**2805 ICT**System and Software Design  
**Assignment | Milestone 2**

**Student Name:** Mitchell Roles **Student ID:** s5132278 **Student Email:** [Mitchell.roles@griffithuni.edu.au](mailto:Mitchell.roles@griffithuni.edu.au)

Contents

[Summary of Requirements 3](#_Toc50812770)

[Product Use Cases 3](#_Toc50812771)

[Summary of Software Architecture 3](#_Toc50812772)

[Summary of Design 3](#_Toc50812773)

[Level of Sophistication Discussion 3](#_Toc50812774)

[Persistent Data Management 3](#_Toc50812775)

[Access Control 3](#_Toc50812776)

[Security 3](#_Toc50812777)

[User Interface 3](#_Toc50812778)

[Code Testing 3](#_Toc50812779)

[Version Control Historical Analysis 3](#_Toc50812780)

# Summary of Requirements

* A summary of the completed requirements
* A list of the still on-going requirements

Please refer to the first Milestone of this project to see the full list of project requirements. Within the scope of this progress report the current development focus has been on the introduction and improvement of core components of the game. These components including systems such as: scoring, timers, player movement, map updates and ghost packages (Placement, movement, pathing and AI).

The following table reflects the list of functional requirements as detailed in Milestone 1. It highlights the Functional Requirement, its priority, its status and any further more specific comments on its implementation.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Priority** | **Status** | **Comments** |
| FR-1 | 5 | Partial Completion | The GUI is largely functioning currently, as it can display most of the information the player needs to the screen. Still pending on the different graph displays and options menu. |
| FR-2 | 3 | Incomplete | A tutorial that will display the information necessary to educate the player on the game works. |
| FR-3 | 4 | Complete | The game correctly displays the players current score to the screen and the score is scale appropriately off the amount of time transpired within the game. |
| FR-4 | 5 | Incomplete | The condition the player needs to meet for them to win the game. |
| FR-5 | 5 | Incomplete | The condition the player needs to avoid, and will end the game if met |
| FR-6 | 4 | Partial Completion | The power pellets are displayed correctly within the maze and are recognised as separate entities in the game. Corresponding power up mode pending completion. |

**NOTE:**

* Remember to update these if they are completed.

# Product Use Cases

* A description of the use-cases relative to the requirements
* Classification of completed use-cases and pending ones

A close up of a map

Description automatically generated

Figure (Use Case)

The following table describes the use cases of the project and their current statuses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Use Case** | **Status** | **Description** | **Requirements Involved** |
| UC-1 | Play Game | Incomplete | Several use cases not complete. Encompasses all process required for a user to play the game | FR-1 to 6 |
| UC-2 | Quit | Complete | Allows the player to exit the game safely | FR-1 |
| UC-3 | Set Environment | Incomplete | Allows the user to select the environment of the game, (Hex, Regular and Graph). Currently only Regular mode is implemented with no way of switching | FR-1 |
| UC-4 | View Tutorial | Incomplete | Displays to the user information regarding how the game functions. Currently not implemented in any capacity | FR-1, FR-2 |
| UC-5 | Adjust Settings | Incomplete | Allows the user to adjust the individual settings of the game. Currently not implemented. | FR-1 |
| UC-6 | Load Level | Complete | Loads the maze to the screen, with all pellets and walls in place. Fully implemented. | FR-1, FR-2 to 6 |
| UC-7 | Load Player | Complete | Loads the player into the maze. Fully completed | FR-1, FR-2 to 6 |
| UC-8 | Load Ghost | Complete | Loads the ghosts into the level. Fully implemented. | FR-1 FR |
| UC-9 | Start Timer | Complete | Starts game timer counts the amount of timer has passed since launch. Filly implemented | FR-1, FR-3 |
| UC-10 | Game Loop | Incomplete | The loop that allows the game to function outside of the loading. Currently missing several use cases | FR-1, FR-2 to 6 |
| UC-11 | Game Lose | Incomplete | The condition that if met will cause the player to lose the game. Currently not Implemented | FR-1, FR-5 |
| UC-12 | Score | Complete | A value that will be increased when ever the player earns score. Currently fully implemented | FR-1, FR-3, FR-4 |
| UC-13 | Reset Game | Incomplete | Will reset the game upon win or lose. Not implemented | FR-1, FR-4, FR-5 |
| UC-14 | Game Win | Incomplete | The condition that if met will cause the player to win the game. Currently not implemented. | FR-1, FR-4 |
| UC-15 | Ghost AI | Incomplete | A set of algorithms that will cause the ghosts to chase the player throughout the level. Not implemented | FR-1, FR-5, FR-6 |
| UC-16 | Player Movement | Complete | Allows the player to move within the maze. Fully implemented. | FR-1 |
| UC-17 | Display Score | Incomplete | Displays the players final score, after winning or losing. Not implemented | FR-1, FR-3, FR-4, FR-5 |

# Summary of Software Architecture

* Summary of software architecture being used and its current feasibility
* Describe the main language, platforms and tools used

## Architectural Software Pattern

The software architecture of the system is based loosely on the Model View Controller (MVC) pattern. With this architectural pattern the system components present can be split into three categories, Model, View and Controller types. The Controller components being responsible for handling user input and delegating the tasks to other systems based on those inputs. The Model components are the internal representation and storage of data regarding the state of the system. Lastly, View components display the relevant information to the user and handles capturing the user’s input.

Using C++, the Graphical User Interface (GUI) was designed to capture all relevant inputs from the user. These inputs include actions such as keystrokes, and context sensitive mouse clicks. The GUI is the representation of the View portion of the MVC pattern, capturing inputs and displaying relevant data. These inputs are then given to the Controller portions of the program which then responds with the required actions, such as moving the player character. Some of these actions will directly affect/change the Model portions of the game, such as eating a pellet, and will require the View to be updated accordingly, removing the pellet from the map and updating the score.

## Development Tools

|  |  |  |
| --- | --- | --- |
| **Tool Name** | **Type Used** | **Reasons** |
| Programming Language | C++ | C++ is the language the development team was most familiar with. It was justified that the familiarity would justify not using a more specialised game design toolset |
| IDE | Visual Studio Community/Code | It is an IDE that smoothly supports the inclusion of multiple libraries. |
| Version Control | Git/Git Hub | Git is the Version Control tool that was most familiar with the development team. |
| Modelling Tools | Eclipse/Papyrus | Was the recommended tool to use for modelling software related tasks |
| DirectMedia Layer Framework | SDL2 | Is the only library that allows for DirectMedia layer interactions for C++ |
| Document Generation | Doxygen | Was the recommended tool to use to automatically generate documentation for code. |

# Summary of Design

* Describe design goals
* Provide:
  + Class Diagrams
  + Dynamic Model
  + Subsystem Decomposition

The desired outcome of the design was to create a system where game components were separated out as much as possible, the system can support extra additions and to alleviate any problems that may occurring during maintenance or updates.

Separating out the components will reduce the overall programs dependencies on said components and allow it to function without them; for instance, the game is capable of functioning without the ghosts. This helps to reduce the amount of coupling within the system and provide increased cohesion between the classes. This is achieved through ensuring that each “component” is capable of supporting itself and does not need to rely on any outside sources to supply its core functionality. This separation also allows new features to more easily be added to the overall system. As there is no dependency on other components, new ones can simply be added to the controller system when needed. Maintenance is also made easier through separation, as finding the source of any issues does not need to be tracked through multiple components at a time. The tests simply need to be done at the controllers level and tracked accordingly when the unexpected behaviour arises.

# Level of Sophistication Discussion

* Level of sophistication regarding Persistent Data management, Access control, security and User Interface

## Persistent Data Management

## Access Control

## Security

## User Interface

# Code Testing

* Describe what has been tested, what testing plan has been performed and what will be tested in the future
* Describe how testing will ensure the quality of new functionality does not regress

# Version Control Historical Analysis

* Extract version control logs
* Analysis commit history