|  |  |
| --- | --- |
| ****Subject Name and CRN:**** | INFT558 - Software Development 1  CRN - 72298 |
| **Assessment Type:** | Project |
| **Due Date:** | April 2nd, 11:00PM AEST |
| **Weight:** | 50% |
| **Marks:** | 100 |
| **Learning Outcomes:** | ALL |
| **Canvas:** | <https://aie.instructure.com/courses/42/modules/items/1663> |
| **Topic:** | C++ Projects |

# Project Brief

You are required to complete a series of small game projects that each demonstrate your understanding with various aspects of C++ programming.

You are also required to complete a debugging exercise to demonstrate your ability at problem solving and bug tracking.

# Project Features

You are to create 3 projects that implement the following games based off of their simple brief:

* **Tic Tac Toe Game**
  + This is a two player game on a 3x3 board. One player uses X tokens while the other uses O. The players take it in turns selecting one of the empty 3x3 squares to place their piece. The game continues until one player has 3 tokens in a row (vertical, horizontal and diagonal), in which case they are the winner, or there are no empty spaces left which results in a draw.
  + The Game must be implemented using 2-dimensional arrays, but you are otherwise free to implement it however you wish.
* **Battle Arena Game**
  + This is a game or simulation that consists of two teams of opposing forces. The forces must be implemented as Class objects stored within two separate arrays. The objects must contain health values. Each turn consists of the forces fighting each other, inflicting damage to one another in some way. Once the turn has finished the forces are to be sorted using a sorting algorithm based off of remaining health. The game or simulation continues its turns until all objects within a force are dead which results in a win for the force still living.
  + You may implement this game however you wish, either as two player, one player, or automated simulation, but it must make use of two arrays of custom Class objects that are sorted each turn.
* **Number Guessing Game**
  + This game consists of the computer trying to guess a number that you have selected from a given range.
  + You may implement this game however you wish, however the computer must make use of a search algorithm to find the number you have chosen within a range you have selected. Potential algorithms include Binary Search.

You must also complete the **Debugging** exercise that your teach will show you. This exercise consists of finding and removing errors from within a badly written project, but also requires you to improve it.

For all projects you will be assessed based on the successful implementation of the project, in addition to how technical your projects are and what systems and features they make use of. You are also to be graded on your coding standards and readability of your code.

# Submission Process

All submissions must be made by the due date and time listed.

Students submitting an assessment after the due date and **without** an approved extension of time will lose 10% of the maximum available mark on the assessment per day it is late. In addition, assessments submitted late will only be awarded a maximum of a **pass** grade. This means that an assessment submitted 2 days late can achieve a mark of 80%, which contributes to the subject’s overall mark, but the assessment item itself will only receive a pass mark.

You are to submit a single zipped project via the Canvas link, or via an alternative approved method, that contains 4 folders. There must be a folder for each of the required game projects and an additional folder for the debugging exercise. Each folder should contain:

* A folder containing an executable build of the project that runs error-free external to any IDE, with all required assets and resources needed for executing.
* A folder containing the project source, **excluding temporary build files**, which can be successfully compiled by an assessor without errors.

# Grading Rubric

Your assessment will be graded based on the grading rubric below and feedback will be given to the student as required.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Marks** | **High Distinction** | **Distinction** | **Credit** | **Pass** | **Fail** |
| **Tic Tac Toe Game** | 20 | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes highly technical systems and implementations. (**20-17**) | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes advanced features and technical code. (**16-15**) | Application runs as expected, with no faults or errors. Source builds with no compile or link errors. Application includes non-trivial features. (**14-13**) | Application runs as expected and implements minimum required features. Source builds with no compile or link errors. (**12-10**) | Application does not fulfil the required features, crashes on execution, or does not compile. (**9-0**) |
| **Battle Arena Game** | 20 | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes highly technical systems and implementations. (**20-17**) | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes advanced features and technical code. (**16-15**) | Application runs as expected, with no faults or errors. Source builds with no compile or link errors. Application includes non-trivial features. (**14-13**) | Application runs as expected and implements minimum required features. Source builds with no compile or link errors. (**12-10**) | Application does not fulfil the required features, crashes on execution, or does not compile. (**9-0**) |
| **Number Guessing Game** | 20 | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes highly technical systems and implementations. (**20-17**) | Application runs as expected, with no faults or errors. Source builds with no compile or link warnings or errors. Application includes advanced features and technical code. (**16-15**) | Application runs as expected, with no faults or errors. Source builds with no compile or link errors. Application includes non-trivial features. (**14-13**) | Application runs as expected and implements minimum required features. Source builds with no compile or link errors. (**12-10**) | Application does not fulfil the required features, crashes on execution, or does not compile. (**9-0**) |
| **Debugging Exercise** | 20 | Application successfully builds and runs as expected, with many improvements including high levels of code refactoring to improve readability, functionality and future use of the code. (**20-17**) | Application successfully builds and functions as expected, with much of the application redesigned to improve performance and readability. (**16-15**) | Application successfully builds and functions as expected, with improvements made to the execution of the application. (**14-13**) | Application successfully builds and functions as expected. (**12-10**) | Application still contains compile or link errors. (**9-0**) |
| **Code Documentation and Coding Standards** | 20 | Code is fully commented when explanation is needed for a section of code. Function declarations include comments that describe parameters and return values. Code adheres to a coherent standard throughout the project. (**20-17**) | Code is commented when explanation is needed for non-obvious sections. Function declarations include comments that describe parameters and return values. Code adheres to a coherent standard through majority of the project. (**16-15**) | Code has few comments for obscure sections of code. Function declarations include comments that explain functionality. Code mostly adheres to a standard. Some external documentation provided. (**14-13**) | Code has few comments for obscure code sections. Function declarations include comments describing functionality. Code is neat and readable. (**12-10**) | Code is poorly written and follows no coding standard. Little to no commenting used. Obscure code has no comments describing functionality. (**9-0**) |