Assignment 2: Reversi

Artificial Intelligence Laboratory - Course 2021-2022

Publication date: 2022-03-02
Final tournament date: 2022-03-25
Moodle submission date: Friday 2022-03-25, 23:59 (for all groups)

General grading criteria:

- The implementation is correct.
- The code is well-structured. In particular, a proper functional decomposition has been made.
- Python programming structures and pythonic style are used (e.g. loops are avoided taking advantage of vectorized operations, or list comprehensions; suitable data structures are employed: tuples, lists, dictionaries, numpy arrays, iterables, etc.).
- The code follows PEP 8 specifications.
- The code is clear, readable, and well documented (e.g., functions are short and perform a single high level operation; the names of variables and functions are informative, there are no magic constants, etc.)

1. The game of Reversi

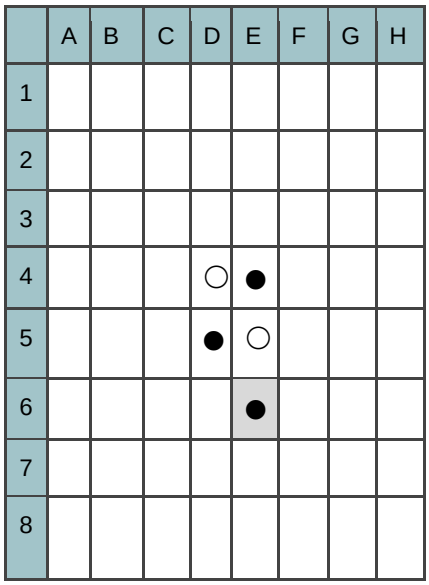
Reversi (also known as **Othello** or **Yang)** is a two player strategy game. It is played on an 8 × 8 board with the following initial configuration:

	A	B	C	D	E	F	G	H
1								
2								
3								
4				○	●			
5				●	○			
6								
7								
8								

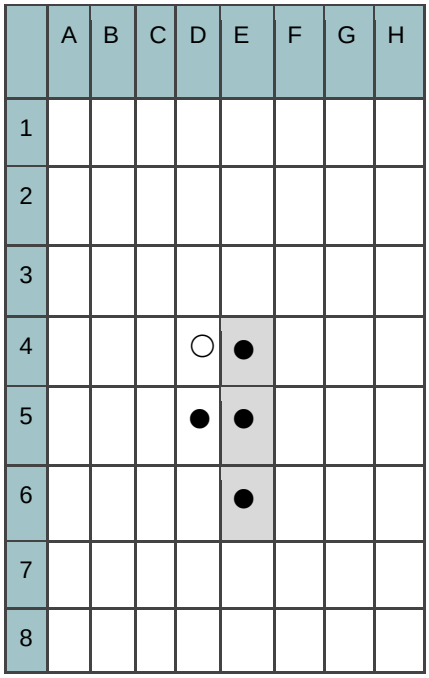
The two players, represented by the black and white pieces, alternate turns, starting with the black ones. On each turn, the player places a piece of her color on the board so that she captures one or more of the

opponent's pieces. A capture is made when the new piece, together with one or more pieces of the same color on the board, encloses a line of the opponent's pieces along a row, column, or diagonal of the board. The captured pieces are replaced by pieces of the color of the player who makes the capture. If a player cannot capture any of the opponent's pieces, the turn passes to the other player. The game ends when neither player can capture the opponent's pieces, or the board is completely occupied. The winner of the game is the player with the most pieces on the board.

In the following example, the player places the token ● in E6:



And captures the token ○ that is in E5, leaving the board as follows:



2. Implementation of Reversi and evaluation functions

The code provided for this assignment requires Python version 3.7 or higher.

The infrastructure for a game between two players is in the module `game.py`. The classes and functions specific to Reversi are implemented in the file `reversi.py`. **These files should not be modified**.

The strategies for the game (random player, manual player, minimax player) are implemented in **`strategy.py`**. **This is the file that must be modified to implement a player that performs a minimax search with alpha-beta pruning.**

The module **`heuristic.py`** contains some examples of game state evaluation functions, which are used in search strategies.

You can find examples of how to play in the file `demo_reversi.py`.

In the different examples that you can see in this file, you should observe the following:

- The game logic (in this case, `Reversi`) is included in the class `TwoPlayerGameState`, which stores the necessary information for two-player games (in addition to the logic of the game, the board, the starting player, if the game is over, the scores, etc.)
- The logic of a two-player game is handled through the class `TwoPlayerMatch`, which receives a state of type `TwoPlayerGameState` and is in charge of alternating turns, printing more or less information on the screen (according to the value of the variable `verbose`), checking that players do not take too long for a move (see variable `max_seconds_per_move`), as well as controlling whether the interface will be textual or graphic (variable `gui`).

To better understand the code, the most important classes and functionalities are explained below:

- The parameter `gui` of the class `TwoPlayerMatch` is used to determine if the game will be displayed as text in the terminal or if a graphical window will be created. In either case, you can use any player type (manual or automatic). If the player is manual, the behavior is different: if it has been chosen to play with the graphical interface (the value of the variable `gui` is `True`), then the player will have to click on the window; in text mode (`gui = False`) it will make its move through the terminal.
- The variable `verbose > 0` allows obtaining information about the minimax search. Specifically,
    - `verbose = 1 # The minimax value of the selected move is printed`
    - `verbose = 2 # The game runs step by step and the minimax values of all the states explored in the search are printed.`
    - `verbose = 3 # The game runs in a continuous way and the minimax values of all the states explored in the search are printed.`

  Additionally, in modes with `verbose > 0`, the possibility of storing the corresponding intermediate state is offered; that is, the information about the player who is going to make the move and the state of the board.

  In `demo_reversi.py` you can find different initialized players: manual, random, and some that use search strategies (applying the minimax algorithm for the search and a function for the evolution of game states). Players are implemented as game strategies (abstract class `Strategy` defined in module `strategy.py`).
- You mauy also explore other simpler games (they could be useful to debug the code): `demo_tictactoe.py` and `demo_simple_game_tree.py`.
- A `heuristic` function is included in the module heuristic.py (`simple_evaluation_function`) that can be passed to players of type minimax. In the next section, you will see that one of your goals is to design a heuristic function that is competitive.

- The strategy type `MinimaxStrategy`, in addition to the heuristic function, receives as an argument the maximum depth up to which the search will be made (`max_depth_minimax`).
- To allow for faster experimentation and testing of different scenarios, the delivered infrastructure allows you to play from an intermediate state. To do this, it is necessary to initialize the variables `initial_board` and `initial_player` with the corresponding values. To play from the standard configuration you can use `board=None` in the instantiation of an object of type `TwoPlayerGameState`.
- The functions `from_dictionary_to_array_board`, `from_array_to_dictionary_board` found in the file `reversi.py` allow you to transform the board from a dictionary form (the one used in the implementation of the game) to an array form (easier to visualize), and vice versa.
- The files `tictactoe.py` and `demo_tictactoe.py` allow you to play tic-tac-toe, a game simpler than Reversi and with a lower branching factor, which may be useful for your tests.
- For the implementation of the minimax algorithm with alpha-beta pruning, it may be useful to implement a simpler game. For example, one defined by a game tree. The implementation consists of
    - Creating a class derived from `TwoPlayerGame.`
      For example: `class MyGame(TwoPlayerGame):`
    - Implementing the functions:
        - `__init__`
        - `initialize_board`
        - `display`
        - `generate_successors`
        - `score`

In addition to trying to play the games that are configured in the module `demo_reversi.py`, you can also carry out a tournament with several strategies and evaluation functions by calling the function `run` of the module `tournament.py` indicating:

1. The strategies
2. The number of games with black pieces (the same number will be played with white pieces)
3. The names or aliases of the strategies.

A possible output of the code in `demo_tournament.py` is the following:

```
            total:     opt1_dummy opt2_random     opt3_heuristic
opt1_dummy      2:     ---        2               0
opt2_random     9:     8          ---             1
opt3_heuristic  19:    10         9               ---
```

In this configuration 30 games have been played in total (20 for each strategy): 30 = 5 × 2 × 3 = 5 repetitions (parameter `n_pairs`) × 2 players (each strategy plays as black and white) × number of strategies (3 in this case). You can see that the strategy `random` wins 9 of the 30 games, 8 against `dummy` and 1 against `heuristic`.

3. Implementation of evaluation functions

To implement your heuristic evaluation functions, you must create a python file with:

1. As many classes that inherit from `StudentHeuristic` as evaluation functions you want to test.
2. Within each class, you have to implement two functions: one named `evaluation_function` (which receives a potential move in the game, encapsulated as a state of type `TwoPlayerGameState`, and it must return an estimate of its value) and another named `get_name` where you will indicate the name by which this heuristic will appear in the tournament.

Here is an example with a trivial heuristic (it does not consider the state), where you can see how auxiliary functions can be included:

```
from game import (
    TwoPlayerGameState,
)
from tournament import (
    StudentHeuristic,
)

class MySolution1 (StudentHeuristic):
  def get_name (self) -> str:
    return "mysolution1"
  def evaluation_function (self, state: TwoPlayerGameState) -> float:
    # let's use an auxiliary function
    aux = self.dummy (123)
    return aux

  def dummy (self, n: int) - > int:
    return n + 4

class MySolution2 (StudentHeuristic):
   ...
```

Important: **the number of heuristics per submission is limited to 3**, so even if you include more classes, they will not be taken into account in the tournament. Through a Web interface, you will be able to upload **the only python file** that is requested so that you can define up to a maximum of 3 strategies, using a unique name that identifies each lab team: ***[gggg] _p2_ [mm] _ [surname1] _ [surname2] .py*** **using the format specified in the last section of this document**.

Examples:

- 2311_p2_01_delval_sanchez.py (strategies of lab team 01 in group 2311).
- 2362_p2_18_bellogin_suarez.py (strategies of lab team 18 in group 2362).

Observations:

- Players who make calls to system functions, use game control functions or manipulate its data structures will not be allowed.
- It is important that the evaluation function is efficient, since the time taken by each player will be controlled, so that the game will be considered lost by a player who is too slow. This timeout threshold cannot be an absolute time (which depends, in addition to the machine, on various factors, including the OS used and the depth of the search). However, to give an indication that is reproducible on any machine, it should be taken into account that **those players that take more than 5 times longer than the trivial heuristic** (class `Heuristic1` in `demo_tournament.py`) on an 8x8 board will most likely *timeout*. It is not possible to ensure that they will always be eliminated, because the duration of an evaluation also depends on the number of possible movements, which introduces some variability.

4. Evaluation of the practical assignment

This practical assignment is made up of three parts.

Part 1 (3 points)

As part of the game code, the student is provided with an implementation of the minimax algorithm (see class `MinimaxStrategy` in the module `strategy.py`). This implementation is based on the pseudocode seen in class:

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

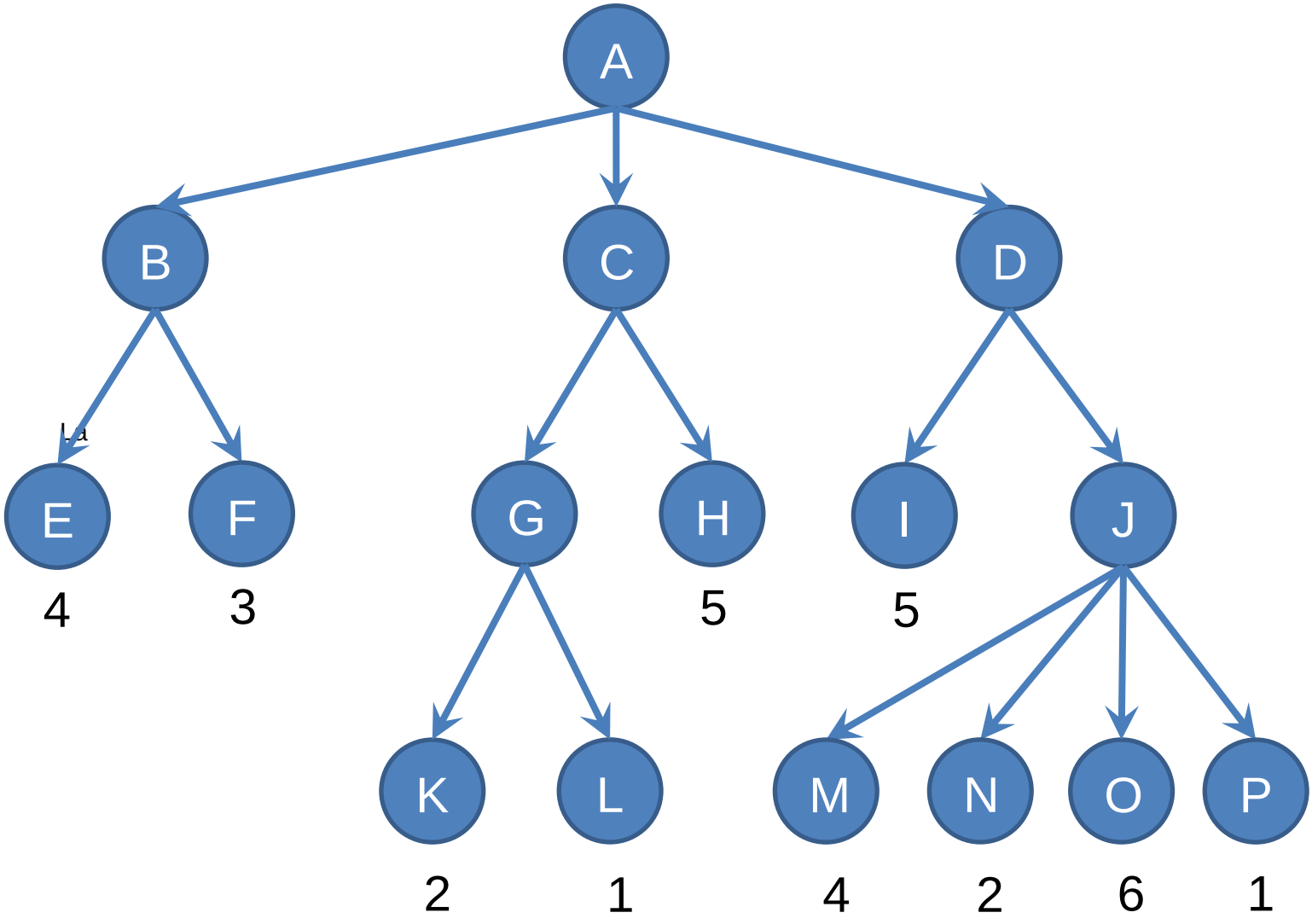
You have to code a function that implements the minimax algorithm with alpha-beta pruning; for this, define a class called `MinimaxAlphaBetaStrategy` that inherits from `Strategy` (you can use the class `MinimaxStrategy` as a starting point). The implementation must follow the pseudocode seen in class:

```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

Your delivered code will be valued for both functionality and style. It is important to clearly document in your report the algorithm used in the function performed, as well as its operation and its most relevant characteristics. To verify that the pruning is working, a time report must be submitted, for which the use of the `timeit`[1] library is recommended. To be more specific and avoid variations due to non-deterministic players, the analysis will be performed with a player using the deterministic strategy mentioned in the previous section, that is, Heuristic1 in demo_tournament.py at depths 3 and 4. Moreover, repeat the same analysis with your heuristic and compare the results, including a discussion in the report.

Keep in mind that the times may depend on the computer you are working on, so, in addition to the absolute times, you should provide some measure of improvement that is computer-independent.

To determine if the minimax implementation with and without pruning are correct, we propose to consider the game defined by the tree



The specification of such a game can be found in file `simple_game_tree.py`.

[1]Timeit documentation (last visited in February 2022): https://docs.python.org/3/library/timeit.html

For this game:

(i) Apply manually the minimax algorithm. On a copy of the tree, identify the iteration number, the node that is being visited, and the estimation of the minimax value for that node in each step of the algorithm.

(ii) Check that the manual execution agrees with the one performed by the minimax player in `demo_simple_game_tree.py` that starts the match. For this, include in the report the corresponding information to the first move of the game with `verbose=3`.

(iii) Apply manually the minimax algorithm with alpha-beta pruning. On a copy of the tree, identify the iteration number, the node that is being visited, and the estimation of the interval [α, β] for that node in each step of the algorithm. Finally, show the minimax value in the root node and the sequence of moves until the end of the game.

(iv) Check that the manual execution agrees with the one performed by the minimax player that uses alpha-beta pruning in `demo_simple_game_tree.py` when starting the match. To this end, **include in the report the sequence of values of the [α, β] interval associated to the nodes visited at each iteration of the algorithm**.

Part 2 (4 points)

This part consists in the implementation and submission of heuristic evaluation functions to a tournament between all lab teams (maximum of 2 students per team). The tournament will be conducted on **8x8 boards** starting with legal intermediate states, and **at Minimax depths no greater than 4**.

As a result of the games between evaluation functions, a classification will be published, accessible through a link that will be displayed in the tournament part of the course. On the dates indicated below, a tournament will be held among the strategies that are uploaded and the ranking will be updated on the Moodle page enabled for it. At a certain point, we will introduce evaluation functions anonymously to serve as indicators of the level of play.

The mark for this part will be calculated as follows (as you can see, it is possible to obtain *more than the 4 points* assigned to this section):

- Each tournament will have an increasing weight: 10, 15, 25, and 50%.
- Participation in the tournament with at least one function is mandatory. Not presenting any function, having been disqualified for form or execution errors, or for not meeting the minimum quality requirements or for any other reason: 0 points.
- The points obtained in each tournament will be calculated according to the ranking position of the heuristic functions: 100% of the grade (1 point to be distributed in each tournament, before applying the weights mentioned before) if the best submitted heuristic is in the 50-90 percentile, 150% if it is above the 90 percentile, 75% of the grade of the corresponding percentile is in the range 10-50, and 50% otherwise.

There will be open tournaments on the following days, in which you will have to upload 3 functions (at least one should be different to the previous ones):

- March 8th (upload before 20:00)
- March 11th (upload before 20:00)
- March 18th (upload before 20:00)

These three submissions are mandatory and **non-submission will be penalized** (1 penalty point for each missing submission). The final submission of evaluation functions will be on:

- March 25th, 2022 (upload before 20:00).

This final submission is separate from the complete submission of the P2, which will take place on the dates indicated at the beginning of this document. For that, you will use the Web interface available at http://150.244.56.40:5000/
To enter in this system, you will use your e-mail address and a password you will find in the Moodle task named “Access to use submission system in P2”.
Important: the submissions are controlled for lab teams, so it is critical the system has this information updated. If the system shows both members of the same team different information about their lab groups or lab numbers, contact the lab coordinator.

However, both in the code and in the report provided (see part 3), the selected strategies will be included again and the different solutions presented will be analyzed in detail, including references to articles (following the APA format[2]) that explain or justify the heuristics.

Part 3 (3 points)

It consists of a report with a description of the evaluation functions implemented as well as the documentation associated with the implementation of the alpha-beta algorithm, which will be uploaded to Moodle along with the code of the other parts.

What should be included in the report?

The report should be clear, complete, and concise. Please include only relevant information.

1. Documentation of minimax + alpha-beta pruning
    a. Implementation details
        i. Which tests have been designed and applied to determine whether the implementation is correct?
        ii. Design: Data structures selected, functional decomposition, etc.
        iii. Implementation.
        iv. Other relevant information.

    b. Efficiency of alpha-beta pruning.
        i. Complete description of the evaluation protocol.
        ii. Tables in which times with and without pruning are reported.
        iii. Computer independent measures of improvement.
        iv. Correct, clear, and complete analysis of the results.
        v. Other relevant information.

2. Documentation of the design of the heuristic.
    a. Review of previous work on Reversi strategies, including references in APA format.
    b. Description of the design process:
        i. How was the design process planned and realized?
        ii. Did you have a systematic procedure to evaluate the heuristics designed?
        iii. Did you take advantage of strategies developed by others? If these are publicly available, provide references in APA format; otherwise, include the name of the person

[2] APA Quick Citation Guide (last visited February 2022): https://guides.libraries.psu.edu/apaquickguide/intext.
Observation: references in APA format can be obtained very easily by searching for an article in Google Scholar, and then clicking on the "Cite" button. For example, when doing that with the following URL: https://scholar.google.com/scholar?q=the%20texbook, choosing the APA format returns:
Knuth, DE, & Bibby, D. (1984). *The texbook* (Vol. 15). Reading: Addison-Wesley.

who provided the information and give proper credit of the contribution as “private communication”.

c. Description of the final heuristic submitted.
d. Other relevant information.

For submission in electronic format, a ZIP file should be created containing all the submission material (complete code of heuristic functions and implementation of alpha-beta pruning + pdf report), whose name, all in lowercase and without accents or special characters, must have the following structure:

[gggg]_p2_[mm]_[surname1]_[surname2].zip

where

[gggg]: Group number: (2301, 2311, 2312, etc.)
[mm]: Order number of the lab team with two digits (01, 02, 03, etc.)
[surname1]: First surname of member 1 of the lab team
[surname2]: First surname of member 2 of the lab team

The members of the lab team must appear in alphabetical order.

Examples:

- 2311_p2_01_delval_sanchez.zip (practical assignment 2 of lab team 01 in group 2311)
- 2362_p2_18_bellogin_suarez.zip (practical assignment 2 of lab team 18 in group 2362)