Sebastian perry

S5132483

2805ICT PACMAN Milestone 3

# At a glance

This document will serve as a progress report on the steps taken to implement the Pacman game from the perspective of being at the end of the development period

Contents

[At a glance 1](#_Toc51361389)

[Functional Requirements 2](#_Toc51361390)

[Non-Functional Requirements 3](#_Toc51361391)

[Constraints 3](#_Toc51361392)

[Product Use Cases 4](#_Toc51361393)

[Summary of software Architecture 5](#_Toc51361394)

[Summary of Software Design 6](#_Toc51361395)

[Map based calculations 6](#_Toc51361396)

[Display of map recreations and sprite sheets 7](#_Toc51361397)

[Dynamic Model 8](#_Toc51361398)

[Testing 8](#_Toc51361399)

[Version Control 9](#_Toc51361400)

[Method of compilation 9](#_Toc51361401)

[Efforts so far 9](#_Toc51361402)

[Efforts in future 9](#_Toc51361403)

# Implementation of Task 1

wwwwwwwwwwBelow is a list of the functional requirements that were generated during milestone 1 in addition to their

# Implementation of Task 2

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Implementation of Task 3

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Cross platform

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Software Version control

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 1

In the project many design principles were used to aid in the implementation of the project. First of all c++ was used for the implementation of the game which allowed for the following of the Principles of ***least privilege***through the use of private class contents, allowing the fulfilment of the ***failsafe defaults*** principle by enabling classes to hide unnecessary data from other classes. Furthermore, in following the principle of ***separation of concerns*** (where different modelling elements were separated for clarity), it became necessary for certain classes to be available in multiple places at the same time; it is this need which lead to the implementation of the singleton model with a few of the classes found within the project.

# Objective 2

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 3

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 4

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 5

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 6

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 7

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 8

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 9

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Objective 10

Below is a list of the functional requirements that were generated during milestone 1 in addition to their

# Functional Requirements

Below is a list of the functional requirements that were generated during milestone 1 in addition to their progress

generated of the functional requirements for the project.

|  |  |  |
| --- | --- | --- |
| Identifier | Functional requirement | Progress |
| F-R-1 | The game should offer a graphical interface through which the player can see Pacman, the ghosts, walls, and any other objects which get added to the game. | 75% the display manager class handles rendering to the screen (through SDL) and has been implemented using the singleton design pattern. As such all aspects of the code can display any of the sprites to anywhere on the screen, however all the parts that use display manager have not yet been completed |
| F-R-2 | Pacman and the ghosts should be able to move around the maze, moving at a constant speed in a constant direction until decided otherwise by the player or the A.I. | 50% based on tests on the display manager class, this is very possible and close to happening, however the game is currently experiencing breaking changes |
| F-R-3 | Pacman and the ghosts should collide with walls and consider them impassable | 40% the play space navigation class has been created and half implemented |
| F-R-4 | Pacman should be able to change his direction whenever, and the ghosts should decide their direction based on heuristic algorithms calculated every time they reach a crossroad. | 30% the map tiles have been designed with very distinct corridors and intersections to ensure that the implementation of this feature can be clean and deterministic |
| F-R-5 | If Pacman collides with a ghost, Pacman should lose a life and the game should restart | 0% as of tight now this has not been implemented however the play space class once finished should make this task quite easy |
| F-R-6 | Pacman should be able to collide with small pellets, which would have the effect of removing the pellet and raising Pacman’s score | 20% most of the key components of this is here, the pellets just need to be implemented by checking their collision with the player, this will be completed by implementing a generic entity collision detection function that will be useful for many other implementations |
| F-R-7 | Pacman should receive a penalty to his score depending on the amount of time it took to complete the level | 40% once F-R-6 has been implemented, this feature will just be a matter of having a logarithmic decay function on the score received that takes the score and the time since started as variables, the method of displaying the score to the player has already been completed |
| F-R-8 | The game should also have the ability to run on three different display / map settings, a square cell configuration, a hexagonal grid configuration, and an arbitrary grid configuration | 40% square implementation is almost complete and was implemented in such a way that graph implementation should only require some minor tweaking to a few of the classes, with some of them still working the same regardless. Hexagon implementation however will require major work to multiple files |
| F-R-9 | At least one ghost should move towards Pacman through the use of Dijkstra’s algorithm | 0% this is yet to be implemented, but once implemented should work on all three methods based on implementation |
| F-R-10 | The game should be able to randomly generate valid mazes | 75% due to the games structure this feature is mostly already complete in the easy and effective way of using tile sets, however as there are currently breaking changes being made this feature cannot be shown |
| F-R-11 | Pacman should be able to collide with large pellets, which would have the effect of removing the pellet and temporarily changing the Pacman ghost rule so that it is instead the ghosts which temporarily die, the will also try to get away from Pacman | 0% however this will be easily implemented once F-R-6 has been completed, it will be implemented in the same way except it will send different information and change a variable that decides who dies upon player / ghost collision |
| F-R-12 | If all pellets have been consumed, Pacman should gain a life and the game should restart | 0% this has not been started |
| F-R-13 | Players should be able to easily manipulate various game options (for example: map type, Pacman speed, ghost speed, map size) | 100% this has been implemented through the use of a config file which is read in upon execution |

# Non-Functional Requirements

Below is a list generated of the functional requirements for the project.

|  |  |  |
| --- | --- | --- |
| Identifier | Non-Functional requirement | Progress |
| NF-R-1 | Universality: the game should be able to run on both 32 and 64 bit versions of windows | 0% this has not yet been attempted, however, as this is all standard c++ code, using packages that have Linux compatibility, this should be reasonably simple to achieve |
| NF-R-2 | Reliability: the game should work without crashing, producing errors, or violating defined behaviours | 50% so far, during all tests the game has been running correctly and reliably without fail, an effort will be made to continue monitoring this |
| NF-R-3 | Ascetics: the finished game should look and feel as close to the original retro Pacman as possible within time limits and requirements (obviously this will not be possible with some of the aspects, but it would be preferable for the overall ascetics to match throughout) | 100% this has been completed by compiling a set of the original Pacman images and editing so that they can serve the function of being a sprite sheet. |
| NF-R-4 | Performance: the finished game should be able to run at a stable frame rate | 50% currently the game consistently runs at a stable 60fps, as implementation continues efforts will be made to ensure this stays the case |
| NF-R-5 | Useability: players should be able to change their control bindings and other game options via a file placed in the same directory as the executable | 100% this has be successfully implemented |

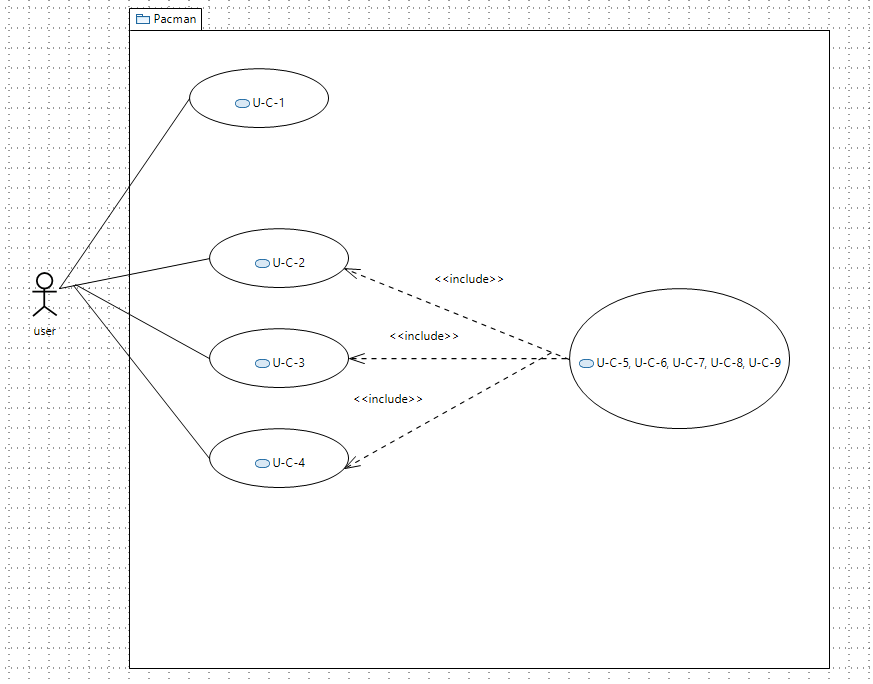
# Constraints

As of right now all previously listed constraints have been considered, worked within, or otherwise remained unbroken.

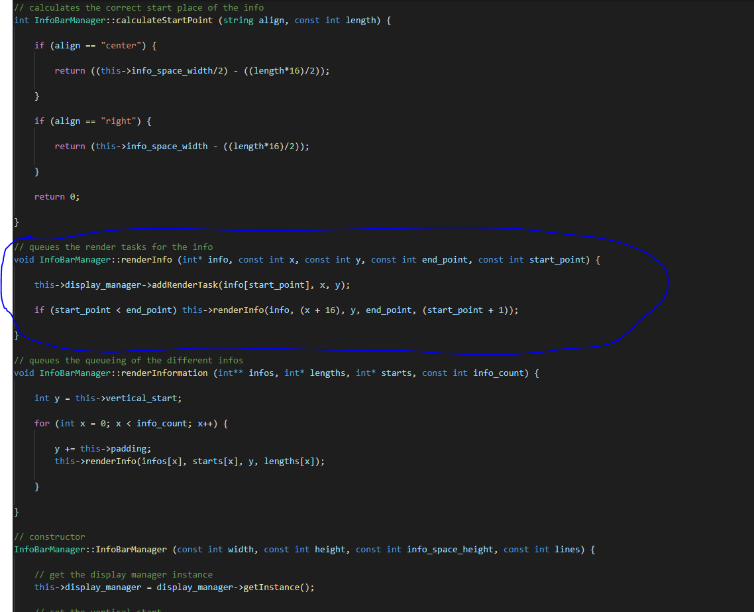
# Product Use Cases

Below is shown a list of various use cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier | Use Case | Status | Included in | Requirements |
| U-C-1 | Change options | Incomplete |  | N-F-R-5, F-R-13 |
| U-C-2 | Play Pacman Square | Incomplete |  | F-R-8, F-R-4 |
| U-C-3 | Play Pacman hexagon | Incomplete |  | F-R-8, F-R-4 |
| U-C-4 | Play Pacman graph | Incomplete |  | F-R-8, F-R-4 |
| U-C-5 | Ghost kills Pacman on collision | Incomplete | U-C-2, U-C-3, U-C-4 | F-R-5 |
| U-C-6 | Pacman gets points when picking up pellets | Incomplete | U-C-2, U-C-3, U-C-4 | F-R-6 |
| U-C-7 | Score increase decreases with time | Incomplete | U-C-2, U-C-3, U-C-4 | F-R-7 |
| U-C-8 | Pacman can get a big pellet which allows him to kill the ghosts | Incomplete | U-C-2, U-C-3, U-C-4 | F-R-11 |
| U-C-9 | Map can change in size, and randomly assemble | Complete | U-C-2, U-C-3, U-C-4 | N-F-R-5, F-R-13 |



# Summary of software Architecture

In the Project the codes’ structure can be explained with these two basic architectural patterns:

Model-View-Controller: all of the different classes can be grouped and split up into 3 separate groups, which are view, model, and controllers. View in this case would consist of the display manager class as it displays the view. Model would consist of the PlaySpace class as well as the various subclasses of the entity class as they model the actions of the characters. Finally, Controller would include all the others such as Game, EntityManager, InfoBarManager, etc as they are responsible for triggering the various behaviours in the other 2 categories.

Figure 1InfoBarManager acting as a controller for Display manager

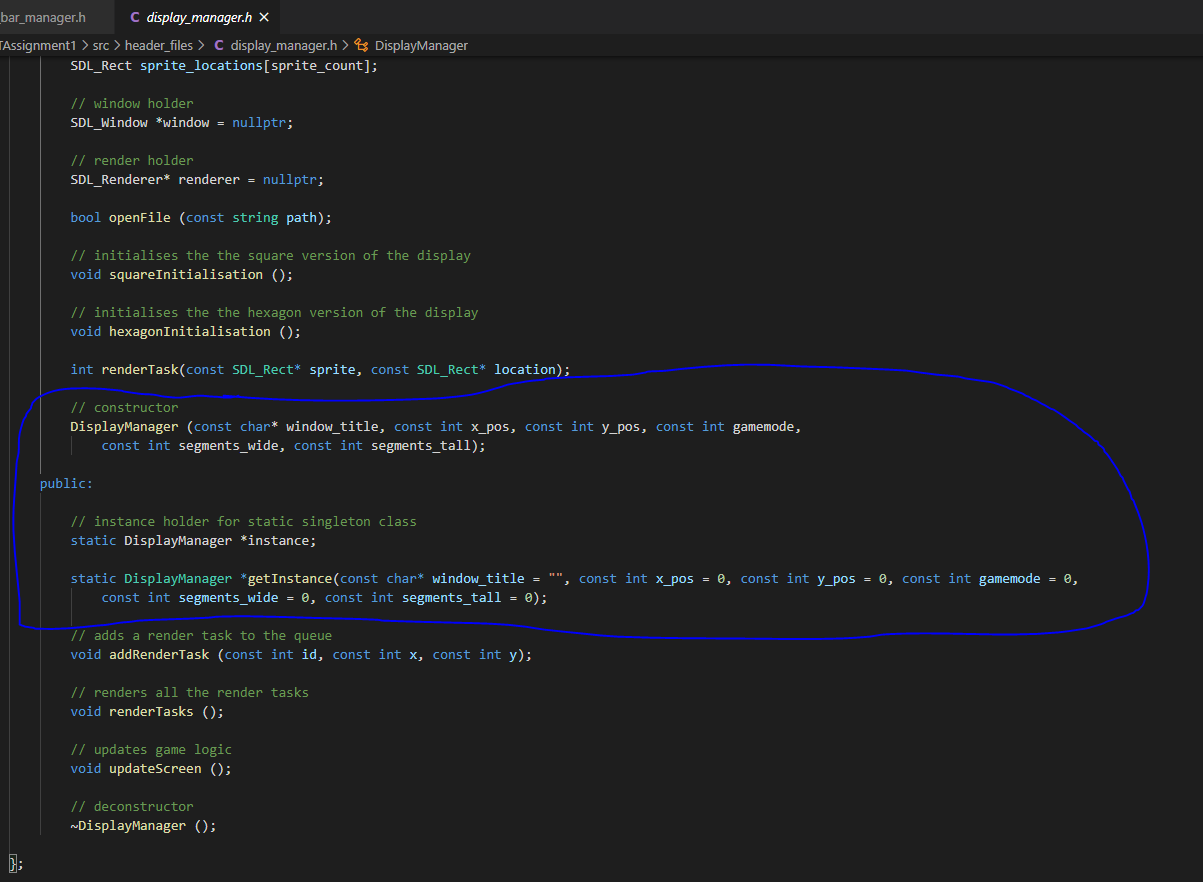
Singletons : Singletons are used for two vital part of the projects function. It facilitates the access of these vital classes from anywhere in the code on the same instance; for example, the DisplayManager class is very important for the program as it is the only why the program can render anything to the window open upon game initialisation. Because of the function it serves, DisplayManager, playspace, entityManager, and infoBarManager all need access to not only the class, but the same instant of the class too due to the difficulty in moving the renderer around. Play space is in the same boat as there are a few classes that require access to it, as such the singleton architectural pattern was instrumental in the project’s implementation.

Figure 2 singleton implementation of DisplayManager

# Summary of Software Design

Figure 3 the full static class diagram model of the project

## Map based calculations

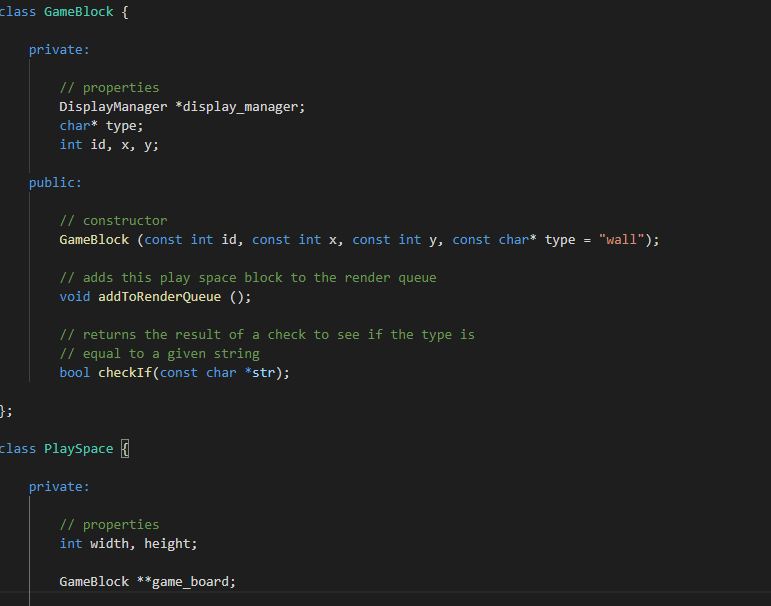
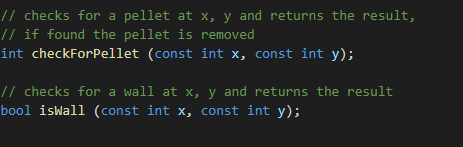
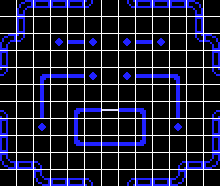
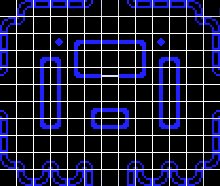
This game calculates the class PlaySpace for its map-based calculations such as navigation and collision with the various map elements. This goal is accomplished by containing a 2d array of ‘GameBlock’s (figure 4) which facilitates the storage of the map tiles, allowing other classes to query and test on the contents of various avaliable functions as shown below.

Figure 4 GameBlocks



## Display of map recreations and sprite sheets

As stated before the game is currently experiencing a breaking change that is preventing it from displaying it’s graphics, however the below sprite sheet and level recreations will be shown.

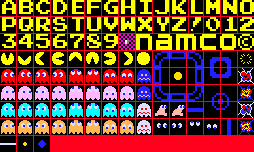
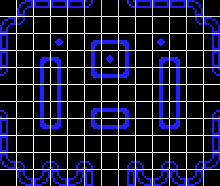
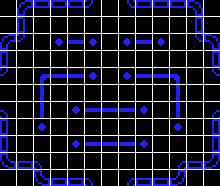
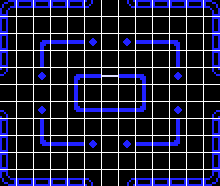


Figure 6 map recreations

Figure 5sprite sheet

In order to create the maps shown to the left sprites are defined by location and stored in an array (figure 8). The indexes of this sprite locations can be later called to display and of the sprite, or in this case entire maps(figure 7)

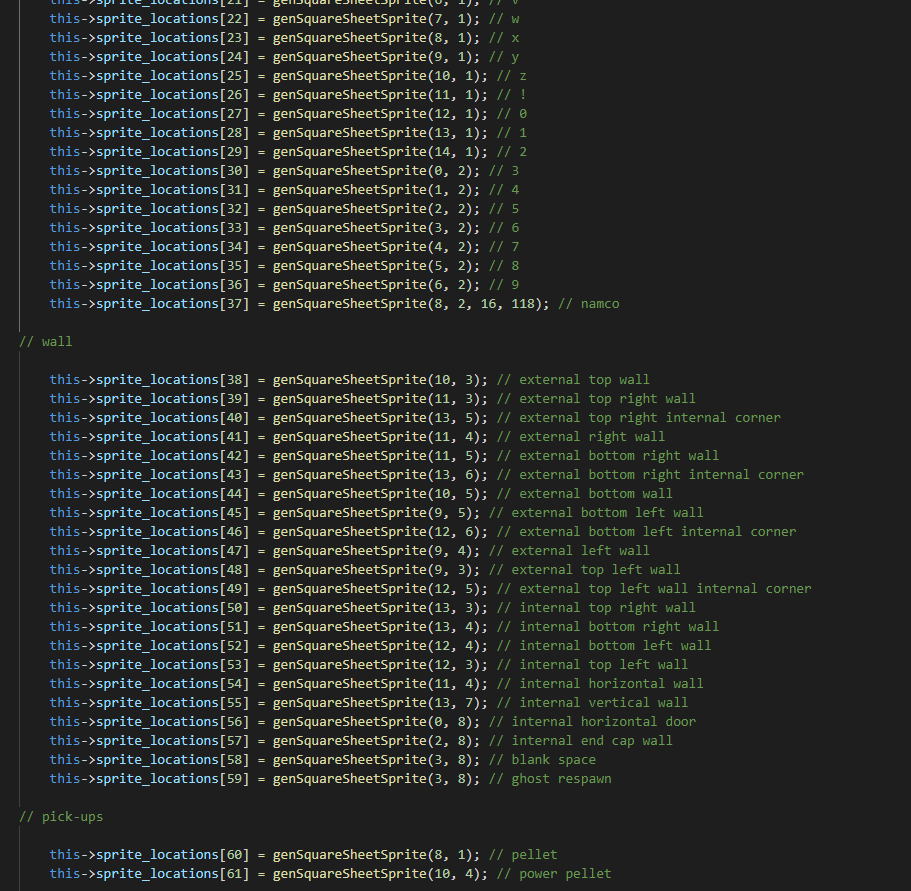
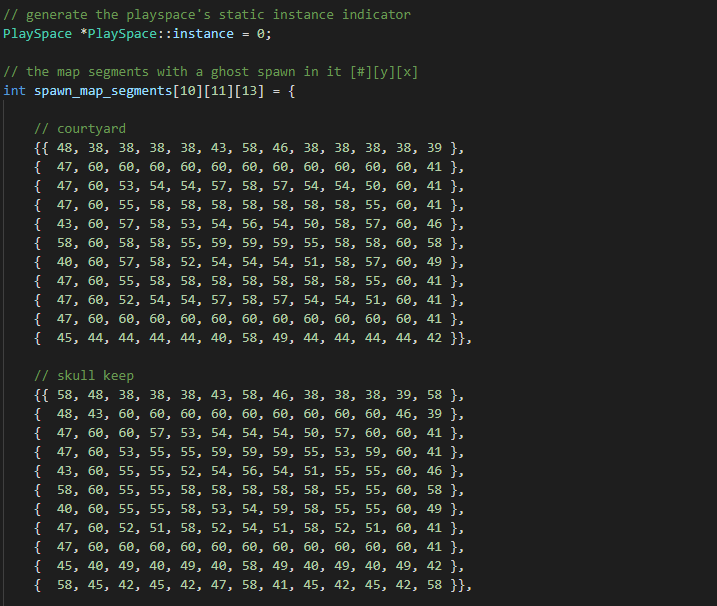


Figure 7 maps stored as a 2d array of ints

Figure 8 stored sprites

## Dynamic Model

Figure 9 dynamic model of the main implementation

# Testing

A small amount of testing has been conducted, however due to the fact that very few of the product use cases aren’t in a functional state it has been virtually impossible to conduct large scale tests. As more progress is made on the game the defined product use cases and functional requirements will be tested to ensure the game is functioning correct and reliable.

# Version Control

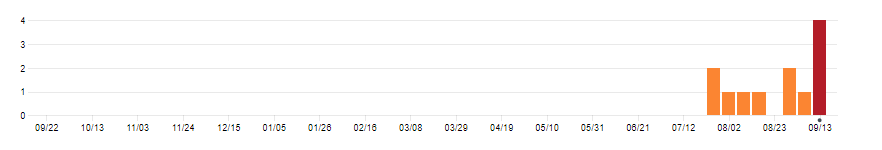


Figure 10 commits by week

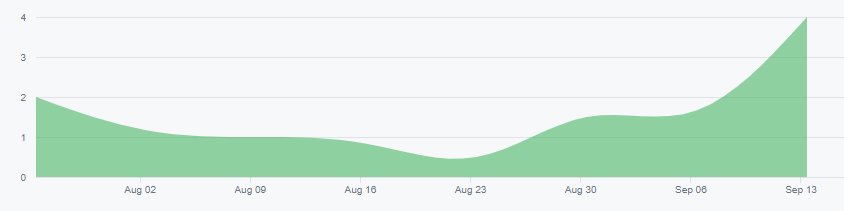


Figure 11 commits over time

## Method of compilation

As all version control has been left to the functionality of git and furthermore GitHub, all commit information has been stored on the repositories’ profile. To obtain the information the profile was temporarily set to public allowing for the acquisition of the data shown above.

## Efforts so far

Over 12 commits a lot of work has been completed, the project is awaiting the implementation of 2 file before it is expected that both the square and graph implementations of the game are working. With that being said it would have been far better if more effort was made to more frequently commit as it is predicted it could make a positive impact on bug fixing and debugging in the case of a breaking change. As can be seen in the graphs there was a slight hiccup around the 23d of august will a lot of difficulty was experienced integrating the SDL image package with the code; the package was difficult in its application and positioning so that the rest of the project could access it’s vital functionality. During this time work had to be done learning the methods of makefile creation was necessary as the project had become too large and complicated to continue using the previous method of .bat command line files (which had to be rewritten every time a new file was created).

## Efforts in future

As mentioned before, an increased effort will be made to commit more often to increase productivity. During development the version control tab in visual studio code was discovered and utilised in the debugging and diagnostics of breaking changes when integration testing. On the subject of project completion schedule is being mostly upheld. On the subject of future progress, it seems that despite the multiple hiccups (that were experienced around the 23rd) as long as the current efforts are at the very least maintained, the project should be finished by the due date of the 3rd milestone.