|  |  |
| --- | --- |
| ****Subject Name and CRN:**** | INFT560 – Introductory Data Structures and Algorithms  CRN - 72299 |
| **Assessment Type:** | Project |
| **Due Date:** | June 25th, 11:59PM AEST |
| **Weight:** | 50% |
| **Marks:** | 250 |
| **Learning Outcomes:** | 2, 3, 4 & 5 |
| **Canvas:** | <https://aie.instructure.com/courses/42/assignments/465> |
| **Topic:** | Game Project |

# Project Brief

You are to write a simple game application, written in C++, that demonstrates the use of Game State Management and custom data structures, including error handling techniques and testing.

# Project Features

You are required to maintain your game project with Version Control software, such as Git, SVN or Perforce.

You may make or remake any game project, graphical or not, as you are not being graded based on visual quality or game-play mechanics.

Your game must make use of the following features which you will be graded on:

**Game State Management**

The game must make use of game state management that manages any and all states of the game including, but not limited to, loading / splash / menu / in-game. You will be graded based on the robustness and complexity of your chosen method of game state management.

**Custom Data Structures and Design Patterns**

You game must make use of custom data structures for its containers, i.e. instead of using Standard Template Library containers you should use your own implementations. Your game should also make use of Design Patterns, such as Component, Prototype, Factory, Pooling and other such patterns. You may make use of the Singleton pattern; however, this specific pattern will not affect your final grade.

**Testing and Error Handling**

You game should make correct and frequent use of error handling methodologies. Assertions and checking error codes and function return variables is recommended, however you may also make use of Exceptions, provided that they are implemented in a way that allows for graceful shutdown of your application without memory leaks of any kind.

You are also to implement Unit Testing for your custom data structures. This should be included as an additional project within your game’s solution. You will be graded based on the completeness of your Unit Tests.

You are not being graded on the game mechanics or completeness of your game.

# 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 file via the Canvas link, or via an alternative approved method, that contains:

* A folder containing an executable run-time of your Game that can be run external to any IDE. This folder must not contain more than is required for executing the runtime and must contain all required resources and assets.
* 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** | **Mark** | **High Distinction** | **Distinction** | **Credit** | **Pass** | **Fail** |
| **Game State Management** | 75 | Game uses a sophisticated state management system, configured in a stack that allows for updating / rendering of states lower in the stack. (**75-63.75**) | Game uses an advanced state management system using state classes in a game state stack. (**63.75-56.25**) | Game uses an improved state management system, such as basic state classes. (**56.25-48.75**) | Game uses a basic state management system; such as switch statements. (**48.75-37.5**) | Game does not make use of game state management. (**37.5-0**) |
| **Custom Data Structures and Design Patterns** | 75 | Game uses more than two custom container classes and more than two Design Patterns. (**75-63.75**) | Game uses two or more custom container classes and two Design Patterns, excluding Singleton. (**63.75-56.25**) | Game uses two custom container classes and one Design Pattern, excluding Singleton. (**56.25-48.75**) | Game uses one custom container class and one Design Pattern, excluding Singleton. (**48.75-37.5**) | Game does not use custom containers or Design Patterns. (**37.5-0**) |
| **Testing and Error Handling** | 50 | Assertions used where appropriate in critical locations that can’t be handled without crashing the program. All return values are checked for accuracy. All custom container class methods are included in the Unit Test. (**50-42.5**) | Assertions used where appropriate. All return variables are checked for accuracy. Unit Testing included for majority of methods for both custom container classes. (**42.5-37.5**) | Unit Testing included for both custom container class’ common methods. Basic error handling testing return variables used. (**37.5-32.5**) | Basic error handling testing return variables used. Basic Unit Testing included for common container methods. (**32.5-25**) | No testing or error handling included. (**25-0**) |
| **Completeness, Code Documentation and Coding Standards** | 50 | 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. (**50-42.5**) | 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. (**42.5-37.5**) | 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. (**37.5-32.5**) | Code has few comments for obscure code sections. Function declarations include comments describing functionality. Code is neat and readable. (**32.5-25**) | Code is poorly written and follows no coding standard. Little to no commenting used. Obscure code has no comments describing functionality. (**25-0**) |