# Connect Four

Connect Four is a two-player connection game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. Connect Four is a solved game. The first player can always win by playing the right moves.

The project has been divided into two packages:

- **GameModels**: it contains the file used for model the problem;
- **Heuristics**: it contains the file used for the heuristics' implementation.

The following file is present inside the first package:

**__init__.py**: it contains the connect four game, where the various operations necessary for the correct execution of the game are carried out.

The following classes are implemented inside the file:

- Connect4Representation: it is used for having a "Snapshot" of the connect four game representation. Thus, it gives information about the board in all the different states;
- State: it is used for having a "Snapshot" of the connect four game state. The following method is implemented inside it:
    - isSolution: method that performs the check on the board after each move, returning 'true' if there is a solution and 'false' otherwise. It also carries out the control on the full table;
- Game: it contains the various operations necessary for the correct execution of the game. The following methods are implemented inside it:
    - neighbords: Method that at each turn returns all the boxes where it is possible to insert a checker before performing the heuristics. A fundamental check performed by this method is that of the possible winning moves (if there was at least one move that would lead to victory, would certainly be performed) and dangerous moves by the opponent (if there is a move that could lead to victory, the latter will certainly be blocked);
    - print_board.

The following file is present inside the second package:

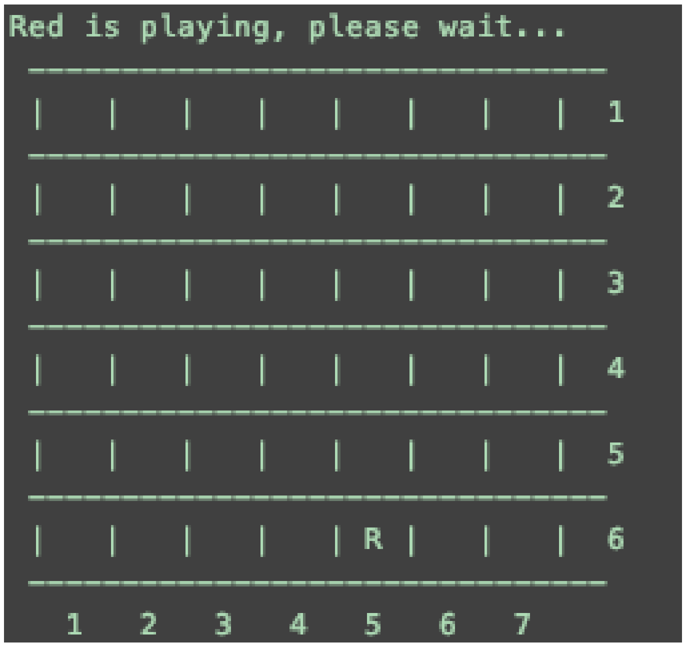
**__init__.py**: it represents the file where the heuristics are implemented. In both classes is implemented the MiniMax algorithm with AlphaBeta pruning that after a search Depth First returns the maximum or minimum value required by the current level.

The following classes are implemented inside the file:

- Heuristic1: sum up the pairs of red checkers aligned by inserting the next checker near them;
- Heuristic2: make the difference between the number of red checkers aligned and the number of yellow checkers.

**main.py** is the last file that represents the main class in which the moves made during the game are managed. Here is called the heuristics implemented as well.

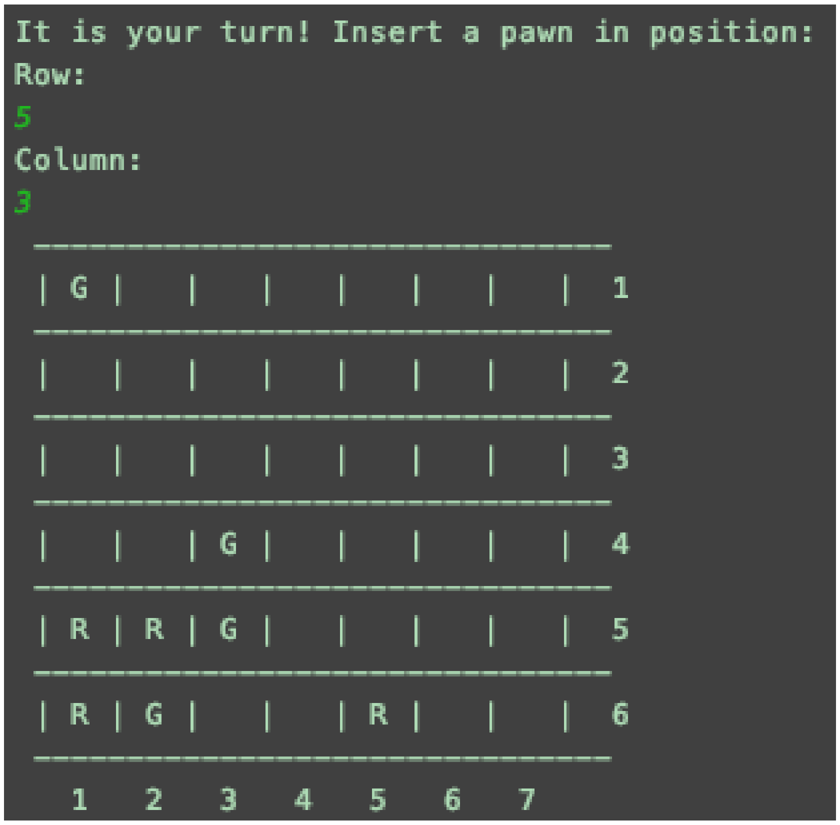
The picture below shows a possible initial board configuration after the first move performed by the computer.



```
Red is playing, please wait...
 -----------------------------
 |   |   |   |   |   |   |   |  1
 -----------------------------
 |   |   |   |   |   |   |   |  2
 -----------------------------
 |   |   |   |   |   |   |   |  3
 -----------------------------
 |   |   |   |   |   |   |   |  4
 -----------------------------
 |   |   |   |   |   |   |   |  5
 -----------------------------
 |   |   |   |   | R |   |   |  6
 -----------------------------
   1   2   3   4   5   6   7
```

Figure 1. Possible initial board configuration after one move

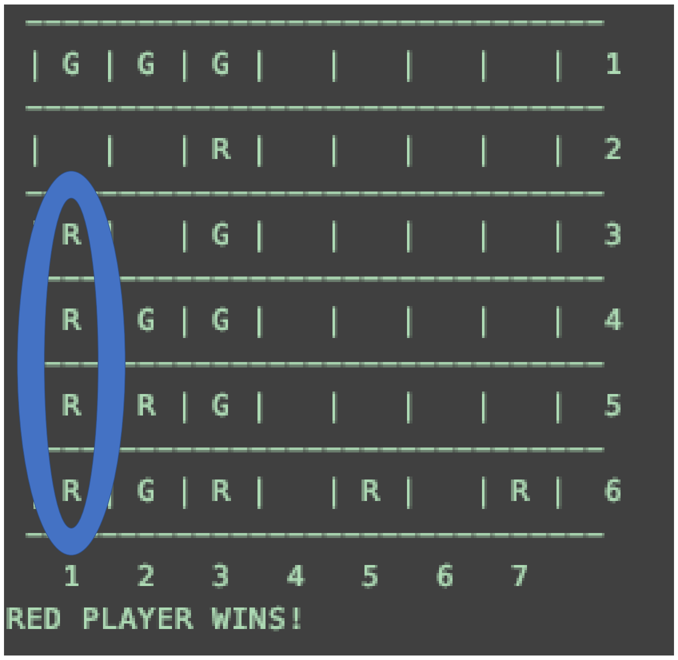
The picture below shows a possible board configuration after n number of moves performed by the computer and the player. In this case, it is the player’s turn. He can choose a cell indicating the number of the row and the number of the column.



```
It is your turn! Insert a pawn in position:
Row:
5
Column:
3
 -------------------------------
 | G |   |   |   |   |   |   |  1
 -------------------------------
 |   |   |   |   |   |   |   |  2
 -------------------------------
 |   |   |   |   |   |   |   |  3
 -------------------------------
 |   |   | G |   |   |   |   |  4
 -------------------------------
 | R | R | G |   |   |   |   |  5
 -------------------------------
 | R | G |   |   | R |   |   |  6
 -------------------------------
   1   2   3   4   5   6   7
```

Figure 2. Possible intermediary board configuration after n moves

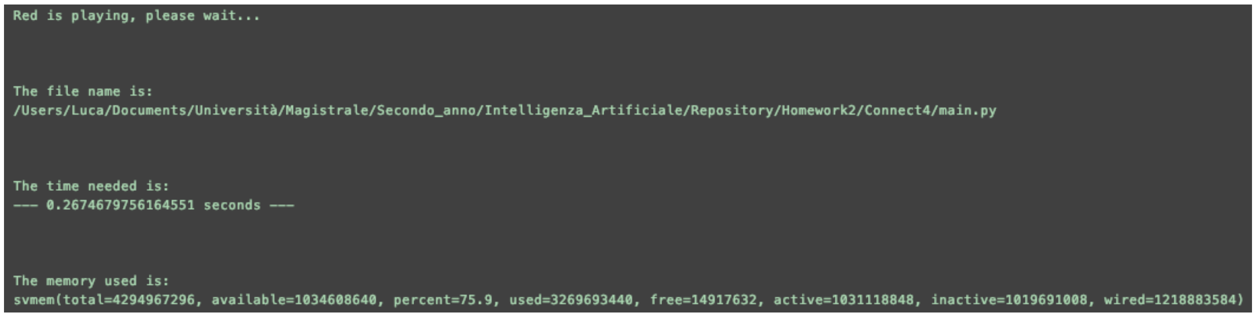
The picture below shows a possible winning board configuration.



```
 -------------------------------
 | G | G | G |   |   |   |   |  1
 -------------------------------
 |   |   | R |   |   |   |   |  2
 -------------------------------
 | R |   | G |   |   |   |   |  3
 -------------------------------
 | R | G | G |   |   |   |   |  4
 -------------------------------
 | R | R | G |   |   |   |   |  5
 -------------------------------
 | R | G | R |   | R |   | R |  6
 -------------------------------
   1   2   3   4   5   6   7
RED PLAYER WINS!
```

Figure 3. Possible winning board configuration

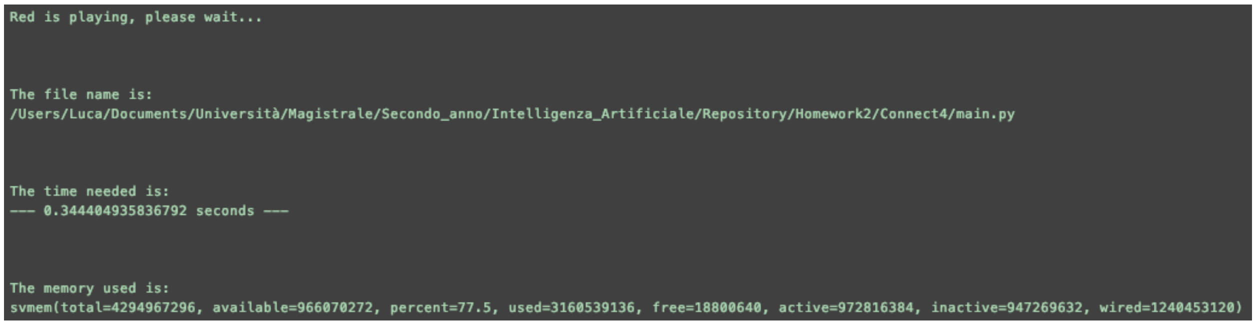
The following picture shows the indicative performance results of the execution of a single move by the first heuristic at the fourth level of research.



```
Red is playing, please wait...



The file name is:
/Users/Luca/Documents/Università/Magistrale/Secondo_anno/Intelligenza_Artificiale/Repository/Homework2/Connect4/main.py



The time needed is:
--- 0.2674679756164551 seconds ---



The memory used is:
svmem(total=4294967296, available=1034608640, percent=75.9, used=3269693440, free=14917632, active=1031118848, inactive=1019691008, wired=1218883584)
```

Figure 4. Performance results with Heuristic1

The following picture shows the indicative performance results of the execution of a single move by the second heuristic at the fourth level of research.



```
Red is playing, please wait...



The file name is:
/Users/Luca/Documents/Università/Magistrale/Secondo_anno/Intelligenza_Artificiale/Repository/Homework2/Connect4/main.py



The time needed is:
--- 0.344404935836792 seconds ---



The memory used is:
svmem(total=4294967296, available=966070272, percent=77.5, used=3160539136, free=18800640, active=972816384, inactive=947269632, wired=1240453120)
```

Figure 5. Performance results with Heuristic2

As you can see, the Heuristics2, in terms of timing, is slightly less efficient than the first one.