UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

NCLab

Project Report

Agenti Intelligenti

Anno Accademico 2021/2022

Vincenzo Salvati – 0622701550

v.salvati10@studenti.unisa.it

Paolo Mansi – 0622701542

p.mansi5@studenti.unisa.it

Prof Marcelli Angelo

Prof Della Cioppa Antonio

Nomenclature

PEAS: Performance measures, Environment, Actuators and Sensors

AI: Artificial Intelligence

GUI: Graphical User Interface

List of Figures

List of Tables

Summary

1. Gomoku game

The aim of this project consists in developing an AI which is able to play to Gomoku.

Gomoku is an abstract strategy board game, also called Five in a Row, played on a Go Board with black and white stones. The game originated in Japan in the Heian period, where it is referred to as "Gomokunarabe", where "Go" means five, "Moku" is a counter word for pieces and "Narabe" means line-up.
This game has a state-space complexity of approximately 10^{105} and a game-tree complexity of approximately 10^{70}. In 1994, Allis proved mathematically that Gomoku is a solved game since it is based on precise mathematic calculations. Therefore, its rules are easier to implement using fundamental algorithms rather than neural networks (Liao, 2019).

1.1. Game rules

Gomoku is characterized by few simple rules:

- the match is contended between two players who alternates in placing black and white stones respectively
- each of the players can put one stone per turn
- the game starts with a black stone
- the victory is granted for the first player who places five stones in a row consecutively (either vertical, horizontal, or diagonal direction is valid), but six or more stones do not consist in a win.

It is important to notice that, if left unrestricted, Gomoku has a strong advantage for the first player. One of the common techniques used in professional tournaments to balance the game is using the "Swap 2" protocol:

- First player place two black stones and one white stone
- The second player has three options:
  - Play with black stones
  - Play with white stones
  - Place one black stone and one white stone so that the first player must choose his own colour.

1.2. Logical agent

As said before, this game involves two players alternating turns and it is zero-summed, since the opponents has the same conflicting objectives. The player's action consists simply in adding a stone on the board, which means that this game is deterministic, since the outcome is completely predictable.
Furthermore, it involves perfect information since the full vision of the board is available throughout the whole game, leading the players to always have the chance of making the best choice.

As far as the PEAS system is concerned, agents can be categorized in delivering the performance measure with respect to the environment, actuators, and sensors of the respective agent. So, the rational agent considers as many possibilities as possible and chooses to perform the highly efficient action.
Performance measure is the unit to define the success of the agent which depends on the precepts received from the environment.

Thus, the agent, which interacts with the Gomoku environment, has the following properties:

- **Fully observable**: the agent can determine the complete state of the environment at all points of time
- **Multi-Agent**: the player pursues his victory by countering the opponent's utility function. Hence, each player has a competitive behaviour against the opponent
- **Deterministic**: in Gomoku the outcome can be determined based on a specific state, because the environment is fully observable
- **Sequential**: the environment is sequential since each action affects the sequent. Indeed, each time one player set a stone in a specific place, none of the next moves can set the stone in the same place
- **Static**: one state of the environment does not change over time
- **Discrete**: there are a finite number of states and actions.

The best suited logical agent in such environment is a **goal-based agent** who predicts the state of the world based on the actions carried out. In that way, it tries to get as close as possible to the prefixed goal, i.e., the victory.

2. AI

This chapter aims to explain the algorithm and the strategies used to decide the agent's next moves. It describes how each state is evaluated and how the most suitable state is achieved.

2.1. Algorithm explanation

As game theory suggests, it is good to have an approach that evaluates strategies rather than carrying out simple searches within a states' space. However, the opponent is unpredictable, and, at the same time, every move must take place within time limits.

Therefore, the analysis of individual behaviours must be done considering that the actions have an effect that depends on the opponent's choices.

Since the goals between the players are conflicting, the logical agent must maximize its own utility, leading to victory, while minimizing the opponent's utility.

Thus, we could observe a sentient choice in which an x action is preferred over an y action if and only if x allows to obtain a higher score despite any other choice of the opposing player.

The best suited algorithm for achieving such goal is the Minimax, a backtracking algorithm used in decision making to find the optimal move assuming that the opponent also plays optimally. In Minimax, starting from the current state of the game, the tree of each possible outcome of the game is constructed. The leaves of this tree are evaluated with a certain utility function and then this value is back propagated in such a way that, for each level, if it is the player's turn (called maximiser) the highest of the possible values is chosen, otherwise the opponent (called minimiser) chooses the lowest possible value.

However, Gomoku has a huge state space, whose states cannot be enumerated totally and, therefore, we made use of the ***α-β pruning*** algorithm, which differs from the Minimax algorithm because tries to decrease the number of nodes that are evaluated. In particular, it stops evaluating a move when at least one possibility is proved to be worse than a previously examined move. This algorithm returns the same move as the Minimax but prunes away branches that will not influence the final decision, leading to a lower computational effort (Russel & Norvig, 2010).

2.2. Design choices

As for the design, taking into account both the experience acquired in this game and considerations on the computational complexity required, it was decided to act upon the utility function, the set of possible moves and the terminal test of the searching tree, to achieve valuable results within reasonable complexities. Such choices will be explained in this chapter.

2.2.1. Utility function

Analysing previous games, it was noticed that there is a series of significant pattern that may lead the player to victory. The considered pattern, involving five places each, consist in:

- *five in row*: five stones in a row, which means victory for the player (Figure 2-1)
- *four in row*: four stones in a row with one empty space on either side (**Errore. L'origine riferimento non è stata trovata.**)
- *broken four*: four stones interrupted by one empty space inside (Figure 2-3)
- *three in row*: three stones in a row with two empty spaces on either side (Figure 2-4)
- *broken three*: three stones interrupted by two empty spaces inside (Figure 2-5)
- *two in row*: two stones in a row with three empty spaces on either side (Figure 2-6)
- *broken two*: two stones interrupted by three empty spaces inside (Figure 2-7)

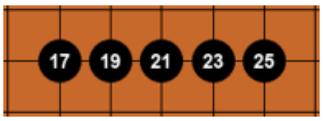

Figure 2-1 - Five in Row pattern

Figure 2-2 - Four In Row Pattern

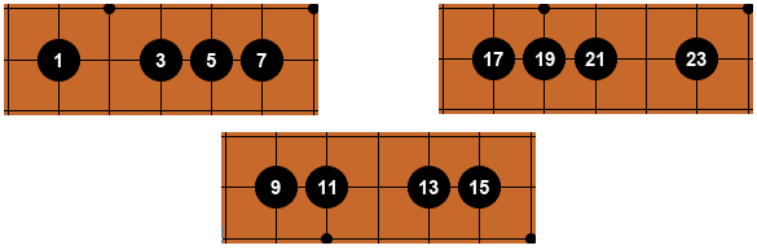

Figure 2-3 - Broken Four patterns

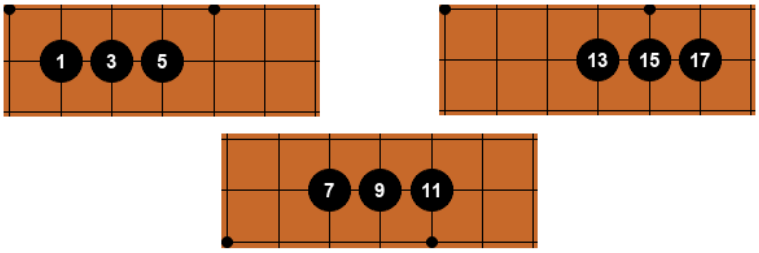

Figure 2-4 - Three in Row patterns

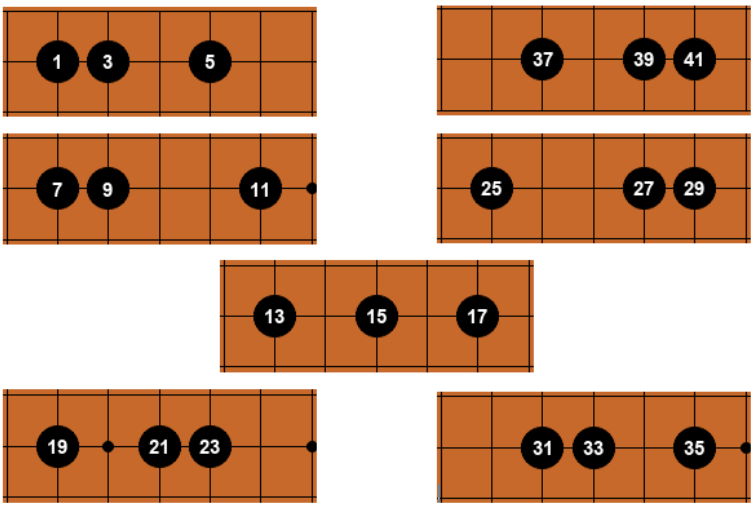

Figure 2-5 - Broken Three patterns

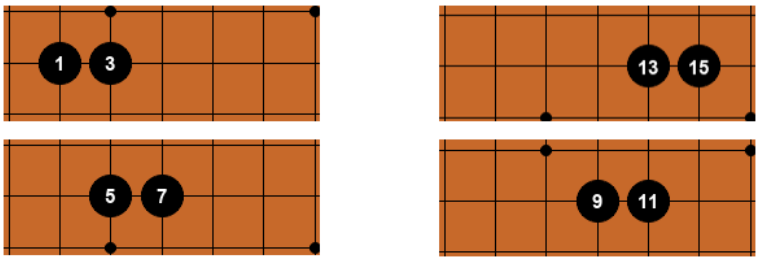

Figure 2-6 - Two in row patterns

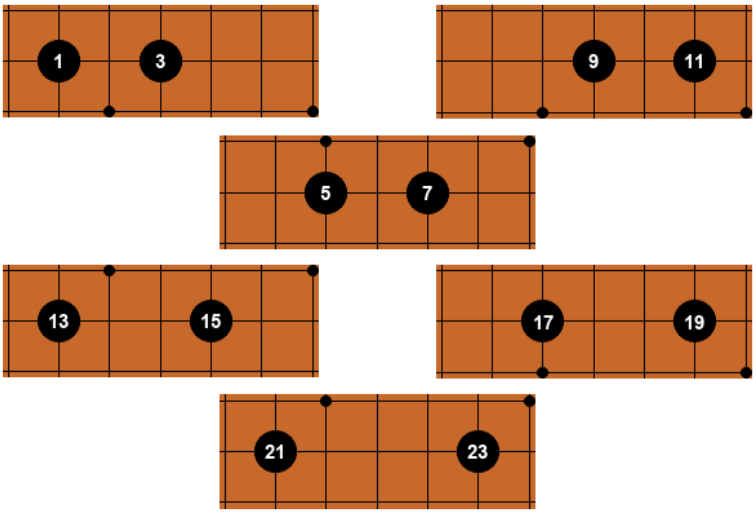

Figure 2-7 - Broken two patterns

The implementation of the utility function carried out consists in, firstly, extracting all the lines of the board; secondly, from these ones, considering all the possible 5 places sub-sequences and then, counting the number of occurrences of each pattern, both for the player and for the opponent. Lastly, a weighted sum is used to define the advantage or disadvantage of the player for such a state of the game. The opponent's utility is summed with the negative sign, so that, the lower the utility, more chances the opponent has to win. The weights are associated in order to discriminate the pattern according to an appropriate importance and the values used are shown in Table 2-1.

Pattern	Player's weight	Opponent's weight
Five In Row	27000	26999
Four In Row	905	900
Broken Four	905	900
Three in Row	25	30
Broken Three	25	30
Two in Row	1	0.5
Broken Two	1	0.5

Table 2-1 - Heuristic's weights

It is important to notice that the combinations of six stones or more do not consist in a win, so it was decided to mask such situations. In particular, during the evaluation of the line, whenever an overshoot pattern is detected, all the values of the stones, plus the eventual empty ends, are masked with a neutral value, so as not to match any pattern and, thus, avoid giving contributions to the weighted sum. In addition to that, also the potential overshoot patterns, which means lines of six or more stones of the same colour interrupted by a single blank space, are masked by replacing the blank space with a neutral value. This masking does not modify the actual state of the game, but only how the state is evaluated.

From the Table 2-1 it can be inferred, firstly, that the Five in row pattern has the highest value since it consists in a victory. If the player has the possibility to win, it does not have to defend anymore and, therefore, its weight is higher than the opponent's.

Secondly, the four stones combinations (Four in Row and Broken Four), the three stones combinations (Three in Row and Broken Three) and the Two stones combinations (Two in Row and Broken Two) have respectively the same weights, since they involve the same number of stones. In particular, the higher is the number of stones, the higher is the associated value, considering that they differ by about a multiple of 30, so that about 30 occurrences of a lower combination can be equal to 1 occurrence of the higher adjacent combination (ex. the opponent needs 30 occurrences of the Three in Row to reach the weight of the 1 Four in Row)

In addition, the weights associated to the player's combinations have a slightly higher value compared to the opponent's ones, leading to attack in case of balance of the game. An exception to this behaviour is constituted by the three patterns, because they can involve a higher risk for the player and, so, the defence against these combinations is suggested.

Finally, some configurations of the board involve matching the same patterns multiple times. For example, three stones in a row, with both ends free will match three times the pattern Three in Row. This makes this configuration stronger and so more likely to be reached.

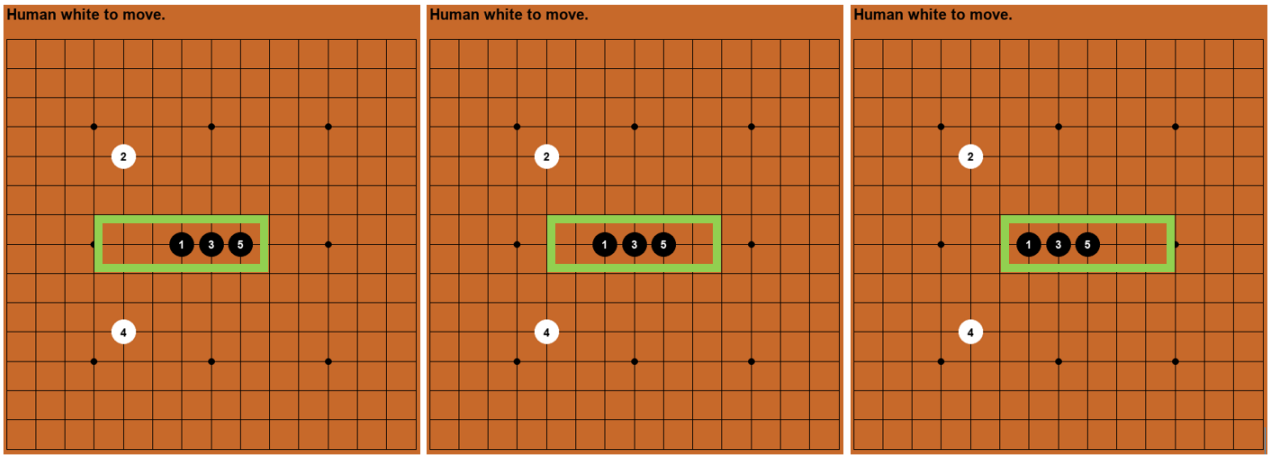

Figure 2-8 - Fetching sequences on board

Next, an example of state utility computation is carried on. The board considered is shown in Figure 2-9 and in **Errore. L'origine riferimento non è stata trovata.** and Table 2-2, the AI's utility (White) and the opponent's utility (Black) respectively are shown.

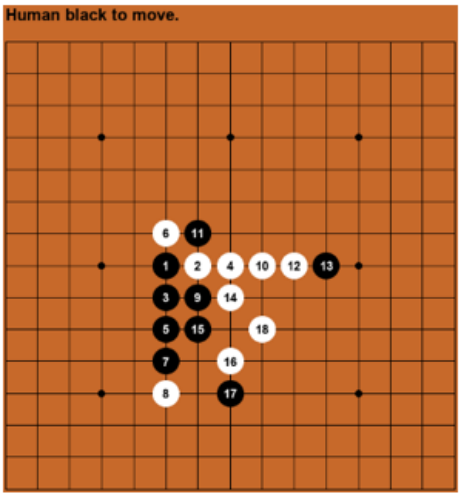

Figure 2-9 - Example match

Pattern	AI's Weights	Occurrences	Value
Five in Row	27000	0	0
Four in Row	905	2	1810
Broken Four	905	0	0
Three in Row	25	2	50
Broken Three	25	1	25
Two in Row	1	7	7
Broken Two	1	3	3

Table 2-2 - AI (White) Utility calculation for the state of the game in Figure 2-9

Pattern	Opponent's weight	Occurrences	Value
Five in Row	26999	0	0
Four in Row	900	0	0
Broken Four	900	0	0
Three in Row	30	0	0
Broken Three	30	0	0
Two in Row	0.5	9	4.5
Broken Two	0.5	3	1.5

Table 2-3 - Opponent (Black) Utility calculation for the state of the game in Figure 2-9

The final Utility for this state of the game is:

$$(1810 + 50 + 25 + 7 + 3) - (4.5 + 1.5) = 1889$$

Which means that the current Black player is in an utterly bad position, and it will probably lose the game, vice versa the White player is on the verge of winning.

2.2.2. Possible moves

As far as the possible moves are concerned, it was noticed that placing the stones in an isolated part, which means a place afar from the zone with the higher density of stones, resulted in a bad move since it does not add any advantage to the player and does not block any opponent's moves. Thus, such moves will never be chosen among all the possibilities, and, for this reason, it was decided to restrict the choice of the set of evaluated moves only to the surroundings of the already placed stones.

This is carried out by considering a 3x3 neighbourhood centred in the position to evaluate and adding it to the set of possible moves only if at least one of the neighbours is occupied.

This filtering of low interest moves decreases considerably the computational effort since many undesired moves are not evaluated although, of course, the number of available moves grows as the number of stones set on the board gets higher.

The Figure 2-10 shows as green points the possible moves considered at the current state of the game.

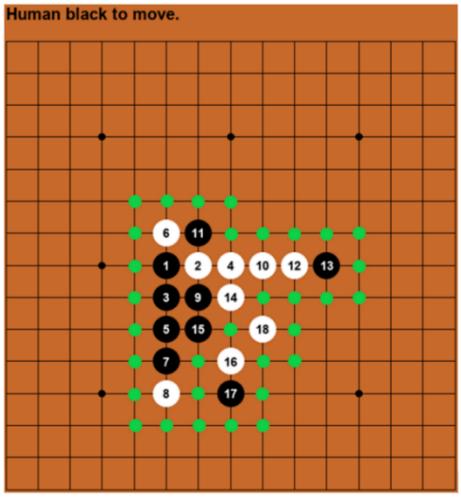

Figure 2-10 - Possible moves of the state of the game represented in Figure 2-9

2.2.3. Terminal test

Taking into consideration the computational effort needed to explore the whole searching space, it was decided to block the branching of the searching tree to the depth of 2. In this way, for each move of the player, also the sequent countermove of the opponent is considered, leading to choosing configurations that the opponent is not capable of blocking and, thus, winning the game.

However, an exception to this behaviour is due to the certainty of victory, which consists in a Five in a Row pattern found. In this case, the AI can avoid searching in the whole searching tree but returns the moves that consists in the victory. So, it was decided to block the search in the tree if the utility of the state being considered is greater or equal to the $\frac{4}{5}$ of the weight associated to the Five in a Row Pattern. The threshold's choice is dictated by two reasons:

- Avoid that the player's configurations which do not involve a Five in a Row pattern could overcome the threshold
- Avoid that the opponent's configurations decrease the total utility of the state such that its evaluated score is placed below the threshold

3. Implementation

In this chapter, an implementation of the proposed solution is presented, discussing its organisation and modalities of execution.

All the code can be found at the following GitHub repository:

https://github.com/VincenzoSalvati/projectAI.git

3.1. Code organisation

The developed code is divided in the following folders/classes:

- The "*bot*" folder contains all the classes related to the AI as well as the heuristic and α-β pruning search. In particular, it contains:
  - *alpha_beta_pruning.py* implementing α-β pruning algorithm
  - *BotGomoku.py* implementing the AI
  - *constants_ai.py* that contains some constants used for the AI
  - *patterns.py* that contains all patterns considered by the AI
- The "*graphic*" folder contains all the classes related to the board of the game as well as the update of the GUI. In particular, it contains:
  - *BoardGomoku.py* implementing the board of the game
  - *ButtonHome.py* displaying and show animations of home page's buttons
  - *constants_graphics.py* that contains some constants used for the board internal structure
- The "*data*" folder contains wav file to reproduce audio effect
- The "*utility*" folder contains all the classes related to the AI and the board. In particular, it contains:
  - *Chronometer.py* implementing the chronometer in order to take in consideration both the elapsed time for each AI move and the match's total elapsed time
  - *utils.py* that contains csv function in order to make the matches' log, which will be saved in the "*log*" folder
- The class *main.py* which allows to start the match initializing both the home and its modalities.

3.2. How to run

1. Run the *main.py*
2. From the Main page (Figure 3-1), use the buttons to choose the modality

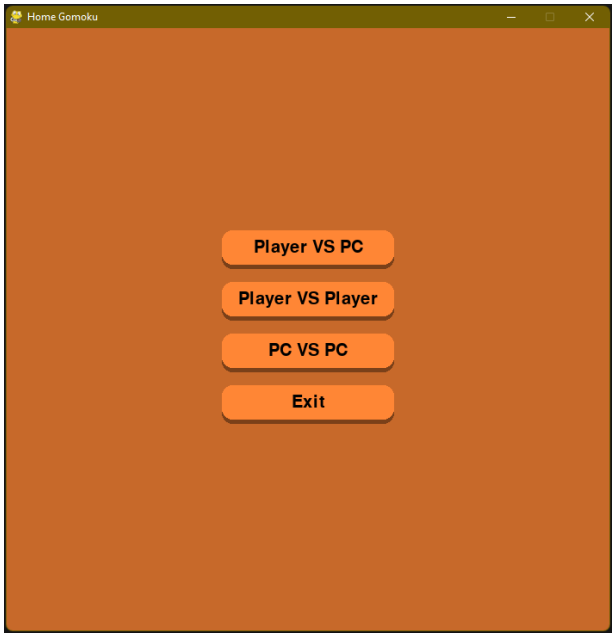

Figure 3-1 – home page - how to run

3.2.1. Modalities

- Player vs PC

Initially the system asks if the player desires to be the first one to place the stones (Figure 3-2):

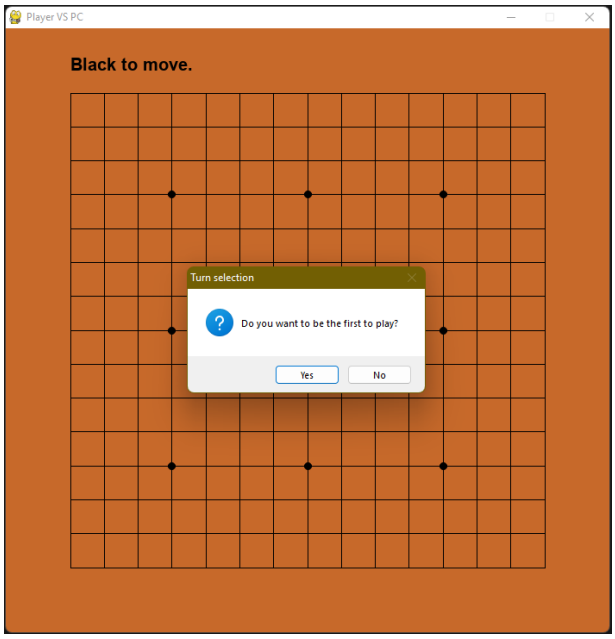

Figure 3-2 – Turn choice in Swap 2 - how to run

After pressing yes, the player is able to place: 1 black stone, 1 white stone and 1 black stones in this order. So, the bot, basing on his utility function, will be able to choose among (Figure 3-3):

- Playing with black stones
- Playing with withe stones
- Placing one black stone and one white stone so that the player has to choose its own colour.

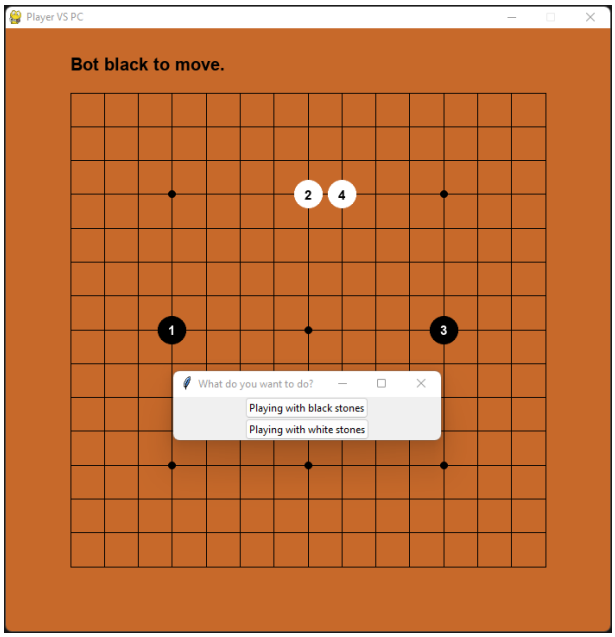

Figure 3-3 - Swap 2 colour choice (when the bot moves first) - how to run

After pressing no, the bot is able to place: one black stone, 1 white stone and 1 black stone in this order. Hence, the player can choose among the followings (Figure 3-4):

- Playing with black stones
- Playing with withe stones
- Placing one black stone and one white stone so that the bot has to choose its own colour.

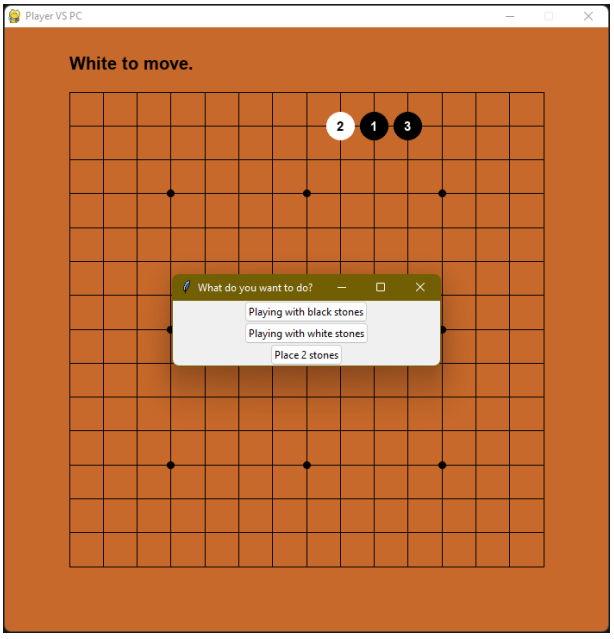

Figure 3-4 - Swap 2 (player moving first) - how to run

- Player vs Player

It is a free table which can be used by two players utilizing the same computer.

- PC vs PC

It has been used to compare different heuristics.

Its Swap 2 consist in:

1. Choosing randomly whose heuristics starts the match
2. Random first move for each player.

4. Conclusions

This chapter discusses the AI's performances and its strengths and weaknesses, analysing the results obtained by several matches against different type of adversaries.

4.1. Result analysis

In this section, some considerations over the performances reached is carried out. These evaluations take into account a high number of matches and the related results obtained, both in terms of winning ratio and regarding the time complexities.

4.1.1. Winning ratio

In the Figure 4-1, the ratio between the victories of the AI over all the matches performed against players is shown. It can be seen that the AI gets a high number of wins against the sample of players considered, which is composed of both low/middle experience human player and software programs expert in this kind of game. The results show that the AI does not achieve the best performance possible, and so it tends to lose against expert players, for the reasons explained in 4.2.

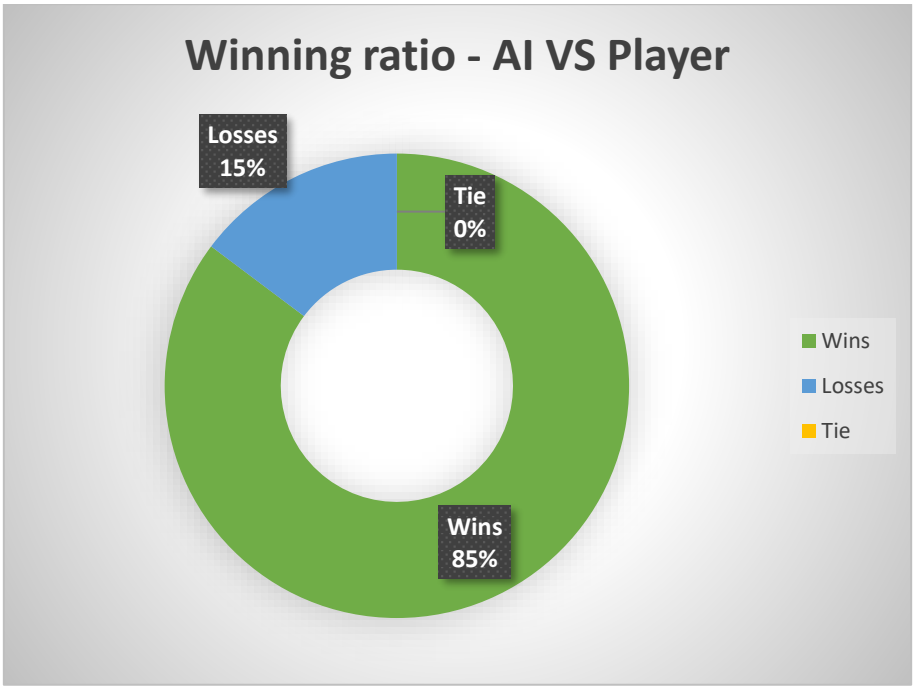

Figure 4-1 - Winning ratio of the AI against a player

It has been also carried out the comparison of wins against a secondary AI, which uses the same techniques for the evaluation of the game state but differs in the weights associated to each pattern. This secondary AI tends to be worse than the main one since the combination of lower importance moves acquire a bigger utility than most important ones, leading to wrong choices. The results of this second case are shown in Figure 4-2, from which it can be inferred that the proposed solution is able to discriminate the better situations and choosing accordingly, validating the choices made in the weights association.

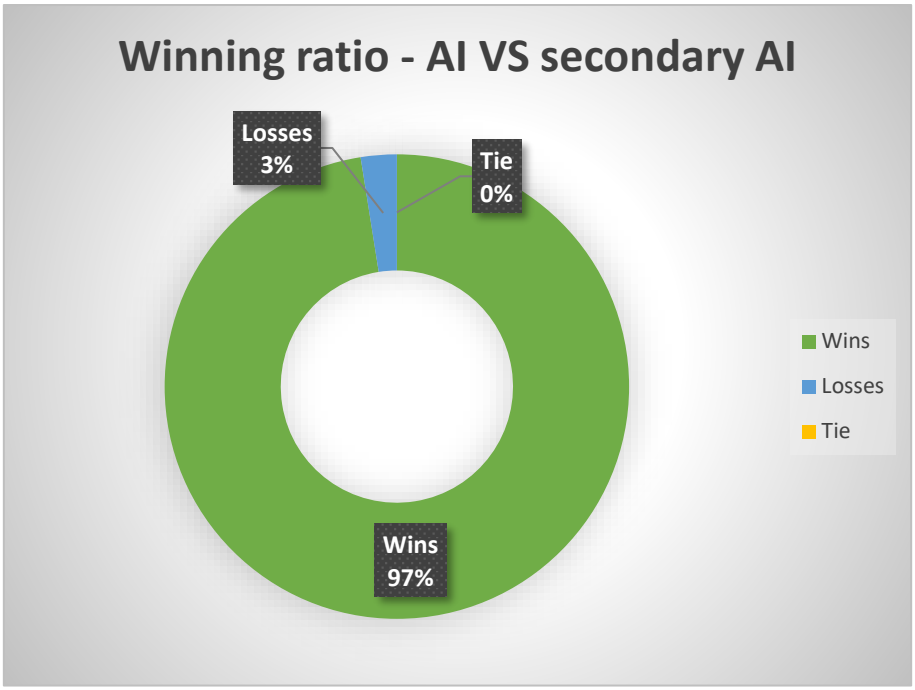

Figure 4-2 - Winning ratio of the AI against secondary AI, differing in weights association to the pattern

4.1.2. Time complexity

As far as the time complexity is concerned, the mean elapsed time of response of the AI was counted and shown in Figure 4-3. It can be inferred that the average time that the AI takes for making a move is highly influenced by the number of moves already performed. In particular, when the number of stones rises, the number of possible moves to be considered rises as well and, thus, a greater computational effort is needed to explore all the game search tree, even if it is cut off to a certain depth.

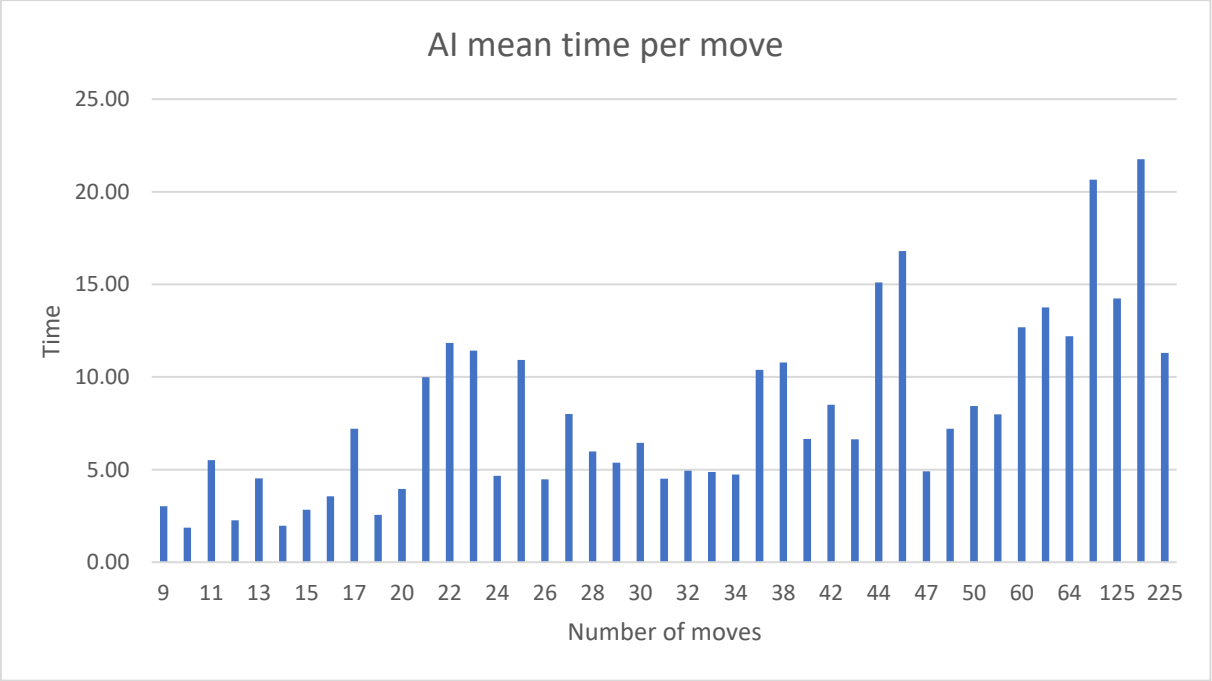

Figure 4-3 - Figure representing the average time the AI takes for making a move. The values are extracted by the log files, performing a mean between matches that involves the same number of moves

4.2. Strengths and weaknesses

The proposed AI it is not perfect under every point of view, but rather it presents a trade-off between quality of enacted responses and the computational complexity required for calculating such moves.

However, some considerations about the strengths and weaknesses of the solution can be carried out. As for the former, the ability of the AI to be alarmed by potential five stones combinations, ignoring potentially longer combinations, is a big strength since some combinations do not result in a victory. It can be also pointed out that the strategy enacted by the AI ignores single opposing stones located afar from the highly dense zone unless they became a danger for the victory.

As for the latter instead, the biggest limitation is that the AI does not takes into consideration complicated strategies because, to keep the computational effort as low as possible, it only considers empty positions adjacent to a stone and does only look ahead of two moves, as described in 2.2.

To overcome these limitations, it could be possible to look at a wider neighbourhood, for example a 5x5 instead of 3x3, and/or proceed in the search tree at a deeper depth to get the best move, but this involves an unavoidable worsening of the computational effort.

5. References

Liao, H. (2019). *New Heuristic Algorithm to improve the Minimax for Gomoku Arti.* Iowa State University, Ames, Iowa.

Russel, S., & Norvig, P. (2010). *Artificial Intelligence - A modern Approach.* Upper Saddle River, New Jersey: Pearson Education, Inc.