Project Specification

Assessment 1: Preliminary design

- Weight: 30%
- Due: Week 3 (Sunday 13 August)

Assessment 2: Final design and implementation

- Weight: 70%
- Due: Week 6 (Sunday 3 September)

Overview

The goal of this project is to gain practical experience in object-oriented software development through object modelling with design diagrams and applying standard software design principles and patterns. You will demonstrate this knowledge by creating and documenting an object-oriented design and implementing it using C# for a real-world motivated problem.

To imitate real world software development practices, you can choose to complete this project either individually or in a team. While you can talk about the project with tutors and peers outside of your team, the submissions must be your own original work. Copying the work of others is not tolerated at QUT and submissions will be checked for code plagiarism against online repositories.

Every student must form and register a project team of **1-3 members** on the IFN563 Canvas (from the "People" section select the "Groups" tab) by **Week 2**. Only one team member needs to submit the assessments (i.e., not everyone in the team needs to submit). Each team will receive a single grade for this project.

Description

You are required to provide an extensible framework for many different two-player board games. To demonstrate that your framework can be easily adapted to different games, regardless of whether you work individually or in a team, your **design (including all design diagrams) must include all the following games** in the same software:

- SOS. Two players take turns to add either an `S` or an `O` (no requirement to use the same letter each turn) on a board with at least 3x3 squares in size. If a player makes the sequence `SOS` vertically, horizontally or diagonally they get a point and also take another turn. Once the grid has been filled up, the winner is the player who made the most SOSs.
- Connect Four aka Four in a Row. Two players take turns dropping pieces on a 7x6 board. The player forms an unbroken chain of four pieces horizontally, vertically, or diagonally, wins the game.

To demonstrate the feasibility and effectiveness of the design, you must implement the games that **correspond to your design using C# on .NET 6**.

- Students working individually (i.e., in a team of only one member) must implement the SOS game.
- Teams of two or three members must implement **both** the SOS game and the Connect Four game.

Requirements

Your design should extract as many commonalities as possible from the different games so that the framework is extensible and reusable for a wide variety of board games.

Your system should cater for different modes of play, including:

- Human vs Human
- Computer vs Human

With human players, the system must check the validity of moves as they are entered. With computer players, the system can randomly select a valid move.

Games can be played from start to finish, and should be able to be saved and restored from any state of play (i.e. a save file). A game should be replayable from any position after being loaded from a save file.

During a game, all moves made by both players should be undoable and redoable (i.e. the full history of moves are tracked). But the history of the moves does not have to be saved into a save file. That is, immediately after loading the saved state from a file, undo and redo operations are not available (until new moves have been made).

You should provide at least a primitive in-game help system to assist users with the available commands (also give some examples if they are not obvious to use).

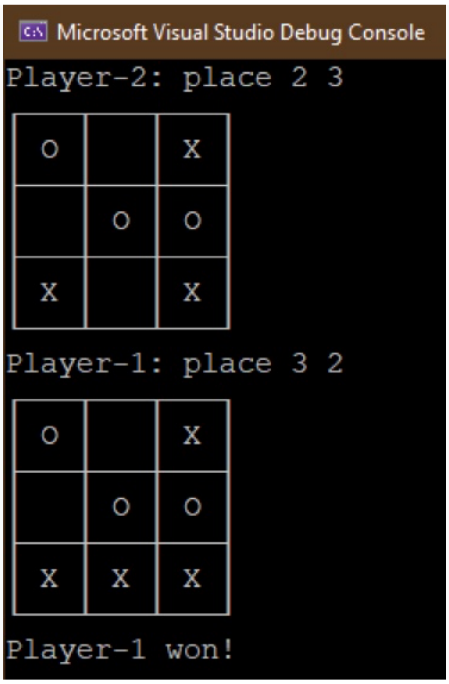
Comments on requirements

The requirements are what you should aim to achieve in your project. Even if you do not implement all features, you should at least cover all requirements in your design.

Some requirements may be vague and open to interpretation. As long as you clearly state any assumptions you make in your design report, we will accept reasonable interpretations.

The important aspect is that your design for the core structure and functionality of the two-player board game meets the requirements, in a clear and easily understandable way. You should steer your design towards a general framework and consider your implementation as a proof-of-concept of your design, rather than a fully-featured, highly intelligent game-playing program.

Your **implementation must be a C# console application on .NET 6** and provide **a text-based command-line interface** (e.g. using either ASCII or Unicode) of the game as it is played. Some marks have been allocated for usability, so be sure that whatever interface you use, the commands are simple and clear. The following screenshot shows a simple example of playing a game of the classic Tic-Tac-Toe by two human players, displayed in Unicode.



Submissions

Just like any design, OO design often is an iterative process and is learnt from experience. In this unit, students have the chance to gain first-hand experience of OO design by going through a supervised design process: draft a preliminary design, receive feedback on the preliminary design, improve the final design, implement and evaluate the final design.

There are two assessments from this project, one for the preliminary design (by the end of Week 3) and another for the final design report and implementation (by the end of Week 6). All submissions are to be made via the IFN563 Canvas website from the "Assignments" section.

Assessment 1: Preliminary design

The purpose of the preliminary design tasks is to provide a checkpoint on your progress early and allow you to receive feedback and improve the overall design of the project in the final design report.

These design diagrams should record your software design based on the project requirements. No implementation detail should be provided in the submission. You must make sure your design diagrams are clean and readable.

Design patterns are optional in the preliminary design. They will not be part of the marking criteria.

You should only submit **one PDF document up to five pages** in length to the Canvas, containing the following design diagrams:

- A high-level **CRC classes** design of objects in your program (one or two pages) from Week 2 Workshop;
- A single **class diagram** including all required games in the same software (one or two pages) from Week 2 Workshop;
- An **object diagram** displaying a snapshot of the program memory at a particular time during the program execution (one or half page) from Week 3 Workshop;
- A **sequence diagram** describing a significant scenario and exercise some important functionalities of the software (one or half page) from Week 3 Workshop.

Assessment 2: Final design and implementation

For this final assessment, you must submit **two separate files**: **a PDF file** of the design report and **a ZIP file** containing all your C# project files. The submission link will open in Week 6.

Final design report

You should only submit one PDF document with no more than 10 pages in length (A4 page size with 2cm in margins on all sides; 12-point Times New Roman or 11-point Arial font or something equivalent, and in single space).

Your report should include:

- a statement of completion (half page), clearly **declaring the requirements that have and haven't been implemented**. This declaration must be complete and accurate, in order to receive marks for the "Fulfilment of requirements" in the marking criteria.
- a list of all team members (full names, student numbers and emails) and a declaration of **contributions of each of your team members** (half page). Please note that any team member who didn't contribute substantially to the project will not receive any mark.
- an overview of your final design (one page). You should provide a short explanation of your **overall design and a brief summary of changes from the preliminary design** including both what and why changes were made.
- detailed final design documents (at most three pages), including **a class diagram, an object diagram and a sequence diagram**. You don't need to provide CRC cards because they are reflected in your final class diagram.

- identification and a brief justification of design principles and patterns that have been used (one or two pages). For each design pattern, you should **clearly indicate the participating classes and important operations** (in design diagrams) and justify their use in your design with a few sentences.
- a brief document on **how your program can be executed** (at most one page).
- a short summary of **classes/interfaces (one page) to be reused from existing libraries and frameworks**. For example, if you use the Collections library, just list the classes you will use, without any further explanation.

Note that you will be marked for the simplicity and elegance of design. It is in your best interest to make your design as neat, clear and understandable as possible. In general, your detailed design should speak for itself. Only include explanations of your design where your intentions may not be clear, but this should mostly be covered by the use and justification of design principles and patterns as requested above.

Implementation source code

You must submit a working implementation including full C# project source code for .NET 6. You should zip all your project files and upload this ZIP file onto the IFN563 Canvas website.

You do not need to provide a user guide or tutorial, but in your design report, you must clearly document how your program can be executed.

Your class implementations must be strictly faithful to the documented class designs in your final design report. That means the classes in the source code must correspond to the same classes defined in the class diagram, including their properties, operations and relations to other classes.

The submitted project files will be compiled and executed on QUT lab computers with .NET 6. You must make sure that your submitted code can be compiled and run properly from the lab computers with the correct .NET version.

Unfaithful class implementations will receive zero for implementation. Uncompilable or inexecutable source code cannot be marked and will receive zero for implementation. To confirm the version of .NET on the computer, simply open a terminal and run the following command:

```
dotnet --version
```

To check that your project code can be compiled and executed on .NET 6, open a terminal in the folder containing the project file (`.csproj`) and run the following commands:

```
dotnet build
dotnet run
```