**Artificial Intelligence I**, *prof. Pasquale Caianiello 25.10.16*

Homework 2: implementing minimax in two-player games. *Presented by: Yuna Frolov*

Gomoku

**Gomoku** is an abstract strategy board game. Also called **Gobang** or **Five in a Row**, it is traditionally played with Go pieces (black and white stones) on a 15x15 go board; however, because pieces are not moved or removed from the board, gomoku may also be played as a paper and pencil game. This game is known in several countries under different names. It originated in Japan during the Heian period.

Black plays first if white did not just win, and players alternate in placing a stone of their color on an empty intersection. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally.

*Official rules:*

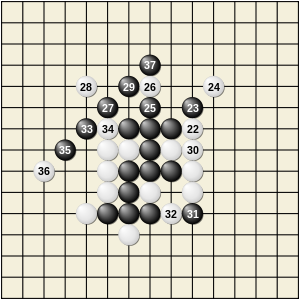
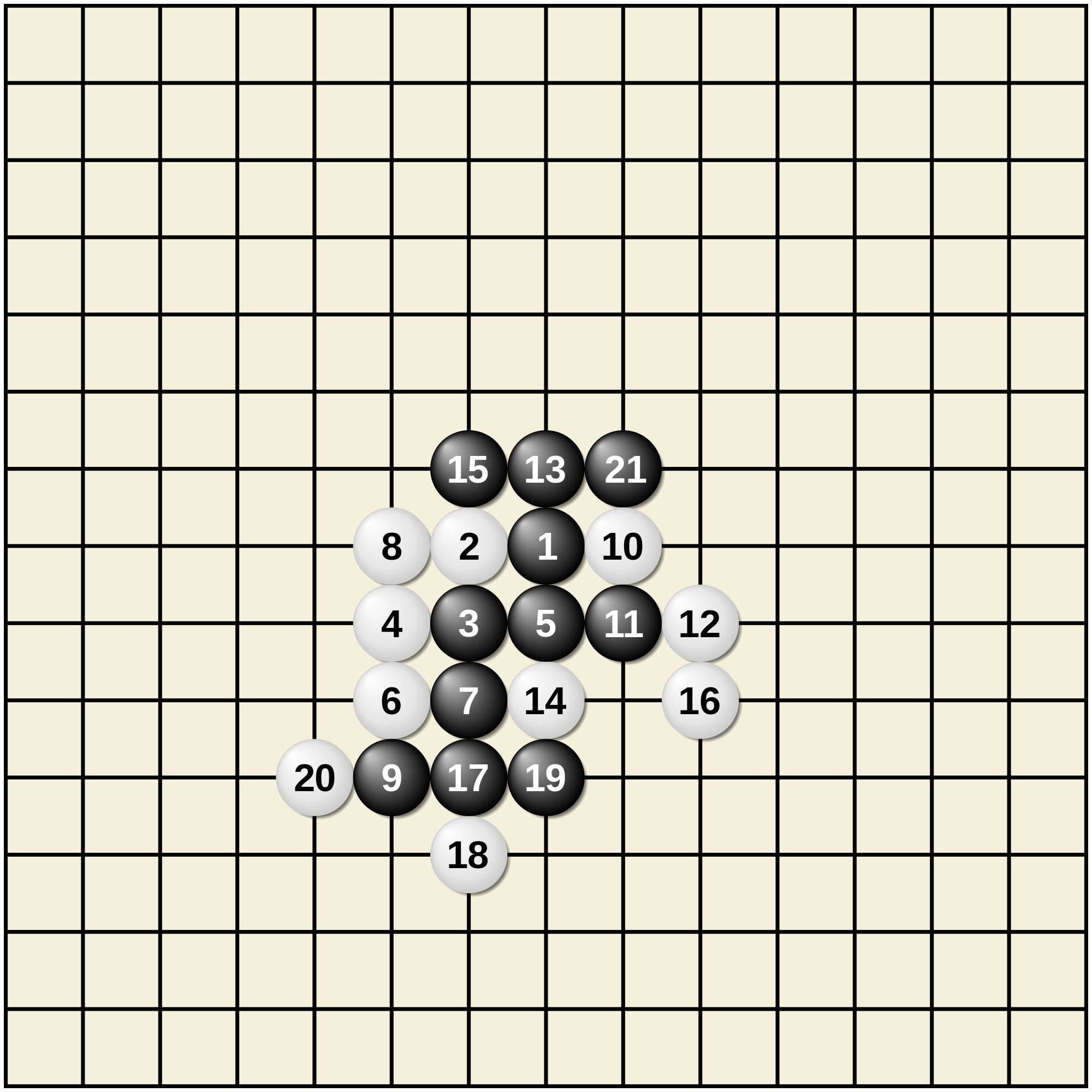
Besides many variations around the world, Swap2 (based on **swap** from [Renju]) is currently adapted in tournaments among professional players. The first player places 3 stones (2 black 1 white, if black goes first) on the board, the second player has the choice to take black/white or to place 2 more stones to change the shape and let the first player choose color. This is essentially a slightly more elaborate pie rule. Swap2 solved the low complexity problem (see L. Victor Allis) and offers more fairness to the game. Like other rules and variations, 100% fairness can be reached by playing 2 alternating games for each point.

*Analysis:*

People have been applying the artificial intelligence technique on playing gomoku for several decades. In 1994, L. V. Allis raised the algorithm of proof-number search (pn-search) and dependency-based search (db-search), and proved that *when starting from an empty board*, the **first player has a winning strategy** using these searching algorithms.

However, neither the theoretical values of all legal positions, nor the balanced rules used by the professional gomoku players have been solved yet, so the topic of gomoku artificial intelligence is still a great challenge for computer scientists, such as the problem on how to improve the gomoku AI-s to make it more strategic and competitive. Nowadays, most of the state-of-the-art gomoku AI-s are based on the alpha-beta pruning framework.

[Explanation taken from Wikipedia – [Gomoku](https://en.wikipedia.org/wiki/Gomoku)]



**\_\_init\_\_.py**

Based on the last paragraph, I have chosen to implement the **quiescence search** variation of *minimax algorithm*, using the standard gomoku (and not swap2). I should achieve a 100% win rate.

I have worked with 3 python files:

**Gomoku.py: (main)**

This file is used as the main file. The first default moves are played, and the, while there is no winner, black and white take turns placing their symbols on the board, making their best evaluation of minimax according to the quiescence search provided in the ‘nextMove()’ function in ‘heuristic.py’. Quiescence search is an algorithm typically used to evaluate minimax game trees in game-playing computer programs. It is a remedy for the horizon problem faced by AI engines for various games like chess and Go.

The main is launching the game with the standard variables – size 15x15 board, 5 needed connection in a row, and a time limit of 60 seconds (to stop the program if a computation is too long).

**Board.py: (game)**

This file is used to initialize the Board and the Color classes.

The Color class contains the basic functions and also a function that *swaps* players and returns the *opposite* of the current player.

The Board class is used also as a game initialization, and contains the functions:

* turn - returns the color object of whoever's turn it is
* \_valid\_move - check if the move represents a valid space
* \_inBoard - is the coordinate on the board
* move - execution of a move
* \_checkPath - return path to winning state of the winning color
* checkWinningMove - if the last move won the game - all appropriate msgs are changed and win is announced

**heuristic.py: (heuristic)**

This file implements the search and the heuristic function (alongside other helping functions).

The file contains these functions:

* Firstmove, secondmove - first and second pre-determined moves
* Heuristic - evaluates each position on the board with significant weights. the more connected colors the attacker has in a row - the more the position next to it weighs the function prefers to make a winning move than a defending move
* evaluate\_position - this function evaluates the attack power of the other player (max of min).the importance of a position is rated as: it's attack power + it's defense power, which is attack power for you + attack power for them.
* attackArea - return the area that may be attacked by opponent from (y, x)
* moveLimit - returns the move limit as valued by evaluate\_position
* justBestMoves - return the best valued moves by evaluate\_position
* nextMove - using Quiescent search - returns a move where search predicts a win or best move
* depth\_1 , depth\_2, depth\_3, depth\_4, depth\_5 - depths used in quiescent search above

The idea is that the next move is made using the search – exploring the board to depth 5 – each player checks what could be the next moves that are 5 deep – using the minimax method to decide which move is the best possible move to make, according to the evaluations. These evaluations take in account the weight of each possible position on the board, according to their attack and defense potential, preferring to make the winning move rather than protect from a losing blow after the possible win.

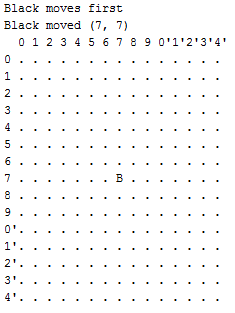
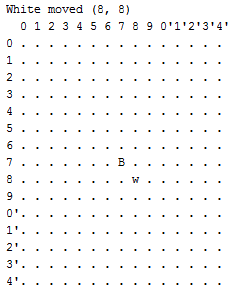
With the basic set-up: standard 15x15 clean board and the first move of each player – this algorithm is always providing the win towards the first player that made the move, in this case the BLACK. This was expected from the knowledge that there is a winning strategy to the standard game.

However, if there is a change in size or connect values – the algorithm does it’s best to win. Given different starting moves it gives different outcomes, but given the same variables leads to the same results because each time the same heuristic values are applied.

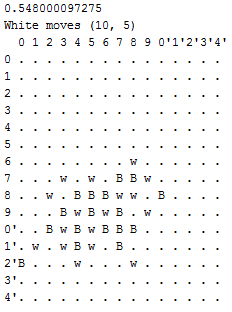
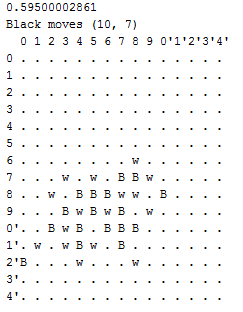
**\_\_init\_\_.py**

Here are some runs of the program on my machine:

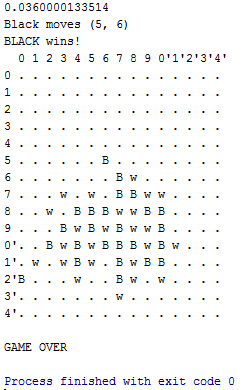
The opening state:

The game continues to build itself, printing each move and the time it took to compute, printing the board after it’s done, for example:



In the final stage, a winner is announced, and game over is declared:



**\_\_init\_\_.py**

In my research I found those sites most useful:

* [Wikipedia page](https://en.wikipedia.org/wiki/Gomoku) – to study about the game
* [Go-Moku and Threat-Space Search](http://vanilla47.com/PDFs/Gomoku%20Renju%20Pente/go-moku-and-threat.pdf) – research on the winning strategy for the game
* [quiescence search](https://en.wikipedia.org/wiki/Quiescence_search) - Wikipedia article on the search I chose
* [Gomoku by Oren Finard](https://github.com/Nero144/Gomoku) – I have studied his project and used some assets from his implementation.