Board Games with some Game Theory Cleverness

Julian Stampfli, Marc Rey, 2019.

Context

This is a console app developed during the course "Game Theory" by J. Eckerle at University of Applied Sciences Berne in Spring 2019.

The goal of this app is to implement Game Theory strategies in a play of a board game. The user plays either against another human or against a computer opponent who may either choose his next move randomly or by strategically evaluating his best move.

There are two board games that can be played:

- Pawn Chess / Bauernschach
- Alquerque / Original Checkers

Board Games

Pawn Chess ("Bauernschach")

- The game is a simplification of Chess such that there are only the pawns and no other figures.
- Play happens on a square board of 4 to 8 fields in width and height respectively.
- Initially each player has as many pawns as the board is wide, placed in the second row closest to him.

```
- - - - -
X X X X X
- - - - -
- - - - -
O O O O O
- - - - -
```

- Players move turn by turn and one pawn per turn.
- Pawns may move by:
  - advancing straight ahead 1 field, if this is not blocked.
  - advancing straight ahead 2 fields, if those are unoccupied and the pawn has not been moved yet.
  - advancing straight ahead 1 field then left or right 1 field, if the target field is occupied by the opponent. The opponent's pawn is killed.
  - stepping left or right 1 field, if this field is occupied by the opponent and the occupying pawn has moved once. (en passant) The opponent's pawn is killed.
- The game is won by the player who manages first to get a pawn to the last row on the opposite of the board.

Alquerque ("Original Checkers" / "Urdame")

- The game is a simplification of the various versions of Checkers played today.

- Play happens on a square board of 3 to 7 fields in width and height respectively. The board has to be odd.
- Initially the board is filled from the top left and bottom right corner to the middle until the pieces meet in the middle. This field is left empty to enable the first move.

```
X X X X X
X X X X X
X X - O O
O O O O O
O O O O O
```

- Players move turn by turn and one pawn per turn.
- Pawns may move by:
  - moving 1 field in any direction including diagonally, if this field is unoccupied.
  - moving 2 fields in any one direction including diagonally, if the first field is occupied by the opponent and the second field is unoccupied. The opponent's pawn on the first field is killed. This way
- The game is won by the player who's opponent cannot move:
  - either because he has no more pawns
  - or because he has no possible moves, that is he is blocked.

AI Behavior

There are various strategies available to choose the best move:

- Random
- Simple heuristic
- Complex heuristic
- Monte Carlo

All of them are implemented for Pawn Chess. Only random choice, Monte Carlo and the simple heuristics are implemented for Alquerque.

No matter which heuristics is chosen, should the depth be enough to win the game, this path always returns a number that is higher than all of the generated numbers by any heuristic.

Random Choice

The random choice AI takes a random move for a random pawn. This way, it moves in unpredictable behavior but will most likely lose.

Simple Heuristic (only Pawn Chess)

Alpha-Beta-Pruning is performed at a deepness of 4. After the depth has been reached the pawns of the player are compared to the pawns of the opponent. The player optimizes for most pawns.

Complex Heuristic (only Pawn Chess)

Search is performed by Alpha-Beta-Pruning at a deepness of 4.

There are various considerations that can influence the heuristic evaluation of the board:

- Total pawns:
    - The more pawns one has compared to the number of pawns the opponent has, the better.
- Position of one pawn relative to the game board:
    - The further advanced a pawn, the better.
    - A pawn that has an path to the finish line, that is not blocked by the opponent,
        - is great.
        - is less great, if there is an opponent's pawn in a neighbouring line up ahead and thus could kill our pawn.
- Position of one pawn relative to our and opponent's other pawns:
    - A pawn is strong if ...
        - he can kill another pawn now
        - without being killed in the next move
        - without another being killed in the next move
    - A pawn is safe if ...
        - he cannot be killed in the next move
        - he can be killed in the next move but the killer himself can be killed in the move after. (both players loose 1 pawn each)

However, in order to reduce the negative performance impact, this heuristic evaluation function cannot implement all those aspects. It includes these:

- The more pawns one has compared to the number of pawns the opponent has, the better. This is given by the simple difference in pawns each player has.
- The further advanced a pawn, the better. Therefore, the value of each pawn is the index of its row as seen from its owner.
- A pawn that has an path to the finish, that is not blocked by the opponent, line is great. Thus, the value of the pawn is multiplied with a factor.

The various considerations might conflict somewhat and sometimes lead to undesired side effects. The evaluation function accounts for some of them:

- To avoid pawns always jumping 2 fields as their first move, the row values start at -1. Note that the pawns start in the second row as seen from their owner.
- To avoid mere pawn count to dominate the pawns' progress, the value of the pawn given through their advancement is squared.

Furthermore, it is assumed, that the possibility of a pawn getting killed is taken into consideration implicitly by Alpha-Beta-Pruning and the calculation of the surplus of pawns in the heuristic evaluation function.

The performance impact of evaluating the board is still considerable. For each leaf in the game tree the whole board has to be traversed. Even worse, since checking for unblocked paths one has to traverse the board from both sides. To keep the latter issue minimal, traversal is done backwards from the furthes line respective to each player. This eliminates the need to evaluate the remaining column for each pawn separately.

Monte Carlo

Alpha-Beta-Pruning search is performed at a depth of 4. After that, each path is extended by one more level, and from there the ai finishes the game by choosing random moves for each player. The heuristics then takes all the won games. This way, the leaves can be compared by seeing how many random paths were won. To increase the quality of the heuristics, it would be better to play games to the end multiple times from each leaf. We decided against that because of performance issues.

App Architecture

Overall Architecture

The application is based on the idea of MVC. Code is grouped in modules accordingly. The models just represend enumerables or data transfer objects. The view is responsible for rendering to the console.

Quite a few classes inherit from "abstract" parents in order to share code and provide different implementations.

User and Machine Actors

All action, be it by the user or by a machine player, is contained in controllers. Where a controller provides human interaction it contains a loop working its way through a simple state machine. Each user input leads to a new state being set. A new state being set leads to new calculations or user prompts.

Note that there is also a "state" as known in frontend architecture. It contains the current data of the application, e.g. user choices and the current game.

Game Engines

The game logic itself is to be found in the module "games". It has no dependencies on controllers or the gui and is therefore relatively loosely coupled. The engines get instantiated and called by controllers only and provide methods to interact with. Foremost, they contain the board, representing the current state of the game, and are the only one to manipulate the board.

Machine Actors and Strategies

Since choosing the next move is a behaviour of the actor in the game and them being represented by controllers, the implementations of the strategies are to be found there too.

App usage

Running the application

The app is built with Python version 3.7.

Start by running the app.py in this same directory. No command line parameters are to be passed.

User Interaction

As a user, interact with the app by entering one of the commands shown at each step.

You may abort the current game or the app as a whole at each step.

Gameplay

The application will guide you through the game.

Note that:

- On the board, you as a user are placed on the bottom side while the opponent is placed on the top side.

```
opponent (human or machine)
X X X X X X
X X X X X X
X X - - O O
O O O O O O
O O O O O O
user (human)
```

- The user is always a human player.
- A human player is presented first with a list of pawns that can be moved and, after selecting a pawn, with the moves it can take. Those moves are also hinted on the board as `?`.
- The opponent can either be a human player or a machine player with a strategy. You will be promted to specify which strategy.
- The history of steps in the gameplay is listed. This hints what a machine opponent has done.

Please be patient for the machine player to asses its best move. The calculation is rather time consuming.

Critical Reflection

Challenges

- It is hard to make a clever machine player due to various reasons, foremost:
  - The game tree expands fast. Thus it is a balance between having the user to wait a long time for the machine opponent to be ready on one side and a deep game tree on the other.
  - "Designed" heuristics, in contrast to Monte Carlo, require the developers to find and optimize a proper evaluation function either by thinking and trial and error or by analysis of vast amounts of data gathered. Due to the lack of data, we had to accomodate with ideas based trial and error.
- Once you get the knack of it, working with Python is nice. However, to keep your code readable and maintainable, we find it better to use OOP and well-accepted code style rules.

Ideas for further development

- To find better heuristics, two machine players should be able to compete and data should be collected and persisted.
- There is currently no backwards navigation through the app or in the game. I.e. you cannot undo an action taken.
- Alquerque is missing a complex heuristic evaluation function.
- Multithreading might help to improve performance.

- There are probably a few well established opening moves or sequences. To avoid performance at its worst, i.e. at the beginning of the game play, they could be implemented and chosen from. Thus render having to perform the evaluation for these moves superfluous.