# Assignment 1 (70% of the Module Marks): O'Neill Game

The objective of this assignment is for you to design and build an application using C# programming language, following the OOP paradigm. You will implement this as a Windows forms application. The application, named **O’Neillo Game**, is a game that allows two users to play the **Othello** board game on a computer.

If you are not familiar with the Othello board game, check the [rules](https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english) page on the World Othello Federation website. You will also find plenty of additional information about the game on the website. You can also have a go at [playing online](https://www.eothello.com/).

To complete this task, you will need to use most of the principles that will be covered in this module.

This assignment consists of three sections, each of which are briefly described here, and then presented in more detail in the remaining sections of this document.

The first section focuses on creating the main Graphical User Interface (GUI). The goal of the second section is to add gameplay functionality. The third and final section has you extending the features of the application working with persisted data.

Some classes would be provided which implement a simple example of separation of concerns – separating the presentation logic from the game logic. You should understand and consume these classes, making changes as needed in the game logic to complete your assignment.

You should submit your completed assignment as a single zip file via **Blackboard**, as stated in the [**Hand-in**](#_Hand-in) section of this document.

## Please Note.

This is an Individual assignment and **NOT** a group task. Should you require additional information, please follow this guide:

* You have an important but non-urgent question that everyone will benefit from having the answer.
  + Post your question in the appropriate discussion forum on Blackboard. You can post source code you’ve been given in the module, but NEVER POST source code you wrote in the discussion board.
* You have an urgent question, need to request support because you are stuck? Reach out using the details in the **Staff Details** section on Blackboard.
* Alternatively, ask during your timetabled sessions.

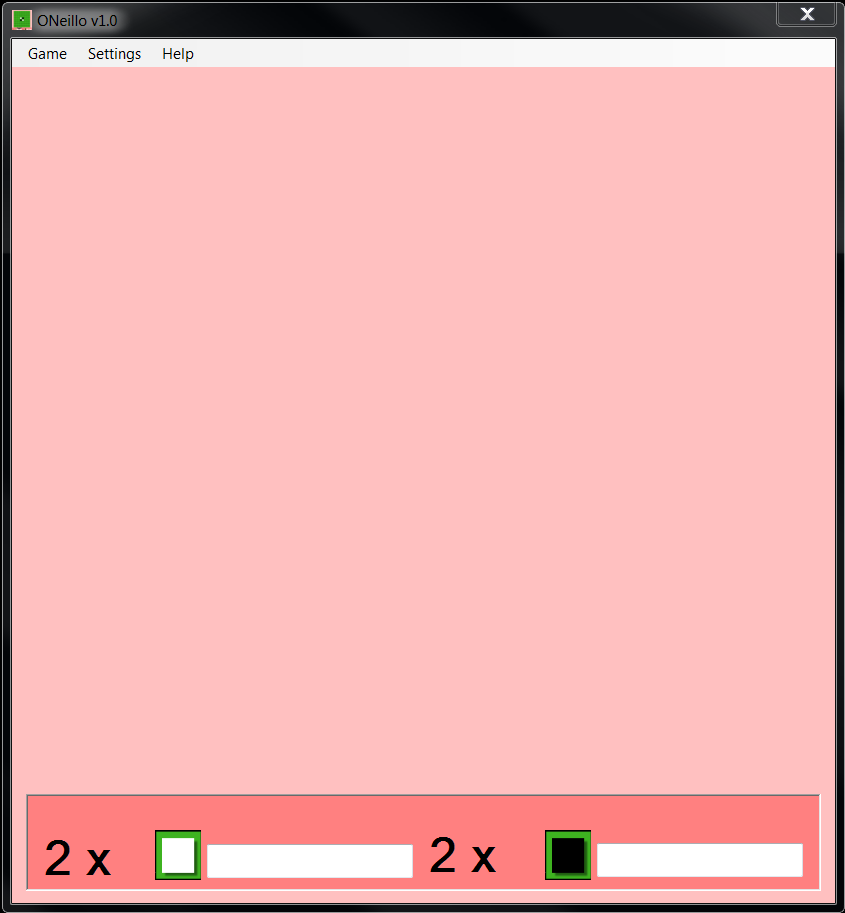
# Section 1. The Graphical User Interface – (20% marks of assignment 1 marks)

Write a program in C# to emulate the board game **Othello**. The application should be called the **O’Neillo Game**. A sample application, **O’Neillo v1.0**, is provided for you on Blackboard and is used in this assignment to describe the task.

**Note**: O’Neillo v1.0is provided for teaching purposes only – a visual aid to help describe the target and a teaching reference. Do not distribute the file.

|  |  |  |
| --- | --- | --- |
| **No.** | **Task** | **Available Marks** |
| 1. | Create a static Graphical User Interface (GUI) with the following features:   * A title bar showing the application name, **O’Neillo Game,** and the app icon. You are welcome to choose your own icon. * A menu bar with the menu items shown in Figure 1. * A gameplay area, which will contain the game board during gameplay. * A game information panel that shows:   + Each player’s token colour.   + The number of tokens each player has on the board.   + Player name – this should not be editable once a game starts.   + Next player to move indicator (see Figure 2).   You may choose to place your game information panel somewhere other than the bottom of the window, but your decision must enhance the game experience, not hinder it.   * Present the application in a fixed-sized window.   You may choose your own colours but must take into consideration the impact on accessibility. | 5 |

Next player to move indicator  
(Not visible here, see Figure 2)



Player names

Player Token

Exit Widget

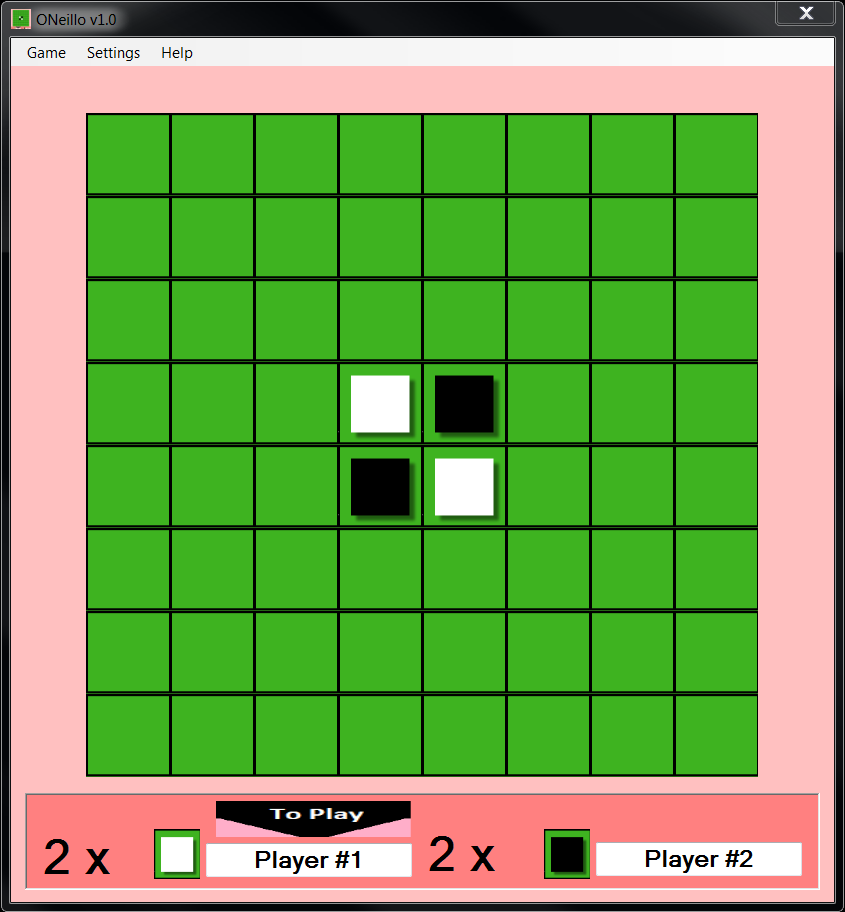
Number of player tokens on the board

Menu

Title and Icon

Figure 1: Static GUI

|  |  |  |
| --- | --- | --- |
| **No.** | **Task** | **Available Marks** |
| 2. | Create the dynamic elements of the Graphical User Interface (GUI). The dynamic elements should be similar to Figure 2:   * The 8 x 8 board for playing the game. * Next player to move indicator: there should be indicators that show the players which of them is playing next. Figure 2 uses an image, you are welcome to try something different. * The name of the players. This should default to **Player #1** and **Player #2** if theusers don’t provide any. | 15 |



Next player to move indicator

Figure 2: Dynamic Elements of the GUI

# Section 2. Game play requirements – (55% of assignment 1 marks)

All of the functionality outlined in **Section 2**, are based on the **O’Neillo v1.0** interface shown in Figure 1 & 2 above, unless stated otherwise.

**Please Note.** Many additional attributes and elements of the source code will be required to make these tasks work correctly. Therefore, you should use your own initiative by writing your own code to ensure your game is functional, free of error, and easy to use. Your implementation should be based on an OO design.

|  |  |  |
| --- | --- | --- |
| **No.** | **Task** | **Available Marks** |
| 3. | Create a 2-dimentional array to represent the **O’Neillo** **Game** information in memory. I.e., information about the board, each token and where they are placed on the board.  **Hint**: Part of the resources you are provided include images representing an empty square on the board, a square with a white token placed and an image with a black token placed. The images are respectively named **10.png**, **1.png** and **0.png**. You should consider using these names as values in your 2-dimensional array.   |  |  | | --- | --- | | **Value** | **Image** | | 10 = | P88C8T4#yIS1Empty square | | 1 = | A white square on a green background  Description automatically generatedWhite token on square | | 0 = | A black square in a green rectangle  Description automatically generatedBlack token on square | | 5 |
| 4. | Consume the 2-dimensional array in task 3 in the game logic and ensure your game data (represented by the 2-dimensional array) is correctly rendered on the board in task 2. | 10 |
| 5. | On selecting a cell from the GUI, have the application work out which tokens need turning to the current player's colour (if any).  These marks are broken down into the following tasks:   * Knowing that the move requested is an illegal move = 3 * Identifies when the current player has no valid move and notifies the player = 4 * Turning the tokens on the horizontal and vertical = 5. * Turning the tokens on the diagonals = 7.   Hint: Implement this as a separate class, which for example may be called OneilloLogic/GameEngine/GameLogic etc. | 20 |
| 6. | **Settings menu:** each setting works like a switch (reference the image below this task description).   * When the **Setting** > **Speak** menu is checked (turned on), the application will speak out loud each player’s move. **Speak** should be turned off by default. * When **Settings** > **Information Panel** is checked (turned on), the information panel described in Task 1 is displayed, it is hidden otherwise. The information panel should be turned on by default. | 3 |
| 7. | **Game** > **New Game** menuitem**:**   * Add the sub-menu **New Game**, to the **Game** menu**.** When clicked should cause the Dynamic GUI to show the start of a new game as shown in Figure 2. However, * If a game is in progress, then prompt the user to save the present game’s state, using a MessageBox, before starting a new game.  - Allow the user to be able to name the state, otherwise, save the game state using the current date and time as its name.  - Also allow the user to select from the list of saved game state to reuse the name, and overwrite the selected game state.  - When saving game state with an existing name, also warn the user they are overwriting a game state and offer an option to stop the action.  - Save the game state data in task 4. All data must be saved to a single Json file named **game\_data.json**.   + It must be possible to have multiple game states saved to this file – up to a maximum of five game states. When the maximum has been reached, the user must be able to specify which game state to overwrite.   + The names of the current players must be saved with the game state.   **Game** > **Exit** menu item:   * Add the sub-menu **Exit**, to the **Game** menu**.** When clicked should close the game. However:   + If a game is in progress, then prompt the user to save the present game’s state as described for task 6. | 15 |
| 8. | **Help** > **About** menu item:   * Add the sub-menu **About** to the **Help** menu**.** When clicked this should open an **About** window (Figure 3), which should be a modal window. Feel free to apply your creative freedom here, but the window should provide information about the application and its developer.  Clicking the **OK** button closes the modal window. | 2 |



Figure 3: About dialog

# Section 3. Working with persisted data – (25% of assignment 1 marks)

In task 2, you were asked to use application default player names or user provided player names. In task 7, you saved the game progress to game\_data.json. In this task, you will be doing a bit more with this Json data file.

|  |  |  |
| --- | --- | --- |
| **No.** | **Task** | **Available Marks** |
| 9. | **Game** > **Save** **Game** menu item:   * Add the sub-menu **Save Game**, to the **Game** menu**.** When clicked, save the current game state to the **game\_data.json** file. You should avoid duplicating code and re-use as much of the code you have in task 7 as possible. Like task 7, when requesting to save, a user should be able to provide a name, select existing names from a list and overwrite game state (after user is warned and confirms) if an existing game state name is used. The restriction of five game states still applies. Check the applicable details in task 7. | 5 |
| 10. | **Game** > **Restore Game** menu item:   * Add the sub-menu **Restore Game**, to the **Game** menu**.** Whenclicked:   + If there is only one game state saved, restore the game to this state.   + If more than one game state is saved, allow the user to select a state to restore. The player names should also be restored along with the state of the game.   When there are no game states saved, this menu item should be disabled. | 15 |
| 11. | * Save the setting changes in task 6 into the **game\_data.json** file. This is an application setting and should not be saved as part of game state. * Load and apply the last saved settings when the application re-opens. | 5 |

# Hand-in

**Hand-in items:** Hand-in your entire solution folder as a zipped file. Ensure the solution folder does not contain any artefact that does not contribute to the assignment. Run **Build** > **Clean Solution** onyour solution before creating a zipped file from your folder. Finally, before submission, ensure the following:

* The solution compile and runs.
* You have no pending git commits.
* Your solution includes a **Readme** file. Examples of information in your readme could be a brief description of your application and a link to a public GitHub repository for the project.

**Hand In Date: By the Date and Time stated on Blackboard.**

You will hand in the required files e.g., zipped file of your entire solution folder (as described above) on, or before the stated Date and Time on Blackboard. Alternatively, if you are still having difficulties in knowing what or how to submit, please contact [t.adeosun@shu.ac.uk.](mailto:t.adeosun@shu.ac.uk.)

To pass this module, you are required to score a minimum of 40% overall. Failing to accomplish this mark can result in you being required to, ‘Resit’ a single assignment, or undertake the complete year and in some cases, being asked to leave the University. (Please see University’s Procedures for more information.).

This assignment offers IMR (In-module retrieval). See the [Assignment brief](https://sheffieldhallam-my.sharepoint.com/:w:/g/personal/ta5403_hallam_shu_ac_uk/EXlIonNQkA9Bh72Eok0grz0BakAjhqzkNuTCWZvgZ_MAQw?e=FrsEJw) document for more about IMR.

# Hints and Tips

Ask if you get stuck - I won't tell you the answer, but I will guide you!

**Email :** [t.adeosun@shu.ac.uk](mailto:t.adeosun@shu.ac.uk)

**Telephone : 0114 225 2614**

## General tip

Do not try to solve the problem in one big chunk – remember the only way to “eat an elephant is a bite at time”. Solve each small part iteratively. Implement a class structure that represents the objects that you will need in your project, where a class is a suitable representation. The sections and sub-divisions of the tasks may be a good place to start decomposing the problem and designing a solution.

Try to solve the problem, thinking about a solution, before trying to express your solution in C#.

Break the application into smaller parts, the ‘Marking’ scheme above should give you some ideas.

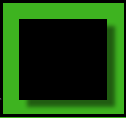
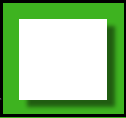
Develop a single part, but think how the part you are developing, will interact with other parts of your application.

Use “UML”, or some other design notation to work out the design of your application. If you are unable to complete all the tasks, your design captures part of your thought process and I can give you some credit for your understanding of the problem and producing a design.

## Code Quality Expectations:

Your application should not only work but follow good coding practices. Some have been discussed in the module, you can look up more on this. At least:

* Apply good code organization: each class created in its own file; reasonable class structure; decompose complex tasks into multiple methods that together complete the task – your decomposition should be logical, not haphazard. Organize related classes into the same namespace where this is useful. Check these resources for helpful tips for code organization: [organizing with functions](https://www.freecodecamp.org/news/organizing-code-with-functions/); and [organizing your code base](https://medium.com/@msandin/strategies-for-organizing-code-2c9d690b6f33).
* [Comment your code](https://aserg.codeberg.page/shu-dev-process/en/construction/comments/#overview), and make the comments useful.
* Choose and stick with a naming convention, and choose meaningful identifier names (variable names, method names, class names etc. are called identifiers). While the module does not specify a convention, you may consider using your employer’s convention or [C# conventions](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names).
* Use [exception handling](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/exception-handling).
* Test critical parts of your code.



10

1  
0

## Conceptual Hints

In memory 2-dimensional Array GUI (Dynamic)

The array above is mapped onto the control array representing the game board shown on the GUI.