COMP 6721
Aymeric Grail
#26810020
aymeric.grail@gmail.com

Project Report

Including AI strategy

General notes

Sources

The game has been developed in Java 1.8, using IntelliJ
To start the game, simply run the “controller.StartGame” class.
The README.md file contains information on some of the game properties and other information useful for the A.I.

Playing

Enter the name of the players. Any player whose name starts with “AI ” will be an automated player (eg: “AI Google Deep Mind”).
The game allows 2 AI player to play against each others.
The game is over when:

- Any player successfully managed to play a polarized ladder
- Or all positions on the board have been played

If no polarized ladder is present when the game is over, then it’s a tie.

Automated Tests

Game Controller:

A test class covers the GameController, ensuring the manual game works. That is, it tests that players are able to play on the board, and that winners are properly detected.

A.I.:

A simple test class covers 2 behaviors expected from A.I.:

- one test method to verify the defensive behavior
- one test method to verify the aggressive behavior

The Program

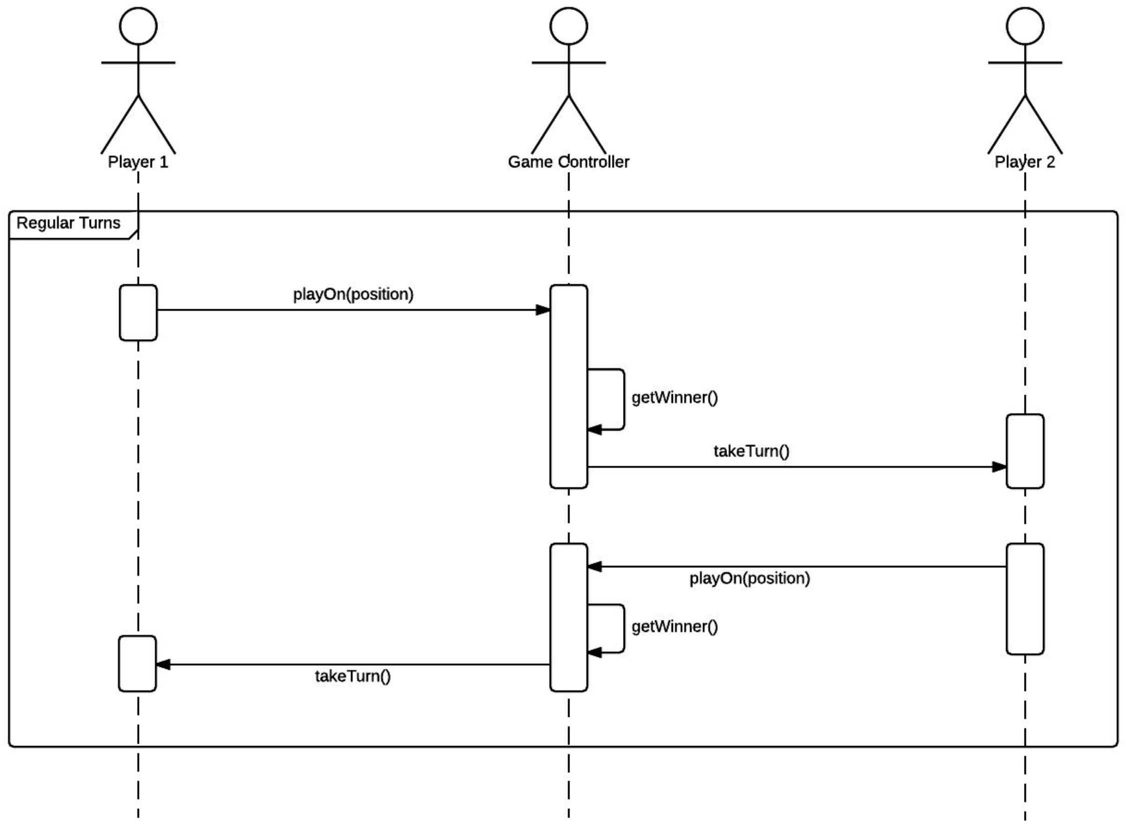
UI:

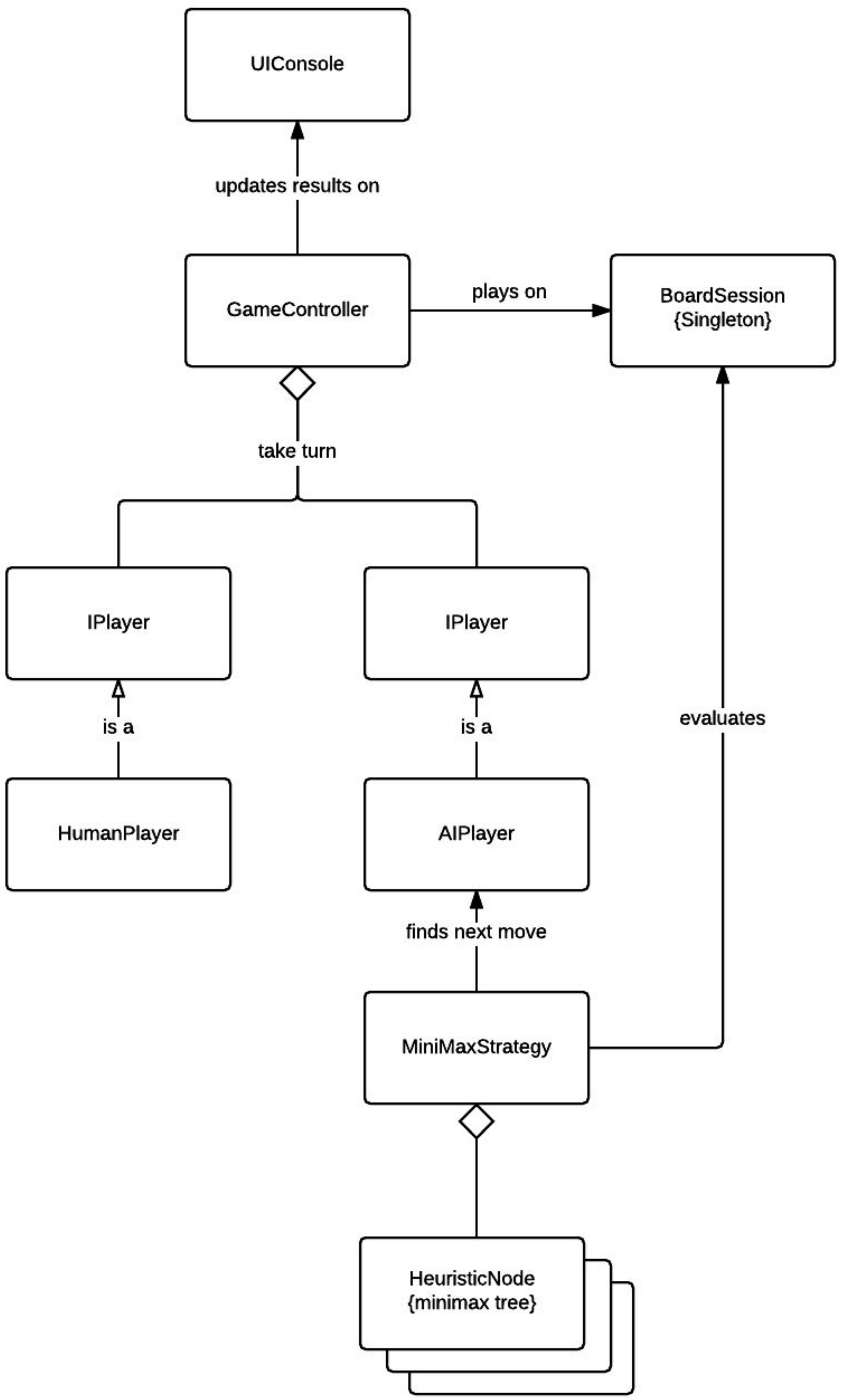
The game has a very simple console UI. The console is hosted by the GameController to update the board, and is also passed to HumanPlayer so it can interacts with the users when they take turn.

GameController:

The game controller is in charge of the game. Its responsibilities are:
- Set up the game (the board, the players…)
- Coordinate player’s actions
- Validate the player’s decisions
- Determine the winner
- Send the results to the UI

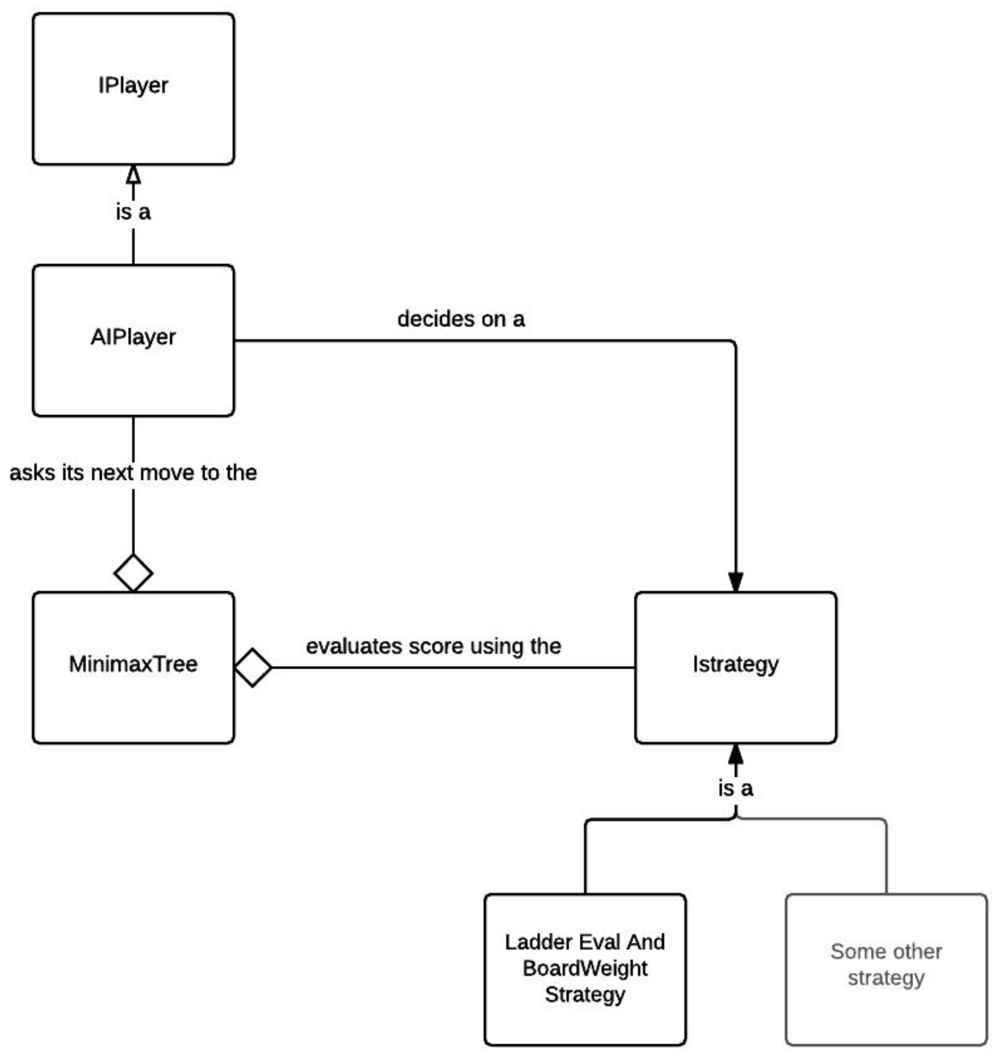
Architecture:




UIConsole
updates results on
GameController
plays on
BoardSession {Singleton}
take turn
IPlayer
IPlayer
is a
is a
evaluates
HumanPlayer
AIPlayer
finds next move
MiniMaxStrategy
HeuristicNode {minimax tree}

AI:

The AI is hosted by the AIPlayer, and is isolated in its own AI module.
When setting up an AIPlayer it decides on an AI strategy. At the moment, it exists only one implementation of the IStrategy, but other implementations could be created. The the strategy could be decided on at runtime, depending on the difficulty level chosen by the player.
See the section on the heuristic for more details about the AI strategy.


IPlayer
is a
AIPlayer
decides on a
asks its next move to the
MinimaxTree
evaluates score using the
Istrategy
is a
Ladder Eval And BoardWeight Strategy
Some other strategy

The Heuristic

Minimax

The minimax tree is set at a depth of 3.

The heuristic used by the A.I. is such as:

- Any goal state encountered by max is worth +1.000
- Any goal state encountered by min is worth -1.000
- When a goal state is encountered, its score is immediately returned
- If no goal state is encountered, then every leaf is evaluated depending on the strategy injected to the minimax tree.

The current strategy (LadderEvalAndBoardWeightStrategy) uses 2 scoring methods in order to determine a given position's score. One is dynamic, and depends on the state of the game, the other is static and allows the A.I. to determine what's the best position to play in case of a tie in dynamic scoring (for example when playing the very first move).

1st method: Dynamic Scoring

The goal of this scoring method is to determine what's the best position to play on, given the current game state.

At each level of the minimax tree, and for each of the nodes deployed, this strategy evaluates how close the newly deployed state (new position played) is from being a goal state.
To determine how close it is from the goal state, it evaluates all potential ladder the played position can be part of, such as:

- If the opponent occupies any position on the ladder being evaluated, then this ladder is worth 0
- If the opponent occupies both counter-polarization position for the ladder being evaluated, then this ladder is worth 0
- If the opponent occupies one of the 2 counter-polarization positions for the ladder being evaluated, then this ladder's score is divided by 2
- Finally, it gives 1 point for every position occupied on the ladder (apart from the position being evaluated)

Then it returns the sum of all the potential ladder's scores in order to determine the node's dynamic score

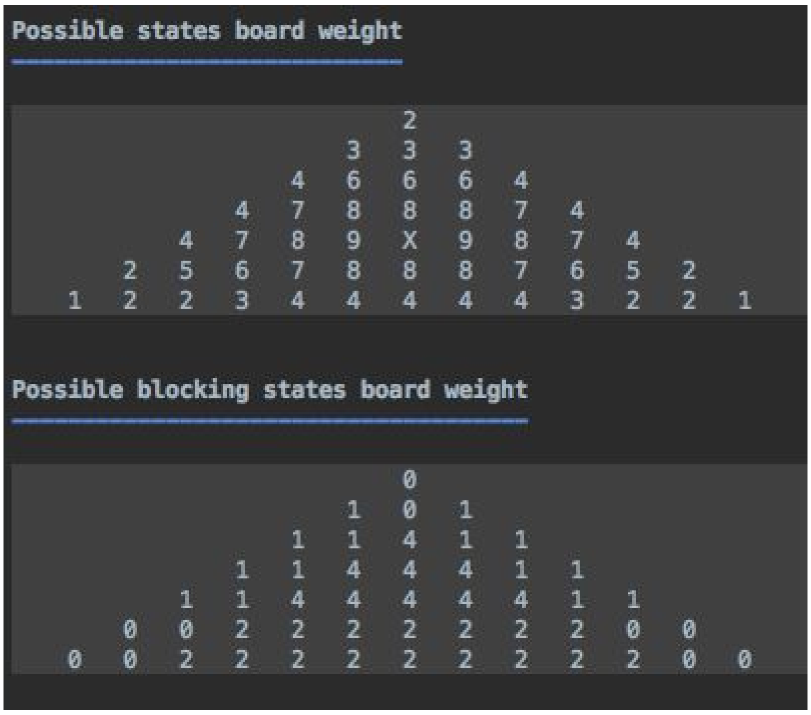
2nd method: Static Weight Scoring

Every position on the board has manually been given 2 static scores such as:

- Possible win score: number of polarized ladder a given position can participate in. Any position can participate in at least 1 polarized ladder, but no more than 10.
- Possible counter score: number of polarized ladder a position can counter. Only positions at the center of the board might counter an opponent's ladder

The total static weight score of a state is the sum of all the AI Player's positions win and counter score.
For example, if AI plays 'X' and only occupies positions {(1,7) (2,6)} then its score would be:

$$\begin{aligned} f(X) &= (2+0) + (3+1) \\ &= 6 \end{aligned}$$



```
Possible states board weight
----------------------------

                           2
                       3   3   3
                   4   6   6   6   4
               4   7   8   8   8   7   4
           4   7   8   9   X   9   8   7   4
       2   5   6   7   8   8   8   7   6   5   2
   1   2   2   3   4   4   4   4   4   3   2   2   1

Possible blocking states board weight
-------------------------------------

                           0
                       1   0   1
                   1   1   4   1   1
               1   1   4   4   4   1   1
           1   1   4   4   4   4   4   1   1
       0   0   2   2   2   2   2   2   2   0   0
   0   0   2   2   2   2   2   2   2   2   2   0   0
```

Manually calculated static weights for every positions on the board

Possible Improvements

One correct but odd behavior of this AI is that it will play the last position on the board if it determines it cannot win the game. The reason for this is that in case it is assured to lose (at least 2 states are only one step away from victory), all states will end up in a tie score of -1000 and therefore the AI will simply play on the last state evaluated.
That does not take into account that humans may make mistakes and not play on a winning state on the next turn, and could be improved to always attempt to counter an opponent's move, even if the opponent is assured to be able to win.
Another improvement would be to work on the score calculation performance in order to be able to evaluate the tree 4 levels deep instead of 3.