PACMAN

Senior Design Project

Kristin Miyamoto

1. Introduction and Background

The goal of this project was to recreate a version of pacman using C++ with the SFML library. The original Pacman game’s core mechanics are for the player to navigate pacman through a maze to collect all the dots while avoiding the four ghosts that are pursuing him. The concept of the game is relatively simple but the complexity comes from the ghosts’ design. In the original game each different colored ghost’s movement pattern would be different depending on its color. All the ghosts are on a set fixed timer and switch between a chase and a scatter mode [5]. In scatter mode, all the ghosts will disperse and rotate around a specific corner of the map. In chase mode, the ghosts will all target pacman, but depending on the ghosts’ colors they will go about pursuing pacman differently. The red ghost will directly target pacman, the pink ghost will target four positions ahead of pacman, the blue ghost will target the position location calculated by taking two positions in front of pacman and drawing a vector from pacman to the red ghost’s position and doubling the length of the vector, and the orange ghost’s target position is identical to the red ghost when pacman is 8 or more positions away and will move away from pacman otherwise[5]. The ghosts’ movements are controlled by a simplistic computer AI; however, this still does not mean that it won’t be challenging. The original board size is a 28 by 36 tile grid with 24 dots and 4 energizers [5]. When pacman eats an energizer it triggers the ghosts to go into frightened mode where the movement is random, and if pacman intercepts one or more of the ghosts they will target the ghost home or the starting position of the ghosts.

For this project the goal was to best recreate some of the original game’s core mechanics, such as having the player be able to control and move pacman throughout the maze, to collect the dots and energizers, and to have the ghosts pursue and do rudimentary elements of their original mechanics. The finished project remained mostly true to the original game with the objective for the player to navigate the character through a series of 5 different mazes while avoiding the four different colored slimes. When the player’s character would eat a power up or potion, the slimes would switch to frightened mode and change color to a dark blue. In this phase the slimes’ movements will be random and if the main character or pacman intercepts one of the slimes in this phase, the slimes will be forced to attempt to go back to their original starting position. The recreation does not allow the player to progress to the next floor or map unless the player collects the map. The map will only appear on the board when the minimum threshold score for the corresponding floor is reached by collecting the pellets and potions which are worth 10 and 50 points respectively.

A story line was added to the beginning of the game to make it more of a customized and immersive project. The game also allows the player to be able to select which of the two character models to play as, and the character selection is carried over to the story sequence and the sprites that are used to explore the maze map. The game tells the story of how the player’s character, who is an unappreciated artist, is in desperate need of money, and the storekeeper offers the player’s character a business opportunity to explore the dungeon and draw a map of each floor to sell to adventurers. In return for the player character’s cooperation, they will receive a share in the profit. The character agrees, which then starts the pacman-like game.

The reset of the game is very similar to the original Pacman game. The player controls the character that they selected from the character selection menu and navigates them to collect the dots throughout the maze. The dots represent locations on the map that the character draws, and when collected the player’s score increases. When the player reaches a certain amount of points, a sprite of a map will appear in the maze and when the player collects the map, it will advance them to the next floor. Throughout the maze floors there are potion sprites that serve as the energizers to repel the slime monster back to their starting locations and give the player’s character temporary invulnerability. The game consists of a total of 5 floors with 5 unique mazes for the player to explore. If the player dies, the game gives the option of either exiting the game or retrying the floor. If the player chooses to retry the floor map, the player’s score from the last complete floor is retained and is the new starting score. Another added feature is that on the start menu the player can choose to start the game, read the instructions, or exit the game. The project does deviate from the original pacman game by adding a story element into the game, a character selection menu, changing the ghosts movement timer, and other things that do not take away from the original game mechanics, and one might argue that they give the game more personality and complexity.

The main purpose of this project was to gain a better understanding of C++ and to learn how to use the SFML library. I had never used the SFML library before and I have never designed a game before, so this project was very daunting to me. Throughout this project new concepts were utilized and a lot of research into both the original pacman game and how to use SFML was done prior to even starting to code. I wanted to challenge myself and to try all the different aspects of game design from software design to graphics design. For this project since I took an extension on it, it allowed me to try my hand at designing and incorporating my own graphics into the project. I researched how to create a sprite sheet and learned how to create one that would work with the SFML library to give the player’s character and slimes the illusion of being animated. It seems like a very simplistic thing to do, but it was actually a little bit of a challenge to get the sizes and timing right. It was understood from the very beginning that this was going to be one of the largest projects I’ve done so far, so the AGILE methodology was used because time management was a concern. The AGILE methodology prioritizes getting a functional software program in a timely manner over maintaining documentation. This method however helped me push through getting the code done in a timely manner and set small goals for myself to complete each week. The project also introduced new C++ concepts and helped strengthen my understanding of the ones I already was familiar with. There was a definite learning curve to the SFML, but a lot of videos and the tutorials on the official SFML website helped ease the process. Music and sound effects were also integrated on a small scale into the game. The background music used in the game is from the Square Enix title Nier Reincarnation but was remade using vsynth software. The main purpose of adding the background music and sound effect was to test out the audio features in the SFML library.

1. Installation

In order to successfully run this program the user must have at least Visual Studios 2017 or earlier and the version must match the version of SFML that you are installing. The SFML library can be downloaded from their website: <https://www.sfml-dev.org/download/sfml/2.6.0/>. For reference purposes, this project was made and run using SFML version 2.5.1 and used Visual Studio 2022 with the SFML Visual C++ 17 - 64 bit compatibility. It is recommended to follow the installation instructions outlined on the official SFML website: <https://www.sfml-dev.org/tutorials/2.6/start-vc.php>, but a brief installation summary of the configuration of what was done for this project will be given to ensure that the environments are the same. In ‘Project Properties’ under the ‘Linker’ tab and in ‘Input’ add the following libraries to additional dependencies: sfml-system-s.lib, sfml-audio-s.lib, sfml-window-s.lib, sfml-graphics-s.lib. Then in debug and release configuration under the ‘C/C++’ tab in ‘Preprocessor’ add SFML\_STATIC to the ‘Preprocessor Definitions’. Compile and run the empty main.cpp file in visual studio so that a debug folder will appear where the solution and files are saved onto your computer, assuming you selected them to be saved to the same location. Navigate to the programs Debug folder and copy the appropriate .dll files that match what you added to the additional dependencies. Be careful when you do this not to mix up files because this will cause an error when trying to link the SFML library. 

In addition under ‘Configuration Properties’ under ‘Debugging’ in ‘Environment’ put in the path of where the SFML bin folder is stored on your computer. Use the code provided by the official SFML site to verify that you set up the environment correctly. If done correctly, a window with a green circle should appear. Once the environment is set up and the code has been downloaded and uploaded into Visual Studios, the program should be compiled and started without debugging.

1. User Manual

The game will start and you can move the up and down arrows to navigate through the menu screen. To select an option you press the enter key and the selected option is colored in red. If you select the instructions option on the start menu, the game instructions will appear on the screen, and in order to go back to the main menu you press enter once. On the character menu you move the right and left arrow keys and press enter to select. Once all the start options are selected, an image of a store background is displayed and you press enter go through the story dialogue. When the story dialogue is finished, a new window will open and the game will start. The player controls the character using the four arrow keys and can move the character in the corresponding direction of the arrow keys. The game consists of 5 levels or floors, and upon ascending each floor the required amount of points to make the map sprite appear in the maze increases. Each pellet or dot is worth 10 points and each energizer or potion bottle is worth 50 points. The only way to move to the next floor is to get the map icon. If the character is intercepted by a slime, the game is over and the game then prompts the user whether they would like to restart the floor or exit the game. The user uses the arrow keys to select the option and presses enter to make their selection. The score from the last previous level if applicable is carried over. While in the maze if the character intercepts with a potion bottle, the slimes will change to dark blue, indicating that they are in frightened mode and that the character is invulnerable to the slimes. If at any point the user wishes to pause the game while in the maze, the ‘ESC’ key will freeze character’s and the slimes’ movement. The game can be resumed again upon hitting the ‘ESC’ key again. The warp tunnels can be used to expedite moving from one side of the maze to another and the ghosts can also follow. If the user wishes to close the game in the middle of the game, the user can click on the ‘x’ button on the right side to close the window.

1. Design Overview

For this project the agile software design method was used. The priority of this project was to get a functional prototype or program that met the requirements. To recap, the goals of this project were to 1) program pacman’s movement 2) program rudimentary ghost movement 3) collision checking 4) create a GUI that takes input from the user 5) incorporate the basic game mechanics of pacman, for example the pellets. This project achieved all of those things and extended the scope of this assignment. Custom sprite sheets were made using ClipSudio Paint, a drawing software program, that created .png files of the images. A main menu, pause functionality, character selection menu, adding multiple floors or maze maps, having the game be able to restart the failed floor, and the previous cleared floor’s score carried over were all added into the game. In addition the original mechanics and rules were adjusted to give the project more complexity. In order to progress to the next floor, a certain score must be achieved in order for the map sprite to appear in the maze map. The map is temporarily hidden by making the map a wall and pacman is unable to interact with the map until the floor’s threshold score is reached. The idea is that the threshold score to progress each floor increases, so it requires the player to clear more of the map than the previous floor to increase the difficulty of the game. One strategy is to try to collect more pellets and potions on the two earlier floors than necessary, because the later floors are a bit more difficult to navigate and require a higher total score for the map to appear. The player can only progress to the next floor when the map is collected.

The first objective of the project was to do thorough research about SFML. Having a strong enough understanding of the most frequently used classes and features in SFML was critical in order to build a solid foundation for the project and to understand how to plan its progression. From the research that was conducted prior to the actual coding, it was evident that debugging in SFML would be very different from the typical C++ debugging, since the code would be constantly expecting user input. The easiest method to debug the SFML code was to test and debug the C++ code by putting little indicators that would print out into the console to alert that something was wrong. Typically in debugging in C++ you can step through the code to find any errors and go down line by line to check the logic. Using the SFML debugger, this is not really possible since it will just alert that the program cannot be built. This was all taken into consideration when deciding the classes and what flow the classes needed to be made in to allow ease in debugging. 

The next priority was to get the SFML window to work with Visual Studios 2022. This would verify that the SFML environment was set up correctly. Once the window was created the next step was creating a rudimentary map using squares for the walls and a circle for pacman. The first image of the prototype can be seen in figure 2. SFML has classes to create shapes respectively named Circle and Rectangle that create those shapes in the GUI window. The next step was to take input from the keyboard and have pacman be able to move accordingly. This was done using the SFML event listeners that incremented or decremented pacman’s x and y coordinates that would allow pacman to occupy the next space in the direction of the arrow keys. Pacman is limited to only four movement options up, down, left, and right, which made the movement algorithm fairly easy to figure out since this is a 2d game locked into the 2d cartesian coordinate plane. Once pacman was able to be controlled by the user with the arrow keys, it was time to add collision checking to make sure that pacman didn’t go out of bounds. Since pacman is limited to four directions the collision checking only needs to check four sides. Basically the collision checking class just checks to see if the intended space that pacman wants to occupy is a wall and would return true if the next space is a wall. If the collision checking returns true, then pacman would be unable to proceed moving there, and the x and y coordinates of pacman’s position would not update. With pacman being able to move and collision checking complete, it was a good time to try to incorporate the ghosts, which would later be changed to the slime class. The ghosts were the hardest part to program and took up the majority of the time. The base of the ghost class is almost identical to the pacman class, which is why pacman was prioritized to be created first. As stated before, each of the different ghosts has a unique movement pattern in chase mode. The red ghost will target pacman’s position directly by making pacman’s current position its target tile or intended position. The orange ghost will target pacman and then, once it gets close to Pacman, it will switch to scatter mode. The pink ghost will target 3 spaces or tiles in front of pacman. The blue ghost takes into consideration both pacman’s position and the red ghost’s position. The blue ghost’s position is calculated by taking two spaces in front of pacman, drawing a vector from the red ghost, and doubling the length of that vector. This part was very challenging to program and there were multiple attempts to try to remain as close to the original game mechanics as possible. 

Unfortunately, for the finished project the original pacman mechanics for the ghosts was slightly changed. The red, orange, and pink ghost mechanics remained the same as the original, but the blue ghost mechanics were changed slightly. Once the ghost classes were completed for the first map, the next step was to have the player be able to lose or win the game. A function to check if pacman collided with the ghosts or slimes was added and an attempt was made to try to account for the sprite size negative space, which will be discussed more in detail in the pacman class. In the main game function, if pacman collides with the ghosts, the function will return true and a game over screen will appear. In order to win the game a certain score accumulated from collecting the pellets and energizers or potions needs to be reached. The pellets or dots are each worth 10 points and the potions or energizers are worth 50 points. The DrawMap class is edited to include, not just the walls but also the pellets and energizers, which are represented by circles of different sizes. With the player now being able to control pacman, the ghosts being able to do rudimentary movement, and the game being able to be won or lost, the base game is completed and additional features of the game are added. The frightened feature, which is when the ghosts turn blue from when pacman eats the energizers or potions and since the ghosts are in frightened mode they target going back to their start positions, is added into the ghost or slime class. Menus like the main menu and character selection menu, which the user can use to select different options, are incorporated to make the game more user customizable and friendly. The pause functionality was also integrated into the game to freeze and unfreeze all the characters’ movement. The addition of different maps was added into the game and also the mechanic that when a certain score is reached one of the wall tiles will turn into a map sprite, so that when the character collects the map they can progress to the next level, was integrated.

The final thing that arguably took the longest was making my own graphics for the game. I designed and created my own sprite sheets for the boy and girl characters that the user can select from at the start of the game. I also redesigned the ghosts so that they weren’t just rectangles and made them into slime sprites. I learned how to design and create the sprite sheets so that they are compatible with the SFML classes, texture and sprites. How the animation works with the sprite sheet is that the sprite class in the SFML library assumes that the sprites are always rectangles. This allows the length and width of the part of the image you want to grab and the position of that rectangle size on the overall picture, which enables you to loop through the sprite sheet. When creating the sprite sheets, the images were changed so that each character image would fit inside a 20 pixel by 20 pixel rectangle. It would then cycle through the 6 images and, depending on the direction that the character is going, would go up or down a row. The sprite sheets I made for the slimes, the male, and female characters are shown in figure 3,4, and 5. 

1. Data Structures Used
   1. Inheritance

Inheritance was used to make the slime classes. An overall Slimes class that held all the functions that the slimes share together was created as the base class. Individual slime classes based on their colors were created. The classes Red, Blue, Purple, and Green are derived classes from the base Slime class. This was done, because even though the all four slimes do basically the same thing, when calculating the new target tile for the scatter mode and chase mode, the calculations change based on the slimes colors. This is to give more variation to the enemies movement and make them less predictable.



* 1. Switch Statements

Switch statements were heavily used especially with the SFML listener as shown in figure 6, because switch statements are only evaluated once and the value of the expression is compared with the values of each case until a match is found [6]. Switch statements make code a lot more easy to understand and read rather than a long if/else statement which is why this was favored over if/else statements. One problem that arose from using switch statements was that it was very easy to accidentally leave out a break keyword [6]. When this would happen, the switch statement would evaluate where a match is found and then keep cascading down until a break or it exits the switch statement. This would not cause an error when the program is compiled which made this a difficult logical error to find when debugging. Since the game runs with the SFML library, debugging the code was very difficult. SFML does not provide much debugging support other than indicating that the code cannot be built. Switch statements were mostly used for the SFML event listener, because the window would loop through in a while loop that kept going until the window was closed and check the switch statements to see if an event occurred, such as when a specific keyboard key was pressed.

* 1. Ternary Operator

Ternary operators function is similar to if/else statements and were used in the pause functionality of the game [3]. It was used to toggle the game's character movement when the ESC button on the keyboard is pressed. This was done for learning purposes and to make things easier to understand rather than writing out an if/else statement to just switch the pause functionality to true or false.

* 1. Arrays

The array standard template class was used to hold the elements of the map [2]. Arrays are fixed sized containers that hold a specific number of elements in a strict sequence [2]. For this project arrays were much more desirable than vectors, because vectors are mutable and the likelihood of something accidentally being added into it is much greater than as opposed to arrays which are immutable. The map array is an array of enumeration elements that represent what the keyboard characters should be converted into for when it is displayed in the GUI.



* 1. Enums

Enumeration variables were used to organize and sort the map elements [1]. The map sketch was sorted into the enumeration values and then stored in an array of arrays of enums. This was to make comparison and looping through all the elements of the array that holds the map elements easier and more understandable to follow.



* 1. Structs

A struct was created to hold the position of the different elements like pacman’s or the main character’s position, the position of the slimes, and the position of the door. The structure takes in two integers representing the x and y coordinates, since the SFML window is based on the cartesian coordinates. This structure was made to help arrange the different items that are to be displayed in the gui window. Within the position structure the == operated was overloaded so a comparison between two position objects could be computed if they are equivalent to each other or not.

* 1. Constexpr

Constexpr is a keyword that is similar to const in that the value of the variable cannot be modified within the program [4]. When a variable is declared as this it indicates that when possible the should be computed at compile time rather than runtime [4]. This is to help the program run faster and use less memory. Since this project is a game it has a slow runtime even though C++ is considered one of the faster programming languages. By having some of the constant variables be declared as constexpr, this allows those variables to be initialized faster and help improve the overall program’s speed even if it’s just by a little. Some of the variables that were declared as constexpr are the size of the cells or the squares of the GUI window. The cell size was declared as constexpr because the cell size is repeatedly called to size the main character, the slimes, and the elements of the map, so it was an efficient thing to do. Declaring this variable as constexpr enables it to be already declared before the code is runned.

1. Description of Classes
   1. Pacman

This class is responsible for all the things related to pacman or the main character. This class controls the main character's movement depending on the player’s input. When the player presses one of the directional arrow keys the main character (pacman) will move in the corresponding direction. How movement works in the SFML GUI window is that the window is broken down into a 2D cartesian plane where the upper most left corner of the window is the (0,0) position and the bottom most right is the (max length, max width) of the window, as pictured in figure 9. Knowing this simplifies pacman’s movement algorithm. In order to move pacman in the corresponding input direction, pacman’s x and y position will need to be incremented or de-incremented depending on where the player wants to move pacman. This class also utilizes the SFML library to draw the pacman or main character sprites. For this project the user has a choice on the character selection menu to pick between two characters the “boy” or “girl” and within the draw function the sprite associated with the user’s selection is passed to the program’s main function to be displayed in the GUI window. To give a little more background on what sprites and textures are, a texture is an image with the very purpose of being mapped to a 2d entity and a sprite is a rectangle entity that has a texture overlay on it. I was overly ambitious with this project and took the time to design and create my own sprites for this project. I wanted to take the learning opportunity to understand how sprite sheets work and how the program loops through the different images to give the illusion of animation. Within SFML there is a class that encapsulates textures. The only role of this class is to load and map the images or filenames that are to be mapped on the graphical entity or sprites. On the SFML official website there is a tutorial on how to load textures, which was heavily referenced and will be linked in the reference section accordingly. In the SFML texture tutorial, it goes into other methods of how to load textures but in this project the load texture from a file was exclusively used. The images were made clipstudio paint, a drawing software, and were saved as .png files so the backgrounds would be transparent. It is worth noting that the actual size of the sprite is larger than the actual displayed image in the GUI because the sprite is really a rectangle shape with an image mapped on top of it, so some of the actual size of the sprite is invisible negative space. This can cause when pacman collides with the slimes some confusion, since the size of the sprite is misleading. In the function pacman\_collide(), it checks that is in the slimes class it checks to see if the slimes intercept with the pacman sprite and it must account for that the actual sprite size is not the same size as the texture so pacman’s actual position is given a tolerance of half the size of the pacman’s cell size which is 20 pixels. 

The pacman class also acts as an intermediary between the main program and the collision class since the collision class only needs to interact with the pacman class and slimes classes. The main program needs to know the score from pacman eating the pellets and potions and the pacman class passes the score information between the two classes. It also lets the collision class know to get the next floor and reset the score. When the pacman class eats a potion or energizer it keeps track of how long the invulnerability to being killed by the slimes lasts. The collision class checks to see if pacman collided with a position bottle and then sends a true or false response to the pacman class signaling if pacman is in an energized state or not.

* 1. Blue, Red, Green, Purple

The blue, red, purple, and green classes represent the different colors of slimes or the enemies that the main character has to try to avoid. These classes are derived from the base class Slimes. The purpose of these classes are to compute the target tile associated with each of the slimes’ different colors. It was briefly touched upon already that each of the different colors of ghosts or slimes is given a different method to compute the target tile or the position. Because of the target tile variation, the function used to compute the target tile was moved to its own classes to make it easier to read and understand the algorithm behind each of the different slimes. Once the target tile is computed the slimes will try to compute the shortest route, note it's not the best most efficient route, and attempt to move there. The original map design of pacman leaves little opportunity for the ghosts to get stuck; however, this project expanded upon the classic design and added more maps to increase the overall complexity of the game. Which is why in some instances causes the slimes or ghosts movement to be grid locked for one phase cycle since the slimes are not able to move backwards.

* 1. Slimes

The slimes class is the base class of the classes red, purple, blue, and green which are aptly named after the colors of the slimes. The main purpose of this class is to control the movement and reset and update the values that are used to control the slime's behavior and animations. This class is very similar to the pacman class that was already discussed in detail early on.

* 1. Manager (slime manager)

The purpose of the slime manager class is to provide uniform control over the slimes. The slimes should all be in the same phase and this class ensures that all the slimes' movement modes, scatter or chase, are consistent with each other. This class is also responsible for keeping track of the movement mode duration timer, which is the duration of how long each of the phases last for. It should be noted that the scatter and chase phases are independent of pacman’s movement but the frightened phase is dependent on pacman’s movement because it is linked to the energizer timer located in the pacman class.



* 1. Selection

The character select class purpose is to create a menu for the character select menu. Figure 10 shows the character selection menu created in the Selection Class

* 1. DrawMap

The purpose of the DrawMap class is to convert the text map created in the program’s main into the defined variables stored in an enum. Certain elements like the slimes, the main character, door, and map key position are stored or passed by reference so that those values can be used later on. Within this class it also is responsible for drawing the map by making the walls of the rectangles, the pellets into circles, and converting the PowerUps and map key into sprites using the SFML classes. This class also contains the function to convert one of the walls into the map key sprite that the character can get and move to the next floor.

* 1. Menu

This class is responsible for creating the menu of the startup screen. It heavily uses the SFML classes like font and texture to create the startup menu. This gives the user the selection to either play the game, read the instruction, or exit the game. This class changes the window according to the user’s input and gives what was selected to the program’s main function.

* 1. collision

This class checks to see if something is a wall and returns a boolean. If the next intended move is a wall it will return true. This class is also responsible for checking if an energizer or potion was eaten by pacman as well as seeing if the player can advance to the next floor. Within this class it also keeps track of the total score the player has earned.

* 1. game over selection

The purpose of this class is to hold the values associated with the game over the menu when the player dies.

* 1. pause

The purpose of this class is to control the games pausing functionality. Within this class it toggles the pause functionality by using a ternary operator. This lets the main function have no whether or not to freeze the movement of the characters. It should be noted that the animation is not freezed just the characters the slimes and main character’s movement



* 1. Intro

The purpose of the Intro class is to hold the opening dialogue text and picture name (string) associated with the block of text. It then draws the picture to the window in the program's main. There is an if else statement that changes the picture of the character selected in the character selection to the appropriate picture.

* 1. stdafx

This header holds any libraries and classes that are used in multiple instances of the program. It also holds any global variables and functions that are used.

1. Summary and Conclusion

In summation, the project was a fun and challenging learning experience. I was able to recreate a version of pacman and incorporate RPG (role playing game) elements into it by adding custom graphics and a creative storyline. The core mechanics are still true to the original pacman game with minor changes. The program is over 4,000 lines of code and was finished within 9 months. During this time I really struggled with time management, because I was working and taking OPL among other classes. I spent about two months designing, drawing, and integrating the graphics for the game. I’m not at all an artist, but it was a hobby that I was really glad to be able to showcase in this project. The storyline of the game added a layer of immersion to the game and let me feel more invested in making this project my own. Throughout the project, I was forced to step out of my comfort zone and teach myself new things. This was my first time using the SFML library so it required me to teach myself a lot about it by watching youtube tutorials and reading the tutorials on the SFML website. The background music in the game was made using vsynth software, which is a voice manipulation program typically used to make covers of songs. This is another hobby of mine that I was happy to include and it is a cover of a song from the Square Enix game Nier Reincarnation. I have never done a large project of this magnitude before and was very overwhelmed with how much preplanning was required before actually starting the code. I found that the AGILE method worked best for me even though it proved to be inefficient in some cases.

The game, despite being working and playable, has several bugs that have yet to be resolved. If given more time I would like to fix them because it would enhance the overall game. One of the glaring bugs is that at certain positions, mostly in the later maze maps, the ghosts will get stuck in between two walls and appear to briefly glitch out. This is most likely due to how the target tile that was discussed in the earlier sections is calculated. Because the ghosts are programmed to not to move backwards, it can only move in three cardinal directions and will always try to move the shortest distance. Where this becomes problematic is when the ghosts have to move backwards in order not to be trapped. Another bug in the game is that because the file is so large there is a noticeable delay when the game initially starts. This goes away after a few seconds and the player starts to move the character. There might be no way to fix this since it seems to be resolved when the program is runned on a stronger device than my laptop. Another issue is that the mode timer for the ghosts was changed from original, because it was easier to manage a more fixed timer rather than one that intrinsically changes. In addition the character, because it is sized slightly larger than the tile of the game grid, will consume more than one dot on certain maps. A remedy to fix this would be to shrink the sprite size of the character but the animation loop would have to be changed and the .png image would have to be resized. Another bug is that the frightened timer is linked to the movement of the player character so if the player does not move the frightened timer would not go down. This would require a change in how the frightened timer is designed in the code and was not something that was apparent right away. By the time I noticed it to be a real issue, too much was already done and it was not worth rewriting two classes to correct it but it should be noted. If given more time I would change it so that it is purely dependent on the ghost movement rather than being dependent on the character. 



There are some things that I wish that I did differently. I am not the best with time management and this project really forced me to be on top of myself and make sure I’m not having tunnel vision on one section that I’m stuck with. There were days and sometimes weeks were I would be hyper focused on fixing this one little trivial part instead of trying to progress the project as a whole. Another thing that I wish I did differently was that I should have started with a stronger base, which was the result of creating an inflexible plan from the beginning. The original plan I thought of required me to be able to finish things in sequential order which was not the best idea given the size of the project. What I ended up doing was creating separate classes that I was able to test in small sections rather than testing the whole project which made debugging it nearly impossible.



Some things that I wish to add would be to add a background to the maze maps to make the game more immersive and complete. To redesign the Game Over and Win screens since they are currently very plain. I would like to add an option to control the volume of the background music. I would also like to add more music features and sound effects to the game to make it feel more complete and to explore how to control the sound using the SFML libraries. I would also like to add another layer of complexity when the player ascends to the next floor like increasing the ghosts speed or shortening the character invulnerability time.



Overall this project was a great experience and furthered my knowledge with different C++ concepts. The project contained some new concepts that I haven’t really tried before and it also served as a catalyst for me to learn how to utilize the SFML library. There are some things I wish I could fix or change but this project was a learning experiment and I plan to continue improving it. It allowed me to gain hands-on experience with all the different facets of software design and game development.

Bibliography

[1]“What is C++ Enum, And How to Use Enums in C++ [2022 Edition],” *Simplilearn.com*. <https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-enum>

[2]*Cplusplus.com*, 2023. https://cplusplus.com/reference/array/array/ (accessed Oct. 01, 2023).

[3]“C++ Short Hand If Else (Ternary Operator),” *www.w3schools.com*. <https://www.w3schools.com/cpp/cpp_conditions_shorthand.asp>

[4]“constexpr specifier (since C++11) - cppreference.com,” *en.cppreference.com*. <https://en.cppreference.com/w/cpp/language/constexpr>

[5]Mike, “SFML C++ Tutorial–Spritesheets and Animation,” *GameFromScratch.com*, Oct. 27, 2015. https://gamefromscratch.com/sfml-c-tutorial-spritesheets-and-animation/ (accessed Oct. 01, 2023).

[6]“C++ Switch,” *www.w3schools.com*. <https://www.w3schools.com/cpp/cpp_switch.asp>

All Reference Used in the Code are cited within the code