))
        tmp_eval, tmp_move = minimax(new_board.getBoard(), depth - 1, "white", alpha, beta, preceding_moves, timer)
        maxEval = max(maxEval, tmp_eval)
        alpha = max(alpha, tmp_eval)
        if maxEval == tmp_eval:
            # if this move is in the player's best moves, then discard it
            if move not in preceding_moves:
                best_move = move
        if beta <= alpha:
            break
    if best_move is None:
        best_move = moves[0]
    return maxEval, best_move
```

It’s a recursive backtracking algorithm that creates a decision tree and returns the best move for the player.

The initial epth when starting is three but if it reaches an internal timeout state it changes to a lower depth, in order to find a faster but «weaker» solution.

Minimax is also powered by some heuristics, defined in heuristics.py, which define a score that it uses to select the best move.

The alpha-beta cuts version prunes all the branches that we don’t need to reach for finding the best solution.

https://github.com/CipStr/Tablut-MALI

Heuristics used

- Black pieces:
  - If they surround the king it wins.
  - If a piece can eat or block a white piece its score increases.
  - Increase the score based on the amount of pieces near the king.
  - If it’s a potential good move its score increases.
  - If the amount of black pieces is greater the score increases.
- White pieces:
  - If the king can escape at the edge it wins.
  - If the king is near throne it’s usually safer so the score decreases.
  - If the amount of white pieces is greater the score decreases.
  - If it’s a potential good move its score decreases.
  - If a piece can eat or block a black pieces its score decreases.

https://github.com/CipStr/Tablut-MALI