2805ICT PACMAN MILESTONE 2

Sebastian perry

S5132483

At a glance

This document will serve as a progress report on the steps taken to implement the Pacman game

Contents

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

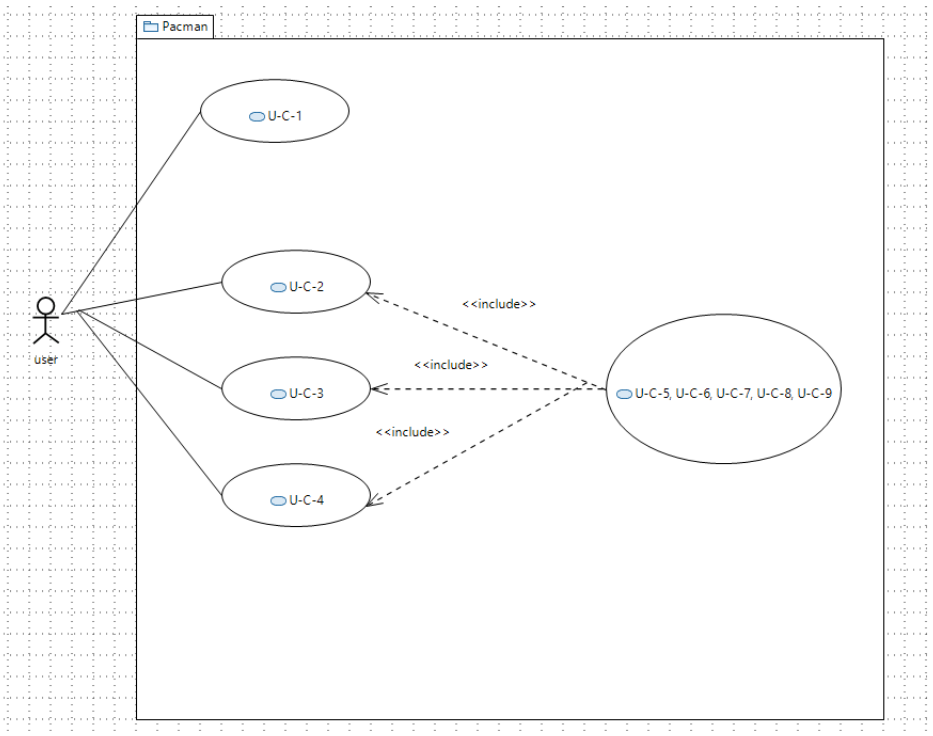
Constraints

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

Product Use Cases

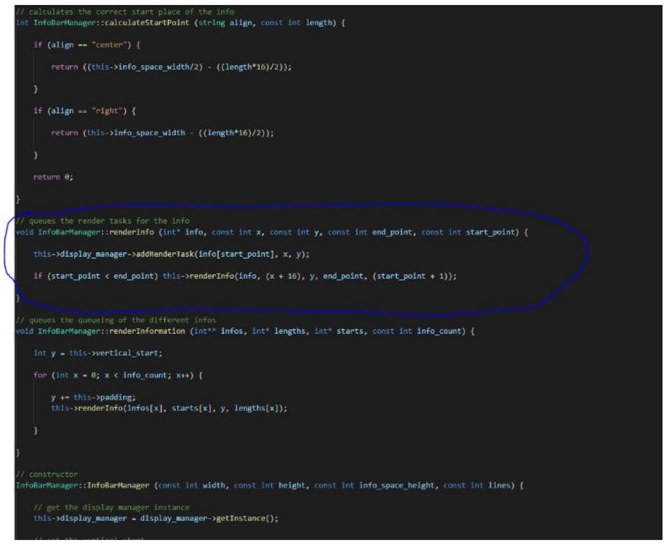
Below is shown a list of various use cases

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:

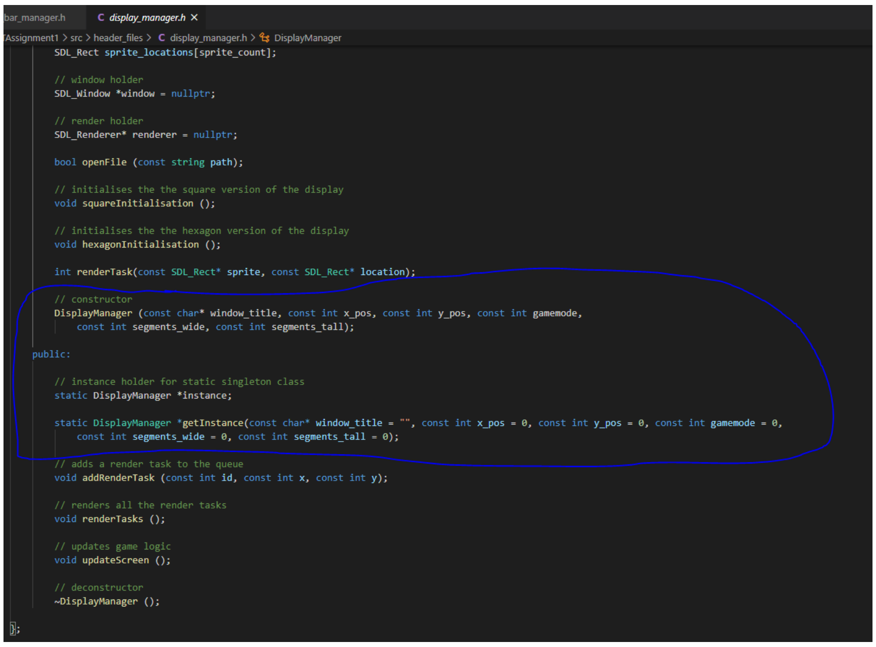


```
// calculates the correct start place of the info
int InfoBarManager::calculateStartPoint (string align, const int length) {

    if (align == "center") {

        return ((this->info_space_width/2) - ((length*16)/2));

    }

    if (align == "right") {

        return (this->info_space_width - ((length*16)/2));

    }

    return 0;

}

// queues the render tasks for the info
void InfoBarManager::renderInfo (int* info, const int x, const int y, const int end_point, const int start_point) {

    this->display_manager->addRenderTask(info[start_point], x, y);

    if (start_point < end_point) this->renderInfo(info, (x + 16), y, end_point, (start_point + 1));

}

// queues the queueing of the different infos
void InfoBarManager::renderInformation (int** infos, int* lengths, int* starts, const int info_count) {

    int y = this->vertical_start;

    for (int x = 0; x < info_count; x++) {

        y += this->padding;
        this->renderInfo(infos[x], starts[x], y, lengths[x]);

    }

}

// constructor
InfoBarManager::InfoBarManager (const int width, const int height, const int info_space_height, const int lines) {

    // get the display manager instance
    this->display_manager = display_manager->getInstance();
```

Figure 1InfoBarManager acting as a controller for Display manager

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.

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.



bar_manager.h
display_manager.h
Assignment1 > src > header_files > display_manager.h > DisplayManager

```
        SDL_Rect sprite_locations[sprite_count];

        // window holder
        SDL_Window *window = nullptr;

        // render holder
        SDL_Renderer* renderer = nullptr;

        bool openFile (const string path);

        // initialises the the square version of the display
        void squareInitialisation ();

        // initialises the the hexagon version of the display
        void hexagonInitialisation ();

        int renderTask(const SDL_Rect* sprite, const SDL_Rect* location);

        // constructor
        DisplayManager (const char* window_title, const int x_pos, const int y_pos, const int gamemode,
            const int segments_wide, const int segments_tall);

    public:

        // instance holder for static singleton class
        static DisplayManager *instance;

        static DisplayManager *getInstance(const char* window_title = "", const int x_pos = 0, const int y_pos = 0, const int gamemode = 0,
            const int segments_wide = 0, const int segments_tall = 0);

        // adds a render task to the queue
        void addRenderTask (const int id, const int x, const int y);

        // renders all the render tasks
        void renderTasks ();

        // updates game logic
        void updateScreen ();

        // deconstructor
        ~DisplayManager ();

};
```

Figure 2 singleton implementation of DisplayManager

Summary of Software Design

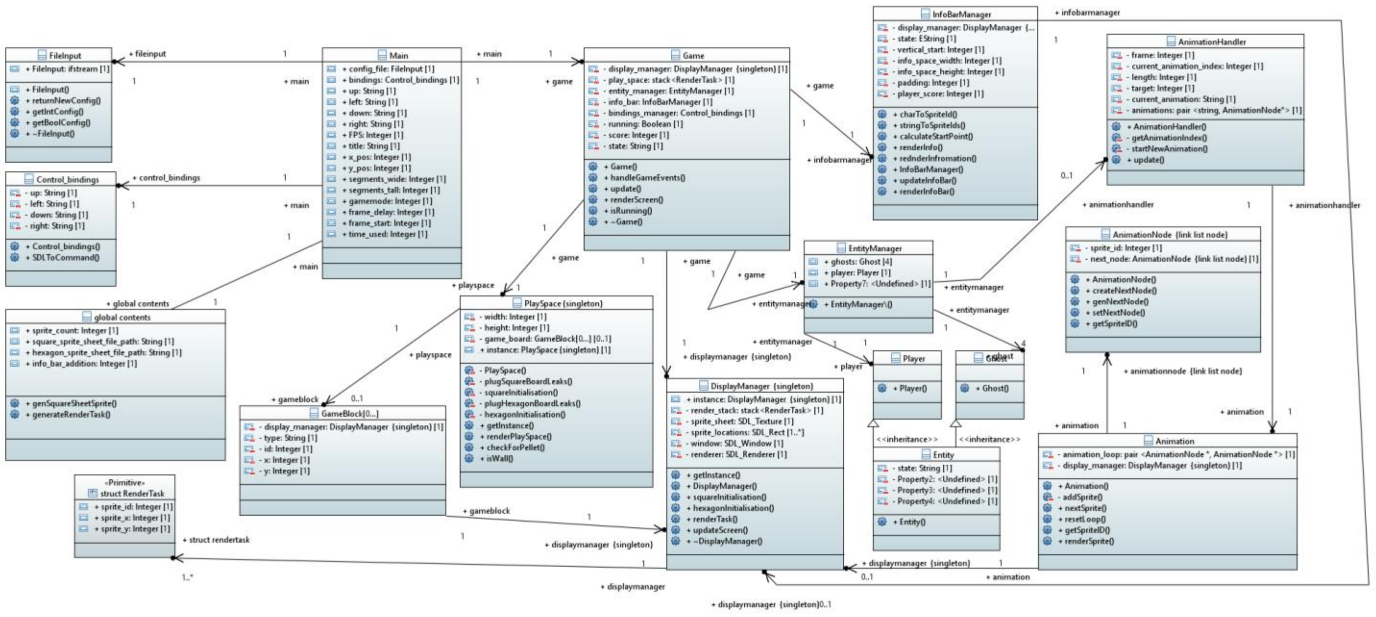
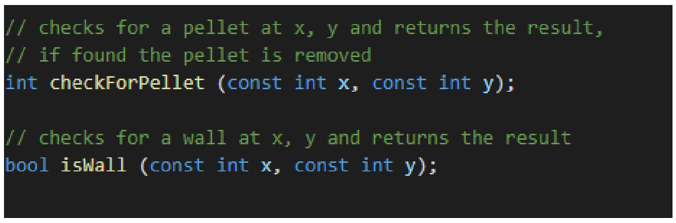


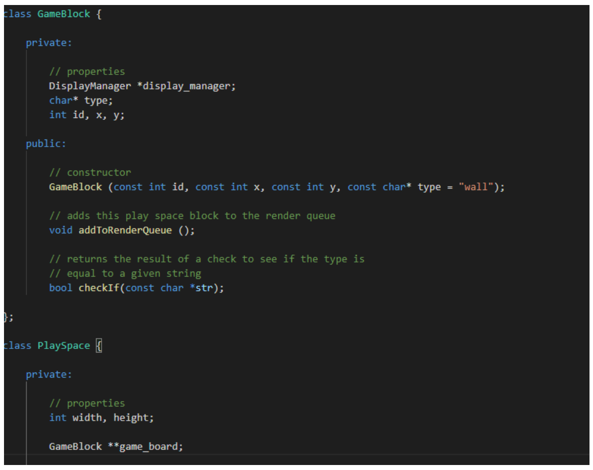
Figure 3 the full static class diagram model of the project

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.



```
// checks for a pellet at x, y and returns the result,
// if found the pellet is removed
int checkForPellet (const int x, const int y);

// checks for a wall at x, y and returns the result
bool isWall (const int x, const int y);
```



```
class GameBlock {

    private:

        // properties
        DisplayManager *display_manager;
        char* type;
        int id, x, y;

    public:

        // constructor
        GameBlock (const int id, const int x, const int y, const char* type = "wall");

        // adds this play space block to the render queue
        void addToRenderQueue ();

        // returns the result of a check to see if the type is
        // equal to a given string
        bool checkIf(const char *str);

};

class PlaySpace {

    private:

        // properties
        int width, height;

        GameBlock **game_board;
```

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.

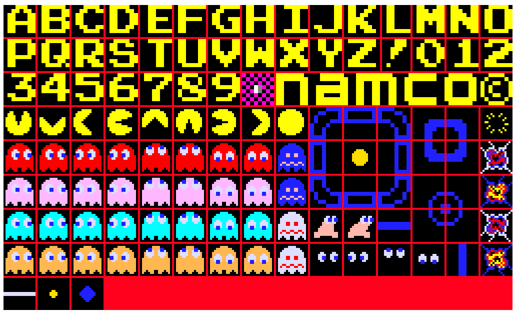
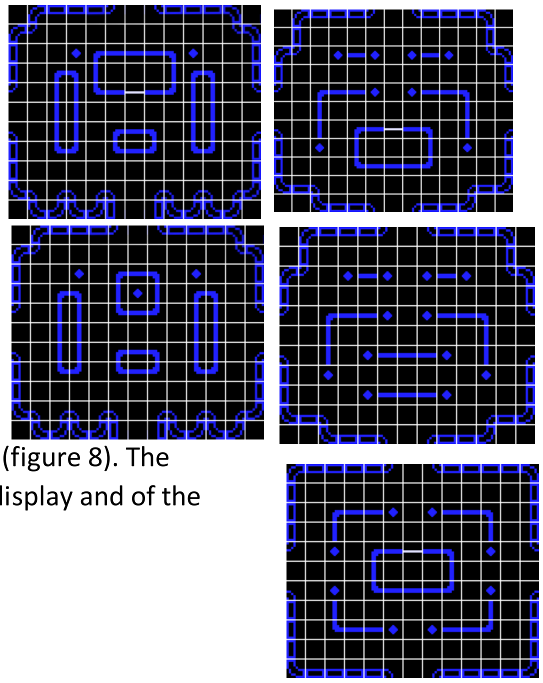
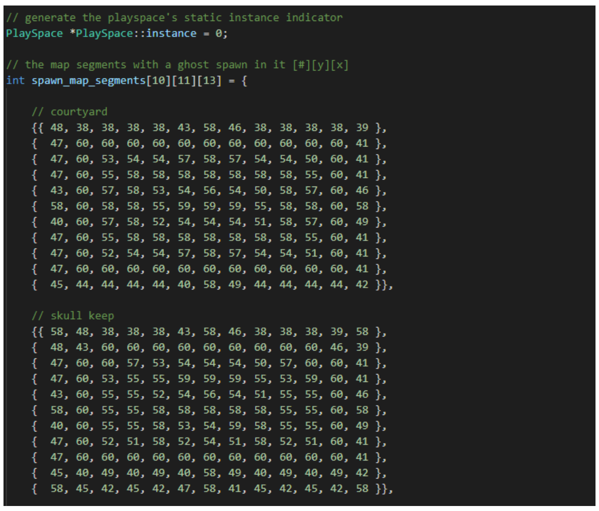


Figure 5sprite sheet



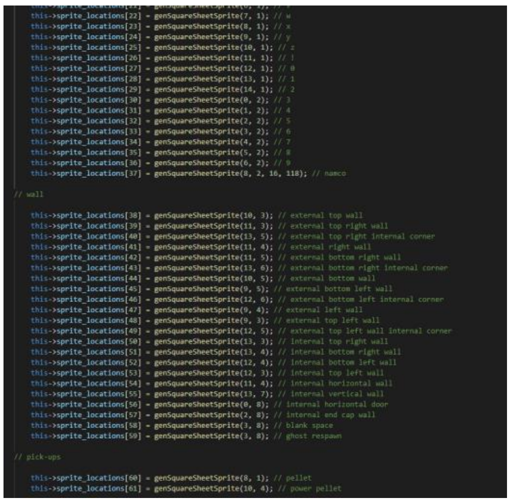
In order to create the maps shown to the left sprites are defined by location and stored in an array (figure 8). The indexes of this sprite locations can be later called to display and of the sprite, or in this case entire maps(figure 7)

Figure 6 map recreations



```
// generate the playspace's static instance indicator
PlaySpace *PlaySpace::instance = 0;

// the map segments with a ghost spawn in it [#][y][x]
int spawn_map_segments[10][11][13] = {

    // courtyard
    {{ 48, 38, 38, 38, 38, 43, 58, 46, 38, 38, 38, 38, 39 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  47, 60, 53, 54, 54, 57, 58, 57, 54, 54, 50, 60, 41 },
    {  47, 60, 55, 58, 58, 58, 58, 58, 58, 58, 55, 60, 41 },
    {  43, 60, 57, 58, 53, 54, 56, 54, 50, 58, 57, 60, 46 },
    {  58, 60, 58, 58, 55, 59, 59, 59, 55, 58, 58, 60, 58 },
    {  40, 60, 57, 58, 52, 54, 54, 54, 51, 58, 57, 60, 49 },
    {  47, 60, 55, 58, 58, 58, 58, 58, 58, 58, 55, 60, 41 },
    {  47, 60, 52, 54, 54, 57, 58, 57, 54, 54, 51, 60, 41 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  45, 44, 44, 44, 44, 40, 58, 49, 44, 44, 44, 44, 42 }},

    // skull keep
    {{ 58, 48, 38, 38, 38, 43, 58, 46, 38, 38, 38, 39, 58 },
    {  48, 43, 60, 60, 60, 60, 60, 60, 60, 60, 60, 46, 39 },
    {  47, 60, 60, 57, 53, 54, 54, 54, 50, 57, 60, 60, 41 },
    {  47, 60, 53, 55, 55, 59, 59, 59, 55, 53, 59, 60, 41 },
    {  43, 60, 55, 55, 52, 54, 56, 54, 51, 55, 55, 60, 46 },
    {  58, 60, 55, 55, 58, 58, 58, 58, 58, 55, 55, 60, 58 },
    {  40, 60, 55, 55, 58, 53, 54, 59, 58, 55, 55, 60, 49 },
    {  47, 60, 52, 51, 58, 52, 54, 51, 58, 52, 51, 60, 41 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  45, 40, 49, 40, 49, 40, 58, 49, 40, 49, 40, 49, 42 },
    {  58, 45, 42, 45, 42, 47, 58, 41, 45, 42, 45, 42, 58 }},
```

Figure 7 maps stored as a 2d array of ints



```
    this->sprite_locations[22] = genSquareSheetSprite(7, 1); // w
    this->sprite_locations[23] = genSquareSheetSprite(8, 1); // x
    this->sprite_locations[24] = genSquareSheetSprite(9, 1); // y
    this->sprite_locations[25] = genSquareSheetSprite(10, 1); // z
    this->sprite_locations[26] = genSquareSheetSprite(11, 1); // !
    this->sprite_locations[27] = genSquareSheetSprite(12, 1); // 0
    this->sprite_locations[28] = genSquareSheetSprite(13, 1); // 1
    this->sprite_locations[29] = genSquareSheetSprite(14, 1); // 2
    this->sprite_locations[30] = genSquareSheetSprite(0, 2); // 3
    this->sprite_locations[31] = genSquareSheetSprite(1, 2); // 4
    this->sprite_locations[32] = genSquareSheetSprite(2, 2); // 5
    this->sprite_locations[33] = genSquareSheetSprite(3, 2); // 6
    this->sprite_locations[34] = genSquareSheetSprite(4, 2); // 7
    this->sprite_locations[35] = genSquareSheetSprite(5, 2); // 8
    this->sprite_locations[36] = genSquareSheetSprite(6, 2); // 9
    this->sprite_locations[37] = genSquareSheetSprite(8, 2, 16, 118); // namco

// wall

    this->sprite_locations[38] = genSquareSheetSprite(10, 3); // external top wall
    this->sprite_locations[39] = genSquareSheetSprite(11, 3); // external top right wall
    this->sprite_locations[40] = genSquareSheetSprite(13, 5); // external top right internal corner
    this->sprite_locations[41] = genSquareSheetSprite(11, 4); // external right wall
    this->sprite_locations[42] = genSquareSheetSprite(11, 5); // external bottom right wall
    this->sprite_locations[43] = genSquareSheetSprite(13, 6); // external bottom right internal corner
    this->sprite_locations[44] = genSquareSheetSprite(10, 5); // external bottom wall
    this->sprite_locations[45] = genSquareSheetSprite(9, 5); // external bottom left wall
    this->sprite_locations[46] = genSquareSheetSprite(12, 6); // external bottom left internal corner
    this->sprite_locations[47] = genSquareSheetSprite(9, 4); // external left wall
    this->sprite_locations[48] = genSquareSheetSprite(9, 3); // external top left wall
    this->sprite_locations[49] = genSquareSheetSprite(12, 5); // external top left wall internal corner
    this->sprite_locations[50] = genSquareSheetSprite(13, 3); // internal top right wall
    this->sprite_locations[51] = genSquareSheetSprite(13, 4); // internal bottom right wall
    this->sprite_locations[52] = genSquareSheetSprite(12, 4); // internal bottom left wall
    this->sprite_locations[53] = genSquareSheetSprite(12, 3); // internal top left wall
    this->sprite_locations[54] = genSquareSheetSprite(11, 4); // internal horizontal wall
    this->sprite_locations[55] = genSquareSheetSprite(13, 7); // internal vertical wall
    this->sprite_locations[56] = genSquareSheetSprite(0, 8); // internal horizontal door
    this->sprite_locations[57] = genSquareSheetSprite(2, 8); // internal end cap wall
    this->sprite_locations[58] = genSquareSheetSprite(3, 8); // blank space
    this->sprite_locations[59] = genSquareSheetSprite(3, 8); // ghost respawn

// pick-ups

    this->sprite_locations[60] = genSquareSheetSprite(8, 1); // pellet
    this->sprite_locations[61] = genSquareSheetSprite(10, 4); // power pellet
```

Figure 8 stored sprites

Dynamic Model

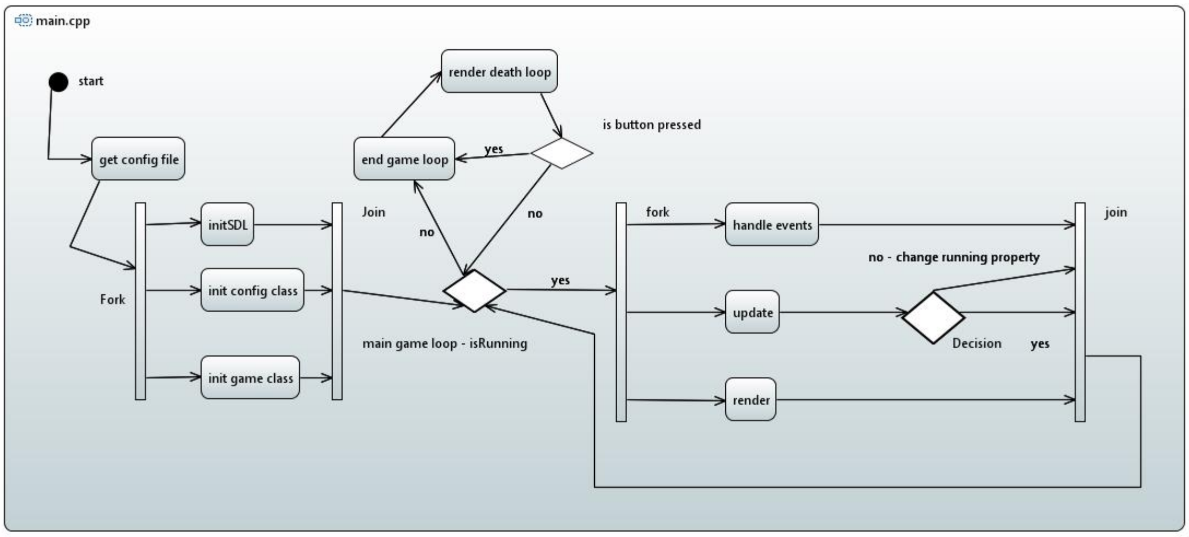


Figure 9 dynamic model of the main implementation

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

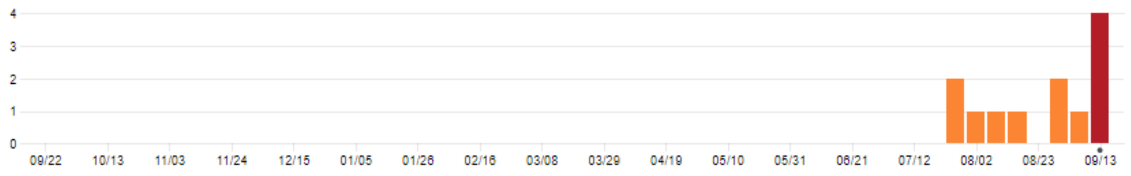


Figure 10 commits by week

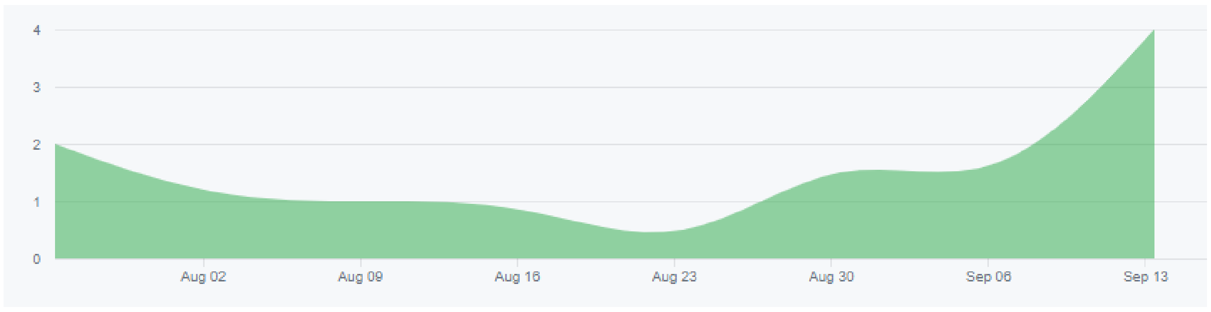


Figure 11 commits over time

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.