University of Exeter

**COURSEWORK SPECIFICATION**
ECM1400 – Introduction to Python
Module Leader: Dr. Abhijit Chatterjee
Academic Year: 2025/26

# Title: **Continuous Assessment - Othello/Reversi**

## Submission deadline: **See ELE2 Page for deadline**

This assessment contributes **70%** of the total module mark and assesses the following **intended learning outcomes**:

- 1 design an algorithm, using sequence, iteration and selection
- 2 write, compile, test, and debug a computer program
- 4 systematically test your programs
- 5 document software to accepted standards
- 9 systematically break down a problem into its components
- 10 understand and choose appropriate programming techniques
- 11 analyse a problem and synthesise a solution
- 12 use technical manuals and books to interpret specifications and technical errors

This is an individual assessment and you are reminded of the University's regulations on collaboration and plagiarism. You must avoid plagiarism, collusion, and any academic misconduct behaviours. Further details about academic honesty and plagiarism can be found at https://ele.exeter.ac.uk/course/view.php?id=1957.

### Use of AI tools in AI-Minimal Assessments.

Assessment Title: Continuous Assessment - Othello/Reversi

Module Code and Name: ECM1400 – Introduction to Python.

The University of Exeter is committed to the ethical and responsible use of Generative AI (GenAI) tools in teaching and learning, in line with our academic integrity policies where the direct copying of AI-generated content is included under plagiarism, misrepresentation and contract cheating under definitions and offences in TQA Manual Chapter 12.3.

This assessment falls under the category of AI-Minimal in the University's Guidance on use of Gen AI in Assessment.

This means: You may use AI tools for checking spelling and grammar mistakes only, with no other impact on the structure or content of the assessment. This is because using GenAI tools outside of these uses prevents fair assessment of your ability to achieve module learning outcomes.

When writing your assessment, you must never use AI tools:

1. For uses other than checking your spelling and grammar.

2. To translate more than a word or short phrase into English.

3. To upload sensitive or identifying material to an AI tool

4. To present material that has been generated by AI as your own work or the work of someone else.

When submitting your assessment, you must:

1. Check the box during the submission process, that confirms you have adhered to the university’s academic conduct policy and the expectations on use of GenAI in your assessment brief.

# Instructions

Create a Python version of the popular board game Othello/Reversi.

Please note that this coursework has been classified as AI Minimal. This means you can only use AI tools for checking spelling and grammar in the DOCUMENTATION, and you must not use it in your code in any way – we are interested in YOUR coding ability and you must complete a declaration that you have not used AI for anything other than what is permitted in the AI Minimal classification. This means turning off all Copilot AI (for VS Code users) suggestions and similar tools within your IDE.

**Background:**

The game known as Othello or Reversi is a popular board game played on an 8x8 grid. 64 reversible counters with a light and dark side are divided between two players. An initial 4 pieces are placed in the four centre spaces of the board, in a pattern with two light side up diagonally opposite each other, and the other two dark side up.

**Rules:**

Dark always moves first.

The two players take turns to place a token with their colour up on the board in an empty square, such that it frames one or more of the opposition's pieces in between another of their tokens. These framed pieces are then flipped to the player's colour – this is known as being outflanked!.

If a player cannot go, they pass play to their opponent, but if they can make a legal move, they must.

The game ends when neither player can make a legal move.

The winner is the player who has the most counters of their colour on the board when the game ends.

For more information and a further breakdown of the rules please see:

English :: World Othello Federation

**Stage 1:**

**components.py**

Create a module called components.py that contains the following functions to setup the components of the game:

- Initialise board

Create a function called ***initialise_board*** with a single argument, *size*, that has a default value 8 and returns a list of lists that represents the board state. The initial board state should be empty so the list should be full of None values, except for the four initial counters in the four center board spaces (in the correct light/dark pattern). The initial data in the lists must contain one of the three states (either 'Light', 'Dark ' or 'None '). These should be padded with trailing spaces to 5 characters to make printing the board easier.

- Print board

Create a function called ***print_board*** that accepts a board as an argument and prints an ascii representation of the board state to the command line.

- Legal move

Create a function called ***legal_move*** that accepts a colour (string), coordinate (tuple) and a board object, and checks to see if the move is legal there for that colour player, returning true or false.

**game_engine.py**

Create a module called ***game_engine.py*** that will contain the main loop that runs as the players progress through the game.

Create a function called ***cli_coords_input*** that takes no arguments. In this function request the user to input coordinates for their move and process the input to return a tuple containing x and y coordinates, e.g. *(1, 4)*. Don't forget to use some error handling to make sure the coordinates are valid.

- Simple game loop

Create a function called ***simple_game_loop*** with no arguments. This is for intermediate manual testing through the command-line interface. Within the *simple_game_loop* you should implement the following steps.

1. Start the game with a welcome message.

2. Initialise the board.

3. Set a move counter to 60

3. Check there are legal moves available to the next player, if not switch to the other player.

4. Prompt the next player (starting with the dark colour) to input the coordinates of their next counter.

5. Check the move is legal (if not ask for a different coordinate) and change any of the opponent's counters that have been 'outflanked'. Reduce the move counter for that player and switch to the other player.

6. Repeat steps 3 - 5 until there are no more legal moves, or the move counter reaches 0.

7. Print game over message, with which player won and how many of each counter there were.

Call the *simple_game_loop* function from an ***if __name__ == __main__*** construct in the global namespace of the ***game_engine.py*** module so *simple_game_loop* can be manually tested by executing the module.

Well done, at this stage you have a simple two player game, and you can play against your friends! There is still room for improvement though!

Once finished save all the working files to a .zip archive of Stage 1.

**Stage 2:**

In stage 2 we need to make the game more visually appealing, intuitive and accessible. This involves creating a Graphical User Interface (GUI), and using .json files to be able to save and load the game state.

Use the provided html template to create a flask variation of the game, based on the working components.py file, but create a new flask_game_engine.py file to run a flask server to host the web page.

The basics of flask will be covered in the lectures and the workshops. You can alter the template to create a user interface for save and load buttons, and you should change the colours of the counters from the traditional black and white to a more contemporary dark/light colour combination of your choice. Remember to take into account the contrast of the colour scheme so it is easy to distinguish.

Once finished save all working files into a .zip archive of Stage 2

**Stage 3:**

In stage 3 you will create an AI opponent to try to beat. In this section you will construct a flow chart of the algorithm you create for your AI that must be presented in the documentation. This opponent should take the place of the second player – giving the human player a slight advantage.

Once finished save all working files into a .zip archive of Stage 3

**Stage 4:**

Clean Code and Documentation

Great! You have a working project. But it is useless unless it can be used and understood by others…

Use PyLint to check your code and make sure that it is correctly formatted! Try to get as high a Pylint score as you can for each of your .py files.

You must create comprehensive documentation for your project as two specific documents:–

1. A technical breakdown of how your code works in the ***readme.md*** file
    a. Flow diagrams of the algorithms used
    b. Explanations of how the code functions and what each module does
    c. The reasoning behind the choices made – i.e. why each module works the way it does,
2. A user manual ***manual.doc***.
    a. Detailed user instructions, from how to install (any dependencies required, specific running instructions), to how the api works so a different gui could be created by the user.

You must also have extensive comments and relevant docstrings within your code.

These documents should relate to the finished stage 3 project.

**Stage 5:**

Testing and Evaluation.

There should be a comprehensive test suite and its results documented as an appendix to the manual.doc submission. This can use pytest if preferred to manual testing, but should be fully explained. These should be performed on the finished product of stage 3, and added to the .zip archive.

**Submission:**

Code will be submitted as a zip file in ELE2. All source code required to run the project must be submitted and all documentation and remote repositories containing the code should be linked and clearly referenced.

Github Classroom will be available to all users but final submission via ELE as a .zip file is still required.

# Marking criteria

**Functionality [60% of the marks]**

Manual testing will be executed by following the instructions you define in the README to run your code and access the user interfaces, including the command-line interfaces and the Flask web interface. We will check for typical implementation as well as handling edge cases such as how the interface responds to invalid counter moves, and complex outflanking behaviour. The AI component will be tested for suitable move selection and algorithm logic.

**Code styling [20% of the marks]**

Style marks are assigned based on pylint evaluation and manually checking the source code against the criteria listed below.

- layout including consistent indentation, line length, etc.
- suitable identifier naming, appropriate variable scoping and appropriate control flow logic.
- type hinting and using default values where appropriate.
- doc strings and use of commenting.

**Project delivery [20% of the marks]**

Config files - Where config files are used, are they well formatted and clearly structured and contain all relevant information to customise the app for redeployment

Logging - Does the app maintain a log of events and/or errors, is the event log well-structured and used throughout, is the error log stratified into classifications

Testing - Does the app test functionality through unit testing beyond what was provided?

Deployment and documentation - Is there detailed README documentation, is there a license, author and metadata listed in the documentation, is the code hosted or published with a handle in the documentation, The completeness of the user manual.