**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 4](#_Toc51343429)

[Product Use Cases 5](#_Toc51343430)

[Summary of Software Architecture 6](#_Toc51343431)

[Architectural Software Pattern 6](#_Toc51343432)

[Development Tools 7](#_Toc51343433)

[Summary of Design 8](#_Toc51343434)

[Development Issues 8](#_Toc51343435)

[Static Issues 8](#_Toc51343436)

[Dynamic Issues 8](#_Toc51343437)

[Class Diagrams 10](#_Toc51343438)

[Dynamic Model 12](#_Toc51343439)

[Sub-System Decomposition 14](#_Toc51343440)

[PacManGame 14](#_Toc51343441)

[GameObject 14](#_Toc51343442)

[Ghost 14](#_Toc51343443)

[GameTimer 14](#_Toc51343444)

[GameScore 14](#_Toc51343445)

[UI 15](#_Toc51343446)

[MapTile 15](#_Toc51343447)

[TextureManager 15](#_Toc51343448)

[Level of Sophistication Discussion 16](#_Toc51343449)

[Persistent Data Management 16](#_Toc51343450)

[Access Control 16](#_Toc51343451)

[Security 16](#_Toc51343452)

[User Interface 16](#_Toc51343453)

[Code Testing 17](#_Toc51343454)

[Version Control Historical Analysis 18](#_Toc51343455)

[Analysis of Effort 19](#_Toc51343456)

[Code Screen Shots 20](#_Toc51343457)

[Main 20](#_Toc51343458)

[PacManGame 21](#_Toc51343459)

[GameObject 26](#_Toc51343460)

[TextureManager 30](#_Toc51343461)

[TileMap 31](#_Toc51343462)

[Score 37](#_Toc51343463)

[GameTimer 39](#_Toc51343464)

[UI 40](#_Toc51343465)

[Ghost 43](#_Toc51343466)

# Summary of 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 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. |

# Product Use Cases

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

## 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

## Development Issues

### Static Issues

The desired outcomes of the design were: creating a system where game components were separated out as much as possible, designing the system so that it 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 controller’s level and tracked accordingly when the unexpected behaviour arises.

### Dynamic Issues

One such issue that was given more attention then others in development was player movement. Player movement is key component in the game’s functionality, as it represents the strongest connection the player has to the game and if found lacking the risk of the player losing interest increases dramatically.

The main cause of this strife is that the player movement can be handled two ways: tile-based movement or dynamic position updating. Tile-based movement is the simplest, the player character is directly moved to the next “free” adjacent tile within the maze. Using this method eliminates the need to perform any collision calculations as the players positions is always within the bounds of the maze. This means that the system simply needs to detect if the next adjacent tile is moveable or not. The main issue with this method is that the characters movement is too fast for the player to comfortably react and make decisions.

Dynamic position updating is where the player character is moved independently of the maze. The method allows for the most amount of fine tuning and can easily replicate a character speed that is comfortable for the player. However, using this method require collision detection algorithms to determine where the player is, and what tile they are about to move to. This created a window for unexpected behaviour to occur, where the player character would ignore the mazes construction and allow for completely free movement

To solve this issue a conjunction of the two methods was employed. This new method uses the maze’s tiles to determine the player’s next position and moves the player dynamically, as seen in Figure 2. This method also solves the collision issue with dynamic player movement, by introducing a conditional check to determine if the player is currently transitioning to a new tile. If the player is transitioning, the system will not allow any new direction inputs to be processed, as seen in Figure 3.

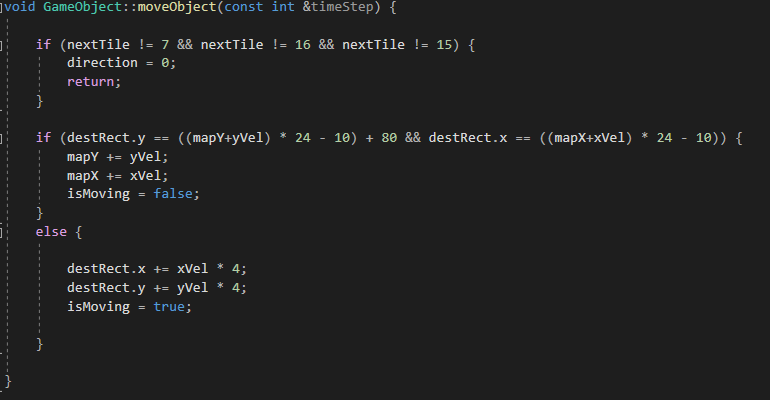


Figure (Player Character Movement)

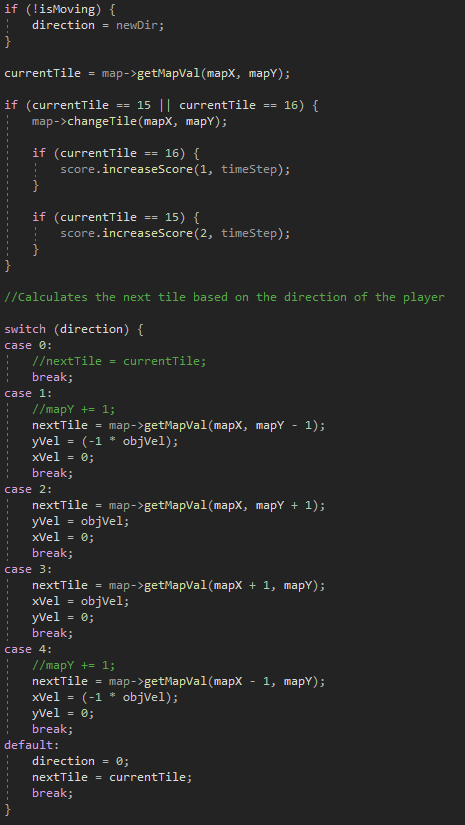


Figure (Direction Processing)

## Class Diagrams

The following class diagram was created using Visual Studio’s Code Tools and reflects the current state of the system. However, it does lack the nuances of other UML creation software and simply reflects the state of the system and not entirely accurate.

A close up of a map

Description automatically generated

Figure (Class Diagram from code)

A screenshot of a cell phone

Description automatically generated

Figure (Code Accurate Class Diagram)

## Dynamic Model

The following diagram highlights the high-level functionality of the system. The upon running, the game will initialise the game object and proceed onto the loop. Within the loop the game will be listening for any events the user might make. These events can take the shape of keystrokes and mouse clicks. The loop will also continuously update the game as well which will take the result of any events that might have taken place. Finally, the game loop will constantly render all of the game components to the screen.

A screenshot of a cell phone

Description automatically generated

Figure (High Level Dynamic Model)

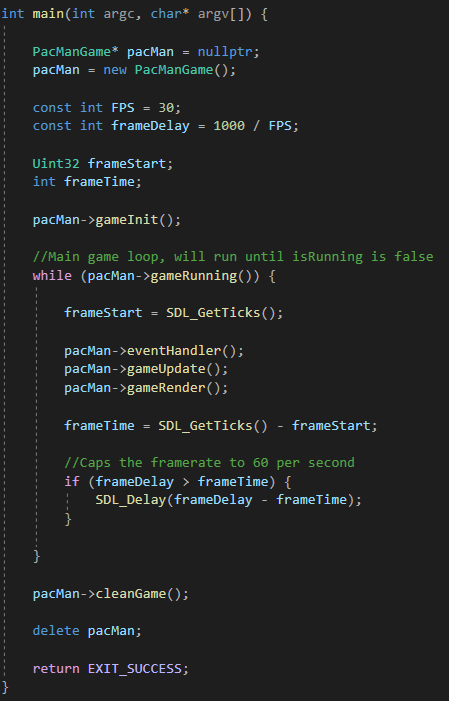


Figure (main code)

A screenshot of a cell phone

Description automatically generated

Figure (Sequence Diagram)

## Sub-System Decomposition

The following diagram is a high-level representation of the subsystems contained within the game. It highlights the general layout of the system and what attributes are dependent on what classes.

A screenshot of a map

Description automatically generated

Figure (Subsystems)

### PacManGame

The main body of the game, it is the class that handles and updates all other classes. It is responsible for: Initialisation of the game/classes, Interpreting player actions, updating objects (as per events), calling object render methods and object destruction. The PacManGame class is defined within the main function and all its methods are called continuously within the game loop; this will happen indefinitely until the user quits the game. Due to its composition, the class is capable of working without other the other classes, it just means that the game will be missing certain functionalities.

### GameObject

GameObject is the class the handles the player character, Pac Man, within the game. It is responsible for: loading the character sprite sheet, moving the player, rendering the character and animating the character.

### Ghost

The Ghost class is the encapsulating class for the computer opponents of the player the ghosts, Blinky, Inky, Pinky and Clyde. It handles the same actions as GameObject, for the ghosts, with the addition of an AI package to decide the movements of the ghosts, rather than receiving input from the player.

### GameTimer

GameTimer is responsible for keeping tack of the time elapsed after the game has started. It uses the SDL libraries to keep track of the number of “ticks” have passed, so it can remain independent from the processing speed of the computer.

### GameScore

GameScore is the class that tacks the players actions and awards points to the player, if they meet certain criteria. The scoring system utilises GameTimer to scale the amount of score the player earns after a 10 seconds have passed in game.

### UI

The UI class handles displaying non-critical information to the player. The information that is displayed to the player is the current timer and the score earned. UI will load a font from file and create a box for the information to be displayed within, afterwards it will convert the values contained within GameScore and GameTimer into a string and then rendering that information to the screen.

### MapTile

MapTile is the class responsible for creating the maze the game will take place in. To achieve this, it loads the information is requires from two files, a map file and an image info file, and the sprite sheet. The image info file contains the pixel positions of all the tiles within the sprite sheet. The map file contains the map size and what tile is located at each position within the map. After this the class will then assemble the map based on the prior specifications and render it to the screen.

### TextureManager

TextureManager is a simple functor class that is used to simply the development process. It was used to extract out a common command set and place it within quicker to use format that is easily accessible for all other classes that require it.

# Level of Sophistication Discussion

## Persistent Data Management

The current system has very little in the way of persistent data, as most defined attributes of each class are always being used in some capacity. The current lack of persistent data does reflect the relatively small size of the project and the development techniques used. These techniques simply refer to the fact that attributes are added to each classes structure by an as needed basis, instead of future proofing the classes and cleaning them later.

## Access Control

The system’s structure features relatively decent access control, but it could be improved. Each of the respective classes has good encapsulation of data, disallowing public access to its private attributes. There are some exceptions to this, but these usually are in the form of class general, static attributes. Through this access control, the amount of coupling between classes is reduced as no attributes can be directly and the only interactions allowed are through public methods. The place where access control could be improved is that the current system makes little use of object-oriented techniques such as class inheritance. Many objects are defined as global variables, due to some initial issues during the earliest stages of development, that are most likely not going to be fixed unless a severe problem arises because if it.

## Security

The security of the system is quite acceptable, with it having no unexpected behaviour outside of its target directory. The system also utilises safe memory management practises, coinciding with standard development practises. All allocated memory is destroyed when not in use, all data is not read beyond its bounds, etc.

## User Interface

The user interface, as it currently stands, is relatively simple. On a surface level it only features a few elements; that being: the map, the player, the score, the timer and the ghost. It does not feature any peripheral features such as menus (options, tutorial, etc). The elements of the User Interface that would typically be treated as complex objects, such as the map, are only counted as a single object on the screen.

# Code Testing

To test the program, no external tools were used. The system, as a whole, would most likely benefit from the introduction of specialised testing tools. These tools would be used as a means of insuring future quality and ensuring that the any new functionality does not compromise the previous established systems. Current all testing performed on the system has been done manually, through the use of variable testing, Visual Studio’s build in debugger and break point testing.

The following table reflects the test performed on the system through the previously stated means. However, this test in not ubiquitous and should not be taken as a reflection of the current state or quality of the present system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project** | | Pac Man | | |
| **Module** | | Game Functionality | | |
| **Requirement** | | Search for Customer (R1-6) | | |
| **Test Case ID** | | TC-1 | | |
| **Test Objective** | | To test whether the game’s GUI system is fully functional | | |
| **Test Date and Time** | | 16/09/2020 2:00pm | | |
| Step No. | Steps | Data | Expected Results | Actual Results |
| 1 | * Launch Game * Close Game | * No Arguments given | GUI will open, display game and close | As expected |
| 2 | * Launch Game * Wait 10 Seconds * Close Game | * No arguments given * No keyboard inputs | GUI will open, count to and past 10 and close | As expected |
| 3 | * Launch Game * Move character right * Move character left * Close Game | * No Arguments Given * Press right arrow key * Press left arrow key | GUI will open, player character will move to the right, the to the left and close | As expected |
| 4 | * Launch Game * Collect 1 pellet * Close game | * No Args given * Press any arrow key | GUI will open, display game, player character will move, player character will collect pellet, score will increment by 10 and close | As expected |
| 5 | * Launch Game * Wait 12 seconds * Collect 1 pellet * Close game | * No args given * Press any arrow key | GUI will open, display game, player character will move, player character will collect pellet, score will increment by < 10 and close | As expected |
| 6 | * Launch Game * Close Game | * Any args given | System will discard the arguments; GUI will open and close | As expected |

# Version Control Historical Analysis

Using GitHub’s commit history features, the above information was extracted and represents the total work done on the project. Figure 9 displays information on the size of the commits over the working periods of the repository. Figure 10 and 11 both show the number of commits on different days. The difference between the two is simply that Figure 11 shows the highest activity period for the project.

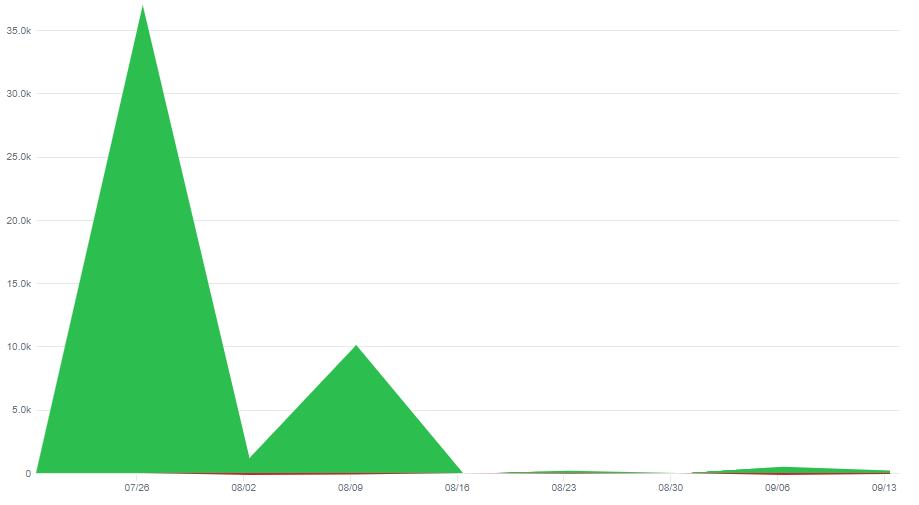


Figure (GitHub No. of: Additions and Deletions)

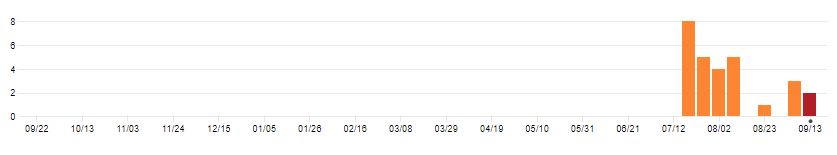


Figure (Commit History)

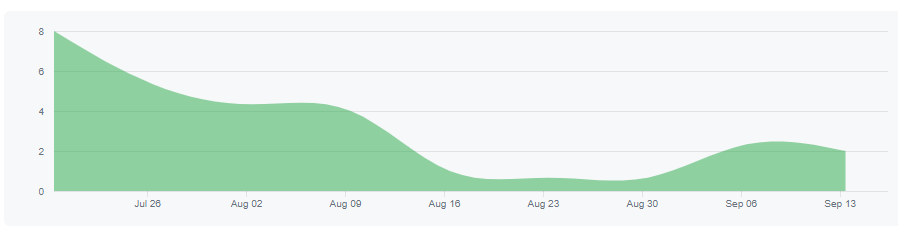


Figure (Commit History - Highest Activity Period)

## Analysis of Effort

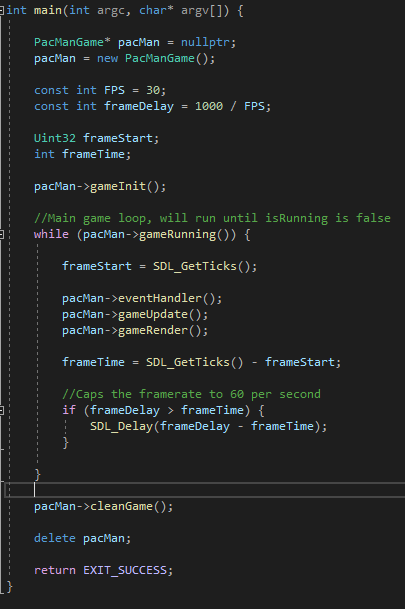
It can be interpreted from Figure 10 and 11 the period of work that given to the project. It can be seen that the project was worked on 25/06 through to 13/09, with the highest number of commits occurring on the 25/06 and petering off from there. The reason for this was due to the tumultuous state the project was in at that point, as the setting up the SDL2 library proved to be difficult. This problem resulted in a high number of additions to the project, as every working version of the system was committed to the repository. This period of frantic work, from 25/06 to 01/07, can also be observed with the number of additions to the repository in Figure 9, where over 35, 000 additions were made. Also, during this period of time, the decision was made to switch IDE’s to Visual Studio Community for its external library support. This change resulted in a settling of the work, where the SDL2 library was working consistently and the real work on the project could begin. Afterwards, from the period of 02/07 to 15/07, great progress was made and the game itself began to take shape; this is where Milestone 1 ended.

A lull in commits to the repository can be seen from 16/07 to 30/07, in figure 10 and 11, this lull is because other projects required attention at the time and were given greater precedence. Though some intermittent work was done, nothing truly noteworthy happened.

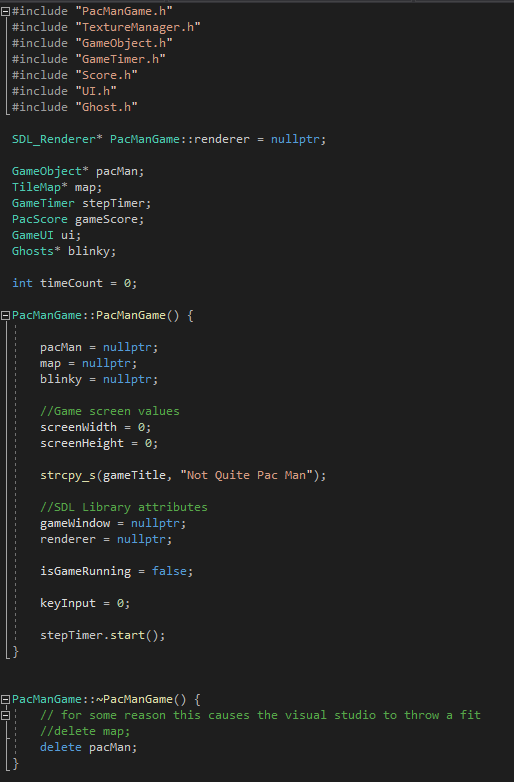
Then from 02/09 onwards, the work was again underway. Though never again reaching the franticness of weeks prior, a steady amount of commits to the repository were made. This reflects the more effective workflow that was developed for this project. Additions were concise and effective, with good progress being made despite the small amount of added code.

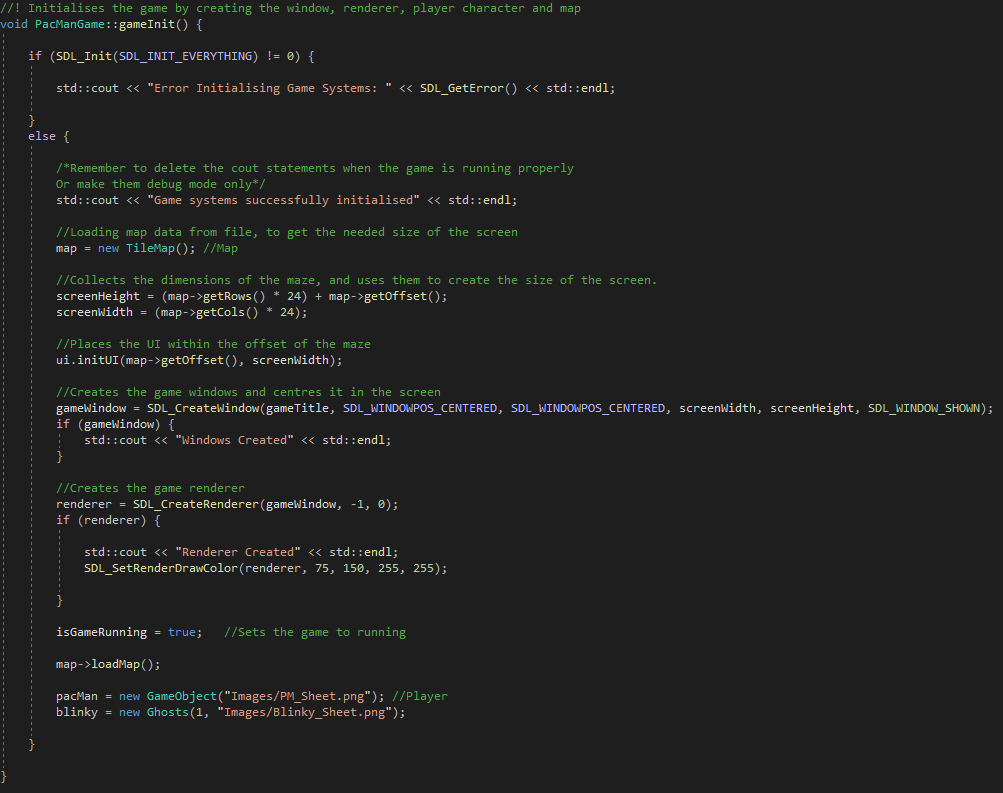
# Code Screen Shots

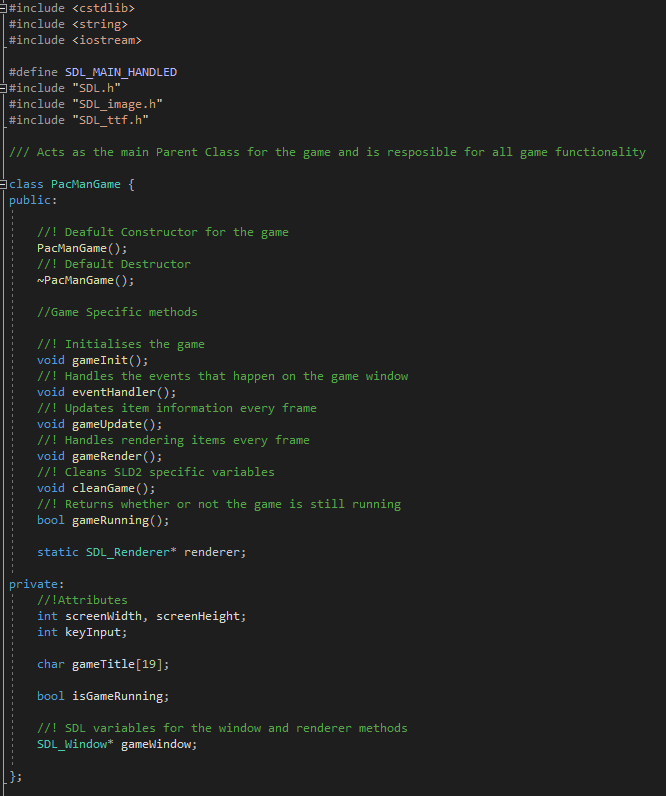
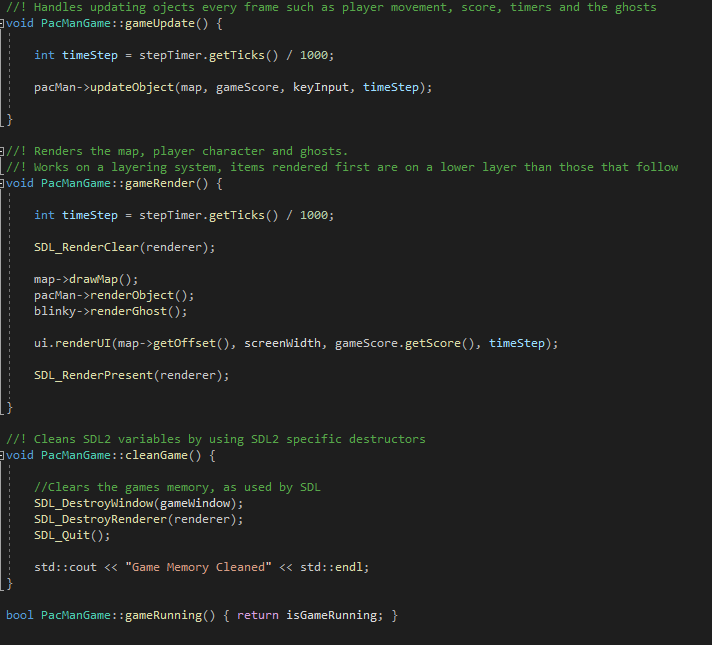
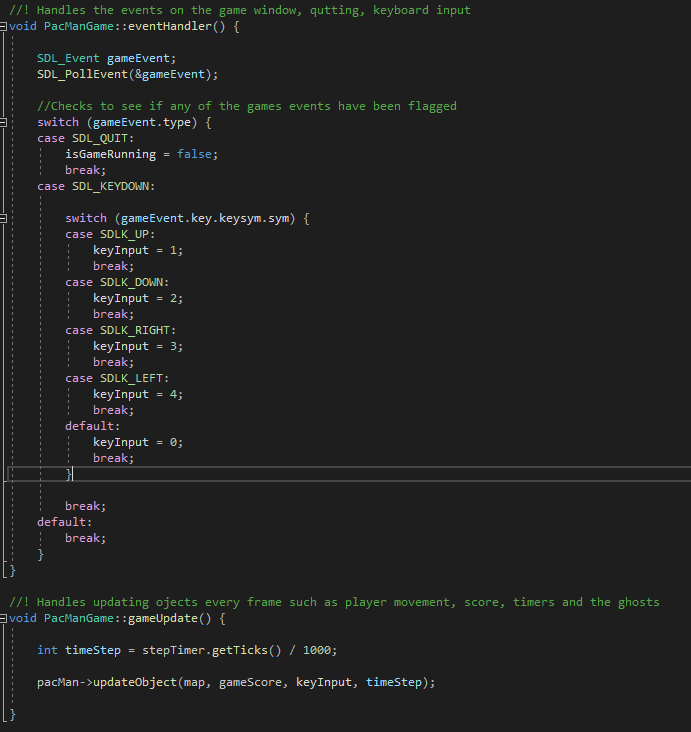
## Main



## PacManGame

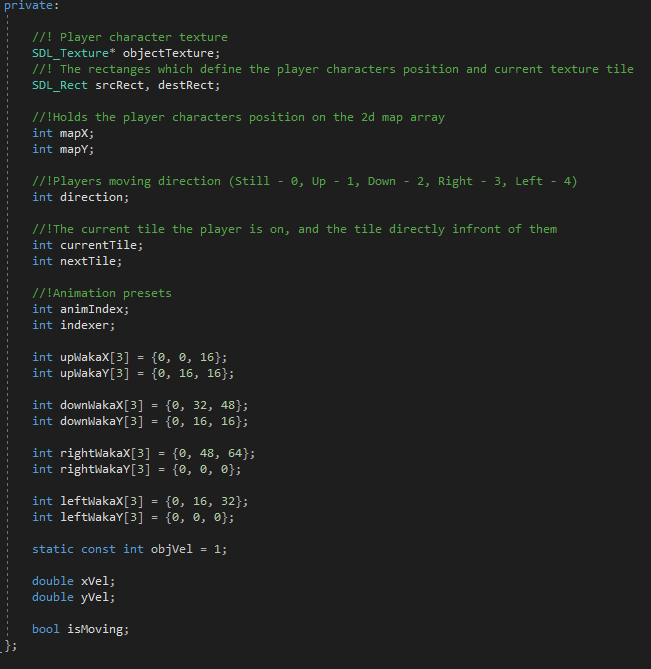
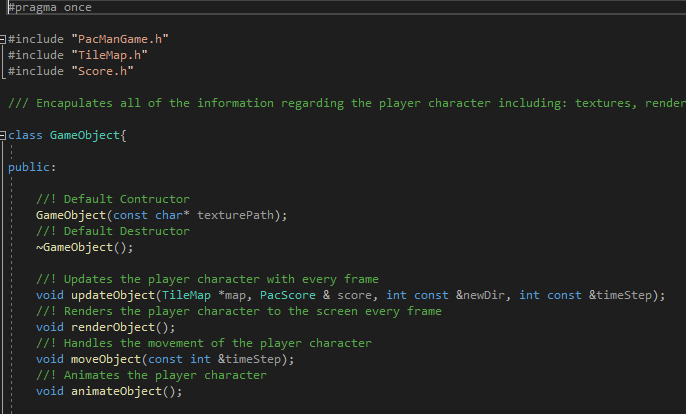
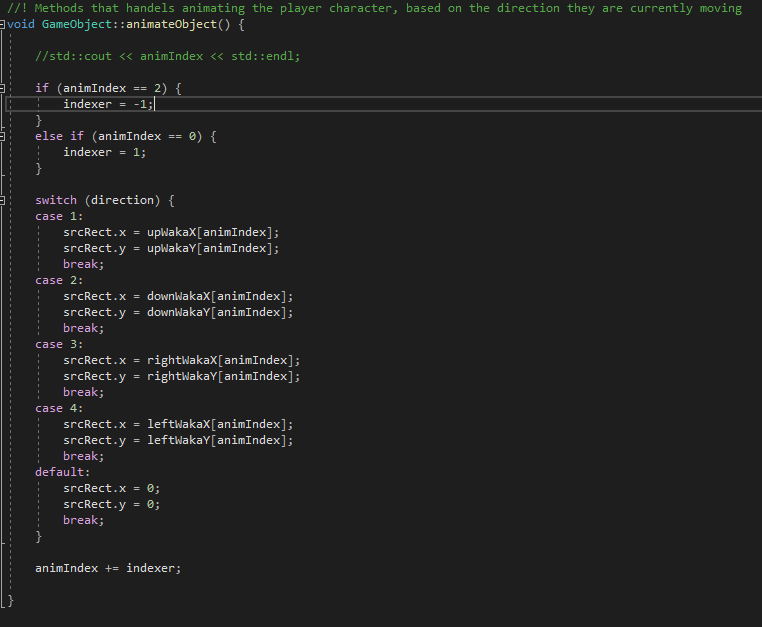
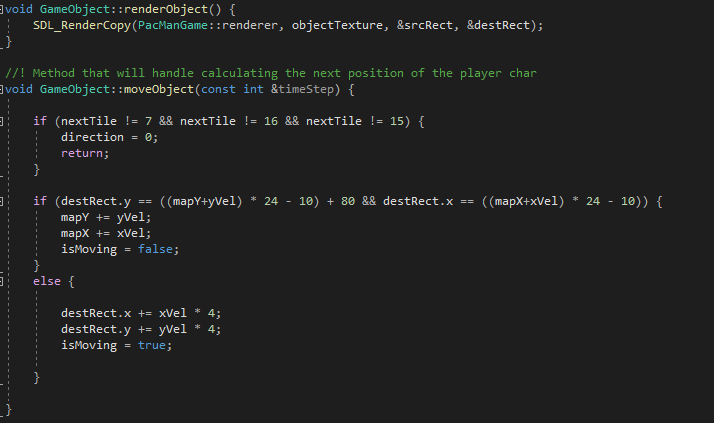
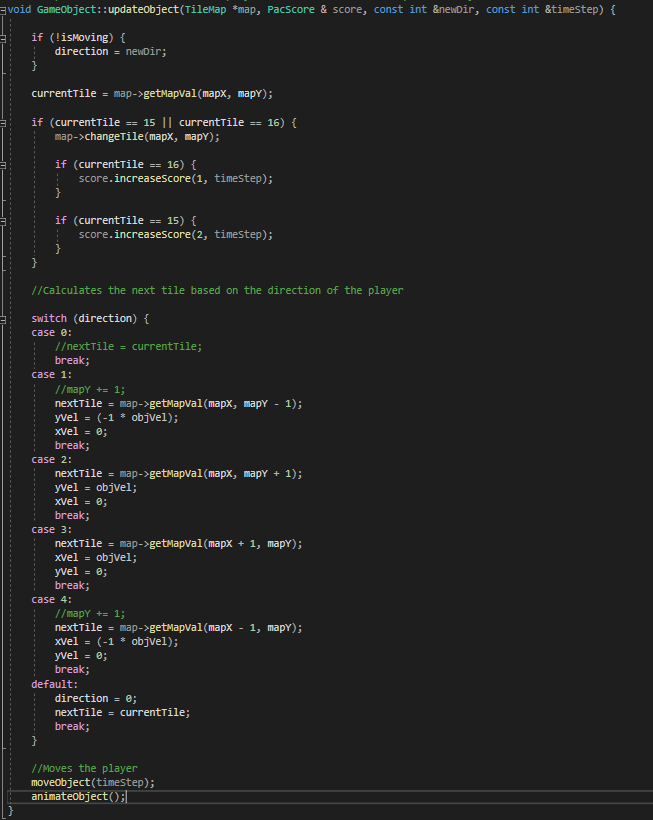




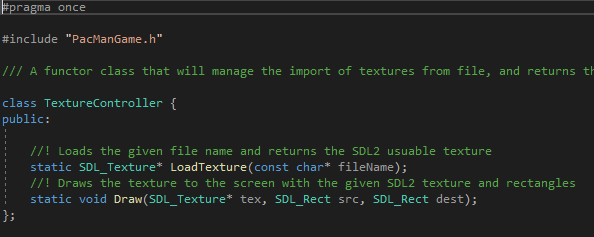
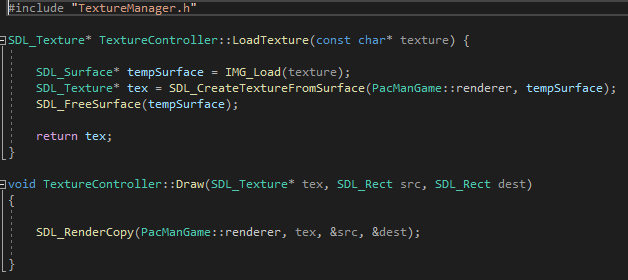


## GameObject

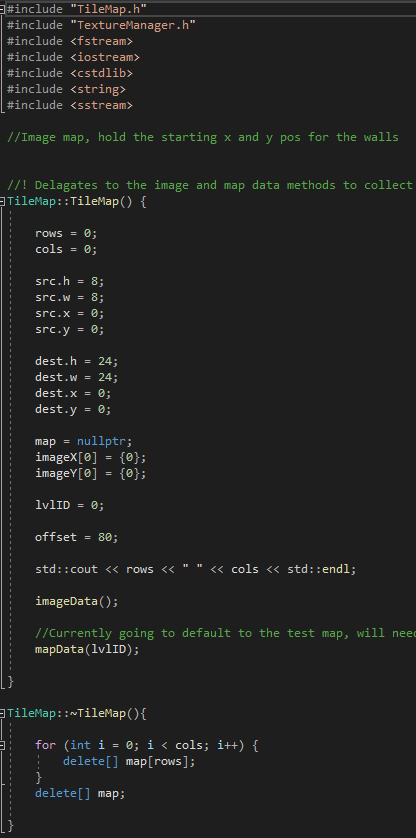


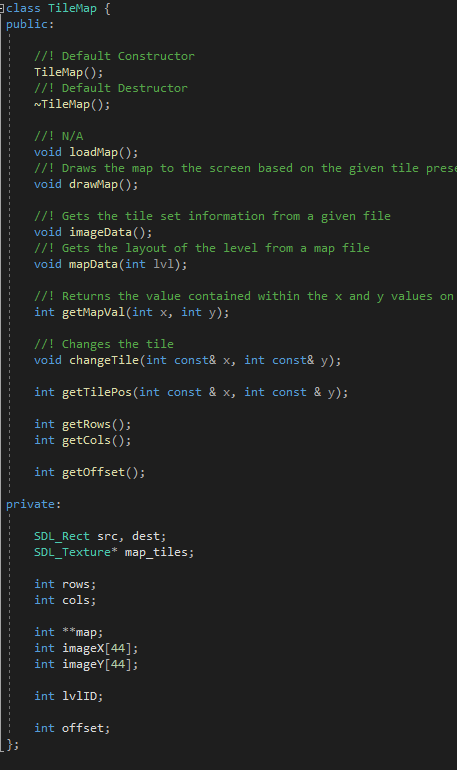
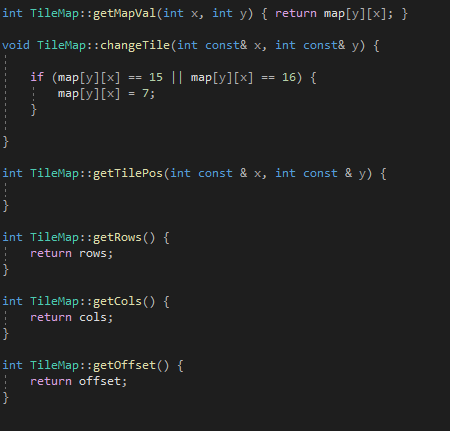
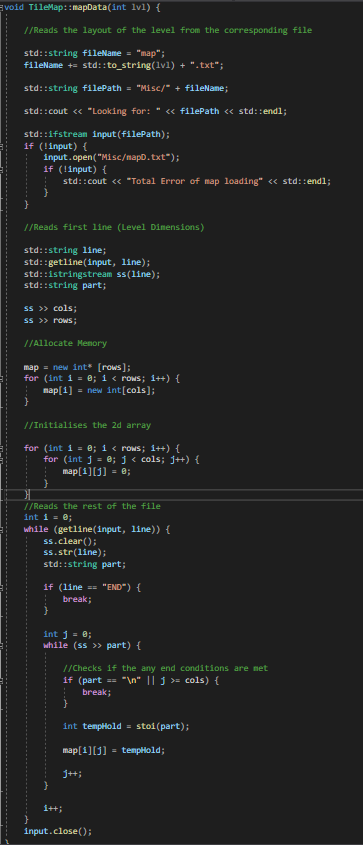
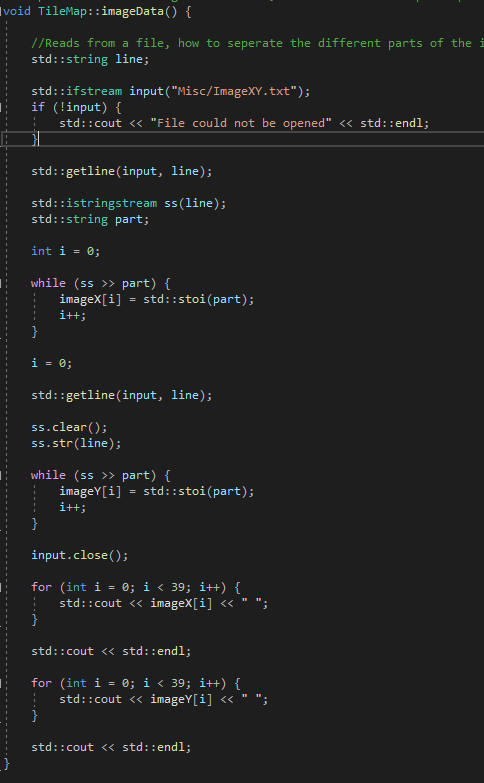
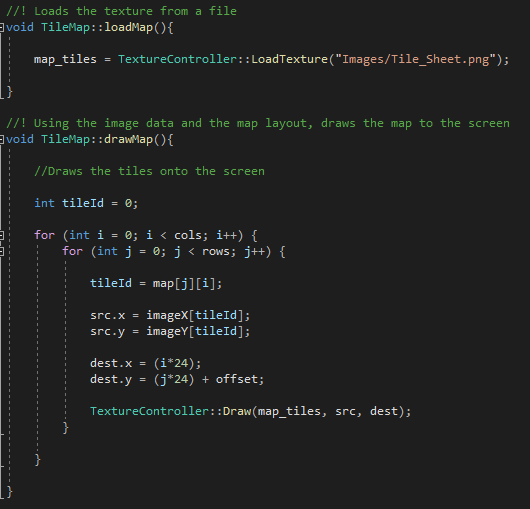


## TextureManager

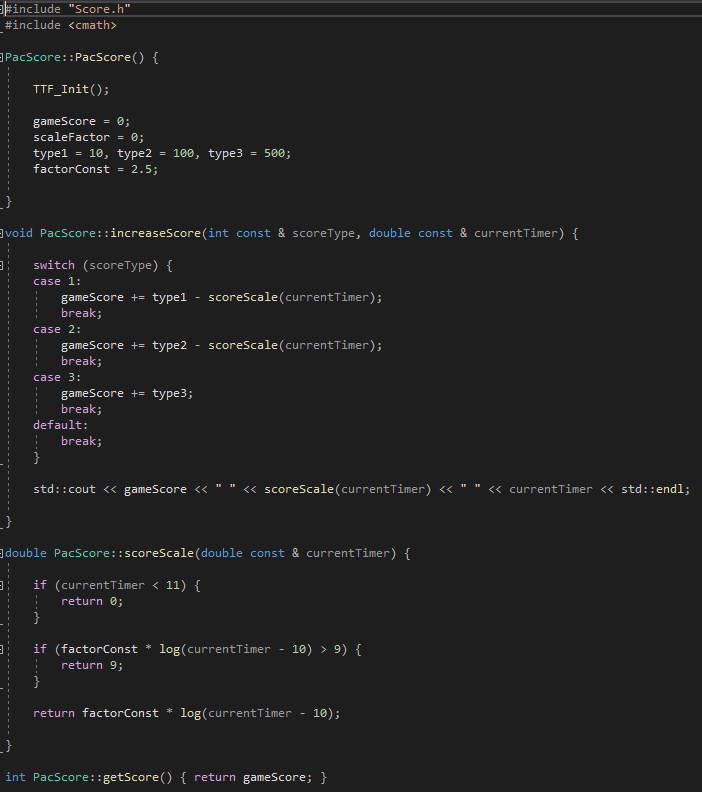


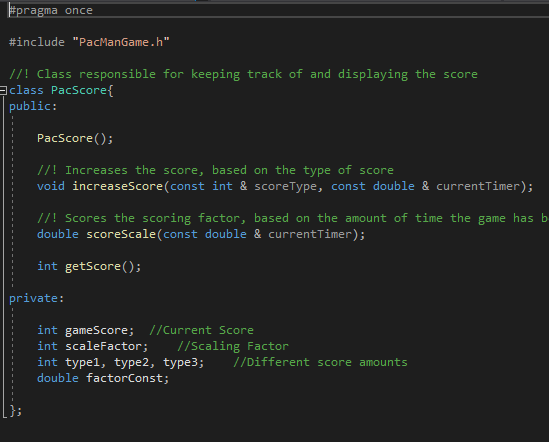
## TileMap



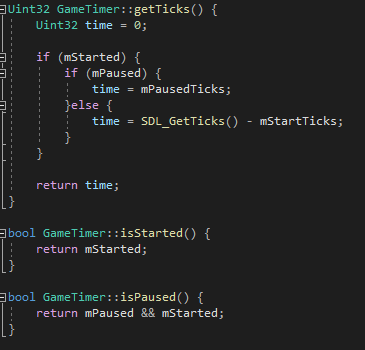
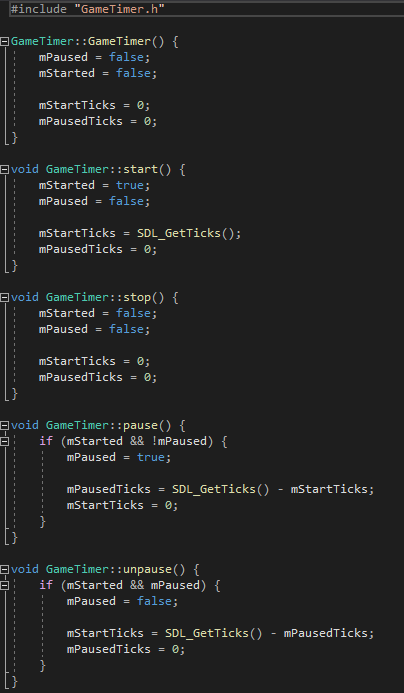


## Score

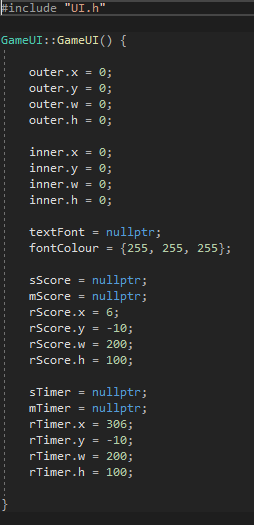


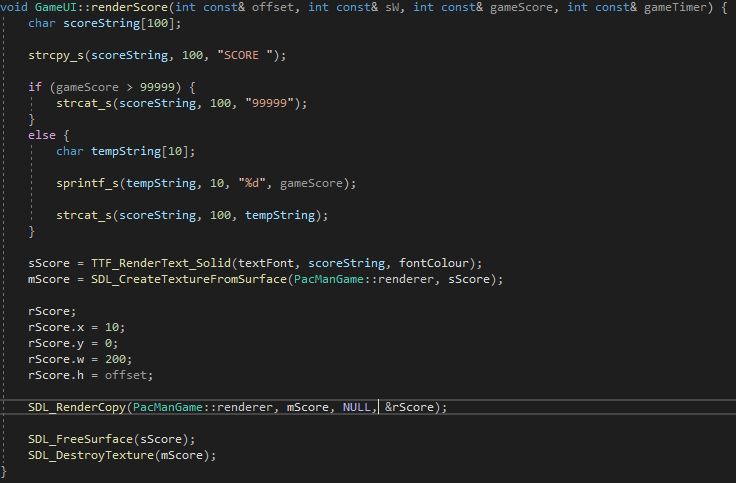
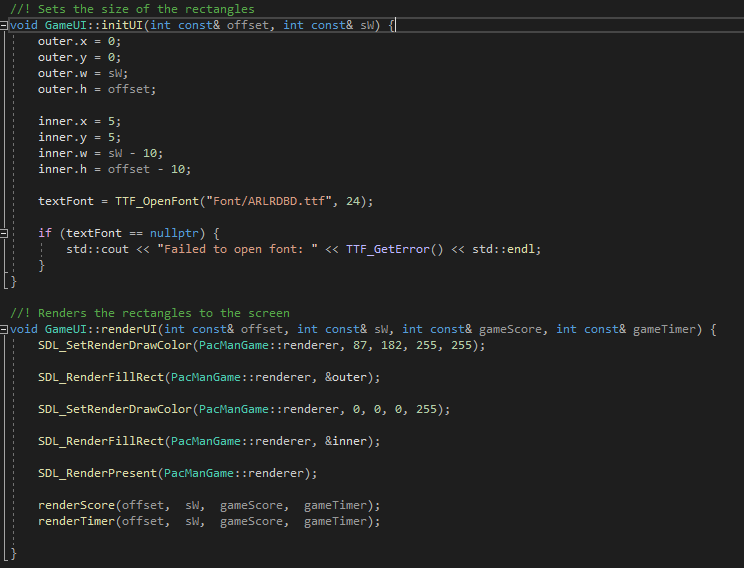


## GameTimer



## UI





## Ghost

