Title of Project	Mini Golf Dungeon
Student Name & Number:	Konrad Winkel K00232458
Student Name & Number:	James O'Neill K00226160
Student Name & Number:	Rejean Lagasse K00244594
Student Name & Number:	Zachary Ayotte K00244551
Date	04- February - 2020
Word Count (11,362) 18.961 Words	

BSc in Computing (Games Design and Development)

Limerick Institute of Technology (Tipperary)

Certificate of authorship

We hereby certify that this material, which we now submit for assessment on the programme of study leading to the award of [degree title] is entirely our own work, and has not been taken for the work of others save and to the extent that such work has been cited and acknowledged within the text.

Name of Candidate:	**Konrad Winkel**
Signature of Candidate:	***Konrad Winkel***

Name of Candidate:	**James O'Neill**
Signature of Candidate:	***James O'Neill***

Name of Candidate:	**Rejean Lagasse**
Signature of Candidate:	***Rejean Lagasse***

Name of Candidate:	**Zachary Ayotte**
Signature of Candidate:	***Zachary Ayotte***

Dated: 04 – February - 2020

Table of contents

- ## List of Figures

Each figure follows a naming convention of the first digit referring to the section, and the following digit(s) referring to the number of the figure in that section.

1 Introduction

The purpose of this document is to record the required technical and research-based information that was required to create the *Mini-Golf* dungeon game. The game was created during the time period of September 15th - February 4th, which includes the different stages of research, design, implementation, testing and documentation.

To code the game the team used the Integrated Developer Environment (IDE) Visual Studio 2019. This software was chosen because the base model is free, and it allows for easy debugging and problem solving.

To display images and play sounds, *Mini-golf Dungeon* was created using the Simple and Fast Media Library (SFML). SFML was built on OpenGL, so it also allowed for nominal use of particle effects and vector graphics which are used to give the player feedback on ball direction and powerup use.

The team used GitHub as their version control software. This allowed the team to share their work across multiple devices, and view or revert to any previous changes made to the code.

Various errors and bugs will also be covered, as well as a description of how they were dealt with, if there was an opportunity to fix them.

2 Problem Definition and Background

2.1 Purpose of the Project

The project's overarching objective was to create a video game in the maze genre while marrying elements from a two-dimensional golf game into it. Ultimately, the project provides users with a unique take on the maze genre while leveraging object-oriented programming and handcrafted physics within the code base. The team considered this to be a novel combination, as there are no games available on the market that both incorporate mini golf with dungeon crawling together.

The player uses golf like mechanics to move around the maze. Different obstacles and enemies such as doors and gophers will block the player's path, making the maze more challenging. These obstacles can be mitigated with power ups that the player can collect. Using these powerups the player can destroy parts of the maze or move past enemies. Just like golf, the goal of each level is to get to the hole in the least number of strokes possible. The player is awarded bonus points from three varieties of shot count ranges, will are given out at the end of the level. The repeatability aspect of the games comes in form of trying to end the game with the highest score possible.

2.2 Background

There are many different maze games. Many of the most notable ones such as Pac-Man and the Amazing Maze game require the player to move around the maze and complete an objective to be able to move to the next level. These games typically use the basic up, down, left, right movement to be able to navigate through the maze, but what makes this project different is that it uses a sport like a mini golf to move around.

By adding golf game controls to maze game design, *Mini-golf Dungeon* was able to home in on the positive aspects of a maze game, while discarding the negative ones. Players must time a button press to determine the power and accuracy to their movement, adding a fun challenge to what is usually a rote and mundane mechanic.

2.3 Needs Statement

Due to this project being a college assignment, marketplace demands are of secondary importance. Instead, emphasis was placed on providing a fun user experience for players and showing each individual member's understanding of SFML and object-oriented programming techniques.

Additionally, the golf theme of the game provided the unique opportunity for group members to display their understanding of two-dimensional physics and geometry. The project used an idealized version of real-world physics to provide a degree of realism and believability to the game.

2.4 Scope

The scope of this project was to create a single-player game and complete it by February 4th, 2020. It required a robust physics simulation that allows the user to move around the map as though they are a golf ball.

To provide a good user experience a framerate higher than 60 frames per second was to be maintained. A responsive control system was also necessary: ensuring that the user's inputs were processed and executed quickly. The game's physics system must also remain consistent and intuitive. The physics system is how the player navigates around the game's world, and is their primary method for interaction, so if the physics system is in any way off putting, then the player will not have a good experience.

There are several levels in the game, each with a different layout and obstacles to challenge the player's movement and navigation skills. Levels progress in difficulty, easing the player into the game's mechanics before challenging them with new material. To accomplish this, individual levels were handmade and featured unique puzzles and obstacles. As such, there is no randomization to level generation or enemy placement.

Because levels are pre-constructed, players are encouraged to replay levels to get a higher score. The game has a system to compare previous high scores to the player's current attempt. Players can also compare themselves to other players by importing a database of other player's high scores *Mini-golf Dungeon* is purely a single-player game, so comparing high scores add a degree of player interaction and competitiveness.

2.5 Project Management and Key Tasks

At the beginning of the project, a Gantt chart was created outlining the work needed to be done during the 15 weeks *Mini-golf Dungeon* would be under development. One sprint was assigned to each week of the Gantt chart. These sprints were then further broken down on an online Kanban board on the website Trello. This approach was used because proprietary Gantt chart software such as Microsoft Project would be an unnecessary expense, as well as more difficult for four individual users to modify. Due to the team's inexperience in long term project planning, being able to easily modify tasks was crucial.

3 Literature Review and Research

3.1 Initial Planning of Mechanics

The development of any game requires deliberate thought on the game's mechanics before production begins. Author Casey O'Donnell states that "Mechanics is a term that encompasses those rules that are applied when the player interacts with the game, and there is no need for a definitional distinction between rules and mechanics" (O'Donnell, 2008, p.3). These core mechanics in a game define the rules of the game as well as the user's experience with it. Because they play such a crucial role in how a game plays, the initial design of a game must focus on the game's mechanics.

Before game production begins, the dimension and perspective of the game must be decided. Dimension and perspective can fundamentally change how the user interacts with a game. For example, in the 1985 game Super Mario bros., the user only goal is to get from point a to point b. This results in a very simple platforming game, where the player is judged on quick reflexes to overcome obstacles. Contrast this with the 1993 game Doom, whose mechanics allowed the player to move around a pseudo 3-dimensional environment. This change in the mechanics of the game added an exploratory element to the gameplay, shifting the user experience to a more methodological approach to challenges. The intended effect on the player must be decided alongside the mechanics.

C++ is one of the most common languages for game programming. One of the main reasons why game development in C++ is important for the use of game development is the accessibility to Direct X. Structural with the coding, the use of C++ allows for the inheritance of classes and the ability of reusing the class for different objects in a game, otherwise known as polymorphic behaviour. Other than there being an accessible route to the graphics, the use of memory management in C++ can be used to utilize the memory objects in a game.

With the C++ programming, the use of object-oriented programming and use of its technique can be used to reuse the code that is already there by instantiating a new object that can use the code. This allows the programmer to manually manage memory, which is important in games where performance is key. Memory can be allocated and deallocated at will. A constructor "is

used to initialize the object's variables. The destructor function is called automatically when an object is deleted." (Kelly, 2012 p.5)

As video games have grown in complexity developers are increasingly utilising Model-Driven Development to create game mechanics. Model-Driven Development is a methodology that works to create abstractions of gameplay concepts and automate simple coding tasks (Cubel and Rayno, 2008, p. 1). Because Model-Driven Development's focus is "the reuse of common characteristics in games of the same genre," it can be used in conjunction with design methodologies such as agile (Cubel and Rayno, 2018 p. 2).

With game development, the use of the libraries and the framework also help the creation of a game. The one framework and library of focus is the Simple and Fast Multimedia Library, but really the use of the OpenGL library is what a game should be looking if developing 2D, because the open use of the library is suitable for the creation of a game.

3.2 Character Interaction with Environment/ Obstacles

Every video game has the player interacting with the game in some way. This interaction with the text is what distinguishes video games as a medium. And these interactions often take the form of an obstacle to be overcome and conquered. Obstacles are such a universal concept in video games because they add complexity to the experience and prevent the player from getting to their destination with ease. And when the player does reach their destination each obstacle can be looked back on as an accomplishment. These obstacles form the backbone of interactions within the game itself and create a reason for people to play the game. When designing games, obstacles can be broken down into two broad categories: environment and enemies.

Environmental obstacles may not be actively hostile towards the player as enemies, but they are just as integral to gameplay. Some games force the player to be affected by gravity, while others make the user float in the air. The Environment of games can help the player reach its destination or even hinder the players movements to make it harder for them to reach their destination. Like for example a hill in a game makes the player have a harder time either seeing or slowing down the player when attempting to climb the hill. On the other hand, the player can see further if they

are on top of the hill and can even move faster going down since gravity is with them rather than against them.

Environmental obstacles come in different forms of difficulty. Something as simple as a wall that cannot be passed, is an impediment to the player because their character must now navigate around this wall. Other obstacles may slow the player down or send them to an disadvantageous position in the level. But no matter how complex the obstacle becomes; the player must be given the tools to overcome it. To accomplish this the game designer must ensure "that the mechanics has features that allow triggering different interactions with (the environment)." (Carlo, 2007 p. 6).

Enemies differ from environmental obstacles by affecting the player more directly. Enemies can interact with the player in a variety of ways such as pushing the player away from the objective or knocking the player into a more difficult situation. Enemies can also interact with the player in less direct ways. The consequence of hitting an enemy could be to reduce the player's score. The most penalizing action an enemy can perform is killing or otherwise destroying the player character, forcing the player to restart the level and resetting all the progress that the player has made. The 1985 game Super Mario Bros.'"core gameplay is centered on avoiding enemies, and the locomotion system is the key core mechanics," and the penalty of hitting an obstacle —either an enemy or a pit— is to restart the level (Carlos, 2007 p.5). This is a particularly punishing example and many games are more forgiving; however, avoiding obstacles is a key mechanic in many games to varying degrees.

A third aspect of obstacles are powerups, though these are more often beneficial to the player. Powerups are bonuses which can be collected by the player and gives them an advantage. These powerups can affect how the player interacts with certain objects, change some of their properties or negate obstacles. The importance of powerups means that they "are the enablers of reaching goals. But they are also the goals themselves most of the time" (Lange-Nielsen 2011, p.10). If a player needs a key to get through a door, then obtaining a key becomes their immediate goal. Without that specific key powerup, the door is just a wall: another obstacle in the player's way.

In terms of programming, each obstacle or powerup is its own object. Object oriented programming is all about segregating code into small chunks of connected code called objects or classes. Despite this, there must be some connectivity between objects. The player character must know when they hit an obstacle after all, but by creating obstacle objects, each obstacle can be programmed to behave the way the designer intended.

In Conclusion, interaction is an important mechanic for video games. And they must be created to allow the player to have the intended experience with the game. Interactive elements make the player feel immersed in the game world and are necessary if the game is to provide the player with challenge.

3.3 Game Physics

Physics are an integral part of virtually any game's design. Whether the system is as robust as a top of the line physics engine or as simple as the ball bouncing off the paddle in Pong, allow the player to understand the rules of a game world, and what they are able to do there. At their best game physics lend a sense of believability to the game world and allow the player to immerse themselves in the experience. However, game physics must be simple, intuitive, and focused; otherwise they become a detriment to the gameplay experience.

When interpreting physics into a video game environment there is a unique challenge. Namely, that humans are innately good at coming up with qualitative solutions and predictions when faced with a physics obstacle (Ge et al, 2016, p. 1). The simple act of bouncing a ball off a wall has a tremendous amount of complex mathematics to it, but in any given park children as young as four years old will be bouncing balls and knowing the exact outcome of this action. Those children, and most adults, view physics as a set of rules that uniformly work for them in their day to day lives.

This innate understanding of physics can be a great asset or a detriment to game design. Humans have physical reactions thoroughly ingrained into them, and this means that players come into games with preconceptions about how a simulated physics system should react. This is beneficial in that it saves time on the game developer. After all, many minutes would be wasted if one had

to thoroughly explain the cause and effect of what happens when a ball rests on a hill. However, if the video game strays too far from these preconceived notions it causes a cognitive disconnect with the player. Physics and animations that do not work as expected feel, in the words of game designer Scott Rogers, “floaty and gamey — a feeling that is always less desirable to players” (Rogers, 2014 p.178). If the ball doesn’t roll down a hill or comes to a halt in a seemingly unrealistic manner, the player can no longer trust their own judgement when it comes to the game’s physics system.

Still, game designers cannot hope to create a completely true to life physics system. Limitations in the number of calculations a computer can perform, not to mention the time it would take to model such a system, make this an impossibility. However, this can work to their advantage. Bataglia et al states that

> For both game engines and the brain’s simulations, the models do not need to be accurate in any sense that physicists would recognize; they need only produce results that look reasonable at the spatial scales that humans perceive and act on (2017 p. 655).

While humans have an innate grasp on physical reactions, they use shortcuts to get this information quickly (Ullman et al, p. 654). So, when programming a physics engine, it is more important that its properties correspond to what the player expects to happen than what a true to life simulation would be.

A perfect representation of physics is neither possible, nor desired for video games. Instead, programmers must limit their system’s scope and create a reasonable facsimile of physics. In their work on qualitative reasoning in artificial intelligence, Ge, Lee, Renz, and Zhang created a physics abstraction that they considered both, the ideal scenario to test their algorithms, and a reasonable environment to present physics puzzle games such as mini golf. The rules of this environment are: “the environment is a restricted 2D plane, objects are 2D rigid bodies…, there may be a uniform downward gravitational force, objects… obey Newtonian physics, physics parameters of objects… remain constant” (Ge et al, 2016, p. 1-2). These basic rules provide the player with enough information to reinforce their preconceived perception of physics without overwhelming or surprising them.

These principles can also apply to the aspects of physics with which the average player would have limited experience. On the topic of the computer game Asteroids, author of "Mathematics and Physics for Programmers" Danny Kodicek stated, "inertia and momentum are conceptually hard concepts to understand theoretically, so a game like this… is an excellent way to learn them" (Schiffler, 2012 p.73). Asteroids is a game about managing propulsion and inertia in a near frictionless environment; however, players were shown to grasp these foreign concepts quickly. By limiting their focus to a few simple physics interactions, the developers of Asteroids ease the player into learning the rules of the game.

Despite human's ability to grasp physical concepts, Kodicek cautions game programmers against creating a physics simulation without due cause.

> Players don't use anything unless it either helps them progress in the game / get a high score, or makes the game itself more fun… making a game more generally realistic won't teach us anything we can't learn by looking out of the window… as opposed to Asteroids, which is an example of a controlled simulation. (Schiffler, 2012 p.74)

Much like Ge et al, Kodicek believes that the best use of game physics, whether to entertain or educate, is in a simplified form with a limited focus.

Physics are a powerful tool for video game design. Because people begin grasping these concepts from an early age, virtually any player will readily understand the underlying mechanics when presented with a physics system. However, in-game physics can go too far. Even if one were able to create a fully realistic physics system this would be inadvisable. Players can become overburdened by systems that are not directly linked to gameplay, and work will have gone into a system which players will, in the best of cases, actively avoid. Physics systems must be focused on a few simple laws that have a tangible impact on the game itself. Therefore, the physics programmer must have a thorough understanding of the game they are making, and what set of physics are most appropriate.

3.3 Art and Prototyping

3.4.1 Video Game Art

All modern video games, since the popularization of console and PC gaming, have had a unique visual art style. Because the player will be looking at the in-game world or scenes for the duration of their play time, how a video game looks visually can be just as vital as having seamless, bug-free programming and appealing features. As such, the player should be able to enjoy or appreciate what they are viewing, otherwise they may find it straining or unpleasant when experiencing the game.

Choosing the right art style can have a major impact on how a game is received by critics and casual players alike. The style should reflect what the game is about and complement its mechanics. Games can be easily distinguished by the art style, which also has an impact on the fans that it attracts. Even games of the same genre can vastly differ in their respective visual styles such as in the case of Darkest Dungeon and Celeste. Both games are played in a 2D perspective, but the overall theme for them is much different. Darkest Dungeon is styled with dark, gothic graphics, to compliment the Lovecraftian vibe the game offers. The latter, however, features a more colourful palette, in a more pixel-based visualization.

Successfully creating acceptable art takes a considerable amount of time, and collaboration with fellow team members. As said by Chris Solinski in "*Drawing Basics and Video Game Art: Classic to Cutting Edge Art Techniques for Winning Video Game Design",*

> *"*present your work daily to members of your team. Artists often find daily presentations an uncomfortable chore because few want to show work, they deem unfinished. But daily presentations are good because the key concepts can be refined through quick iterations and team feedback before precious time is spent on details and polish*" (*2012, p. 189*).*

As such, it is key to ensure all project participants are on the same page with the design aspect. Receiving feedback from regular meetings will help shape the artistic direction of the game. Even though there are dedicated artists in a game development team, it doesn't necessarily mean their choices alone will shape the graphics of the game.

The visual style in video games is often considered to be “the most important aspect in marketing a game” (Mary Keo 2017, p. 2). Whenever a new game is announced, the thing that catches people’s attention at first is the art. It is something that causes *“people... to gravitate more towards” (Mary Keo 2017, p. 2)* certain titles, especially if the game has a well-made, or visually appealing graphic design. Among the styles present in games nowadays, the most popular types include realistic and stylized art. Realistic games focus on providing as close of a recreation of the real world as possible, while stylized games tend to take a more unique, custom-made design by the artist. The latter is also generally considered to be the preferred style for 2D games, as many of the classic platformers such as Super Mario, Legend of Zelda and Mega-Man (Mary Keo 2017, page 15), to name a few, were all created with pixel graphics due to the limitations of technology at the time. In modern days, many video game developers decide to use the pixel art style as it provides a nostalgic feeling for many players who grew up playing SNES and PlayStation games. Another reason for this is that the style is “appealing and relatively easy to create” (Mary Keo, page 18). Some examples of modern pixel-art themed games include Crawl, Pixel Dungeon, Celeste, Dead Cells and Papers, Please.

Hand-drawn games have also made a resurgence in modern times, as they are another way of providing a unique art and feel to the game, knowing that all assets were drawn by hand by the artist. Such games include Hollow Knight and Don’t Starve. Both of these, In conjunction with the right game genre (In this case 2D side-scrollers and top down- dungeon crawlers) can help the players to feel immersed in the game world.

3.4.2 Prototyping and Game Testing

The development cycle of a game requires constant attention and planning to ensure all tasks are achieved by the developers. As such, the entire process must be carefully monitored and assessed constantly. This is where the creation of prototypes comes into play. Prototyping holds an important role in development, because they are used to help set up the basic structure and test features of the game that will be included in the final version.

The very first type of prototype which can be created is paper based. Just as the name suggests, it allows the developers to do basic, rough sketches of certain aspects of the game. Drawing on paper allows for developers to visualize the game early, as well as show some of the functionality or game progression. According to Giles Schildt, an ex-director of game development at Steve Jackson Games, another great idea is to use old board games as inspiration and a means of prototype play testing. These games often come with maps, figures and tokens of sorts, which can be used to make custom maps and scenarios involving game characters. The key advantage of using paper based, or alternatively called, analog prototyping, is the fact that it can cut cost and help decide "whether the project is even worth pursuing as a digital product" (citation from Schildt). Neverwinter Nights is an example of a game that came to life from a board game: Dungeons and Dragons. As Schildt states, the key difference is that enemies in the game are scripted with certain behaviours, must like the dungeon master in D&D would call out information about monsters and their encounters in the tabletop game.

A step forward from paper-based prototyping, is the combination and implementation of art and narrative assets in a dedicated software, such as Construct 2 or GameMaker, which are used for "rapid prototyping of games for specific game genres"(Engstrom, Brusk, and Erlandsson 2018, p. 156). Both are programs that provide an easy to use interface, which allows the user to spend more time arranging the different art assets. Rather than coding everything from scratch, these programs come with pre-established interactions and triggers for events, that let the player assign them to objects in game, to provide functionality. The general use of these programs is to provide a rapid, working prototype that will serve as a basis for the finished product.

The game development process consists of two testing phases: alpha and beta. Both have their own specific goals which need to be accomplished. The alpha phase comes first and is used to test the software for bugs, and ensuring all features are working correctly, as intended. In general, the beta phase is used to determine whether a game is ready to be released to the public. As such, game developers generally invite a select number of people to play the beta version and receive feedback from the public.

An example of how tests are carried out in software creation, is the game that the American Veterinary Medical Association (AVMA) which was made via the cooperation of AVMA members Lori Goszczynski and Colleen Krahulik, and the game developer staff of Game Gurus. The former, initially unaware of the development process, were given a full experience of how a game is made, particularly the testing phase. The game was designed by both teams, and thanks to the communication between the two, all features and issues in the alpha phase were dealt with efficiently. As described by L. Gosczynski and C. Krahulik, the beta test of the game performed with the inclusion of the target audience showed that there is "too much text within the gameplay" (2014, p. 5). Thankfully, the Game Gurus developers were able to edit the text content in the game to a smaller amount, and added some additional flavour to the player characters, on the AVMA representatives' request, thus eventually concluding the beta phase, and allowing the preparation for the full release of AVMA Animal Hospital game.

3.5 Conclusion

The purpose of this review was to gather and describe the information gained from studying the topics of relevance to the game design development cycle. The research documented in this review served to increase the knowledge of this team's members in regard to the video game development process about to be undertaken. The topics that have been researched consist of game mechanics, system structure, game physics, player interactions with obstacles and environment, video game art, prototyping and testing. All the areas covered served an important role in the design, creation and testing stages of the team's game.

In the physics system of *Mini-golf Dungeon*, effort was put into ensuring that new players were not surprised by the physics system. Players may miss a shot due to their own inaccuracy, but the game behaves consistently, giving players the opportunity to learn from their mistakes. Care was also taken to not overwhelm players. Focus was placed on simple trigonometry and friction, so that players do not need to worry about more complex concepts such as gravity or wind resistance.

Likewise, the art was designed to immediately convey to the player all necessary information. Upon a quick glance, players are aware of where the ball is and what obstacles lie in their path.

Simple concepts such as golf balls get stuck in sand, and the goal of golf is to go to the hole are conveyed entirely through visuals.

Finally, the rules of the game were decided upon and tested prior to the start of full development. Every team member knew the game they would make, and what features needed to be included beforehand. Because of this, time wasted writing complicated scripts was kept to a minimum, and the team could focus on work that needed to be done.

4 System Design and Configuration

The system architecture that was involved in the production and the creation of the project contains the structural layout of the project's design. In the design phase of the architecture, the first thing that was done was to draw up the suspected concept of what the application to the game engine would look like on the diagram. The concept that was decided was the application layer, the game as it works, to the engine layer / logic, of how the game operates and runs. From the concept, the GPU serves as the unit to manipulate the data processed for image processing into the frame buffer and subsequently displayed on the application layer of the game. The CPU in this case is used to serve the program and the project as the central processing unit of the game, where the logic, instructions, etc are executed as a part of the computer program. In the instance of the team, the CPU acts in this way by performing the primary instructions of the game / computer program. The RAM component for the system architecture loaded the instructions from the engine class and the other classes into memory, a lot of the pointers in the project related to the use of memory in the game and where the instruction could then be called from. The database module was designed so it would link up to the application layer the game, so in this instance, the team decided that in doing this, it would allow for a remote connection to the server in the location it was based in Europe and the application could send the data over the network to the server in that location, therefore, the team would have stored the details of the player from the game into their database. They would then be able to retrieve that data from the server, in its location and transfer the data back to the application layer over the internet, when commanded to do.

The other thing the team had to consider with the architecture of the game was the game logic behind the code and the structure of the game. To ensure that nothing would happen with the application as ran, the team had to consider a design in which the game as it ran would remain running or active until the user had successfully closed the application. They also had to take into consideration the fact that the game should play within the menu state or any other state in the game, except for the play, so there weren't any mishappens or flaws in the design of the project. This is covered in the bottom of the diagram, where the game logic is mostly found in the engine class of the project. From here, all the necessary commands given into the CPU are executed to run the application.

Finally, the team had decided that use of the TCP protocol was slightly important with the connection of the database to the game or application layer, since the data from the database would travelling over the internet. Even though the IDE had handled some of this connection status for the team, there were few key configuration steps from the project properties. Taken that needed to configure their database settings as well to be able to connect to the database from the application layer., for this a connection string provided by the Azure database had to be used.

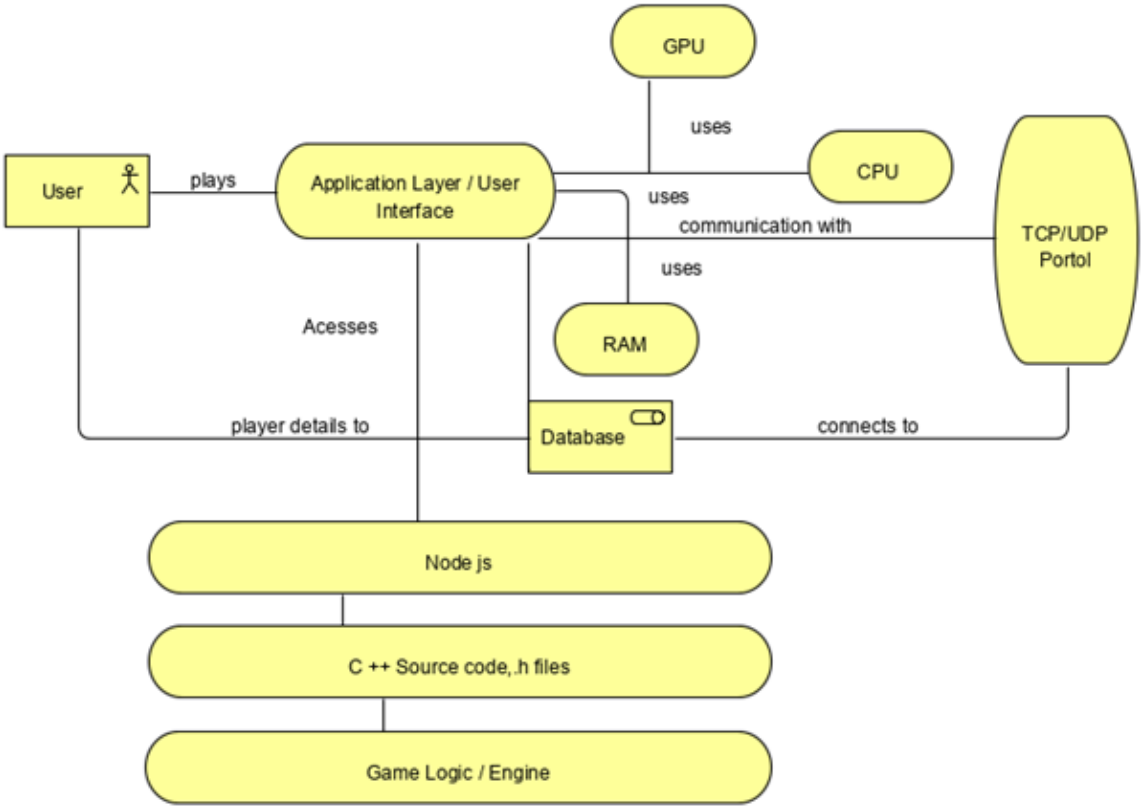

Fig. 4.1 – system diagram

4.1 Class Diagram

From the class diagram taken from Visual Studio, the team made a visual representation of the class diagram from the code and classes involved. The idea of this was to show the class structure and the use of inheritance of where it might be in the game as well as polymorphism, etc. From the team's experience in software development, there is some form of inheritance mainly done with the objects / obstacles that they have placed around their game. Also, it shows that they performed some polymorphic behaviours from the obstacles that they create.

Their diagram of the classes allows for the use of planning and understanding in the development of the project. In some of the cases, the class designer in Visual Studio acted as guided to see what class was connected to another as well as a way to where the inherited features of their game will work or won't work at, or in the case where they think that the classes are not using the proper form of inheritance, like an inherited class is being used in a two way street, since inheritance is only a one way street, usually, down. The class designer not just shows the use of inheritance in the project, but it also shows the functions and the variables used by the classes in each of them.

The use of the functions and the variables that are contained in the classes of the class diagram shows whether that a function or variable would be accessible to a class or not. It also shows the variables in which are fixed by a value.

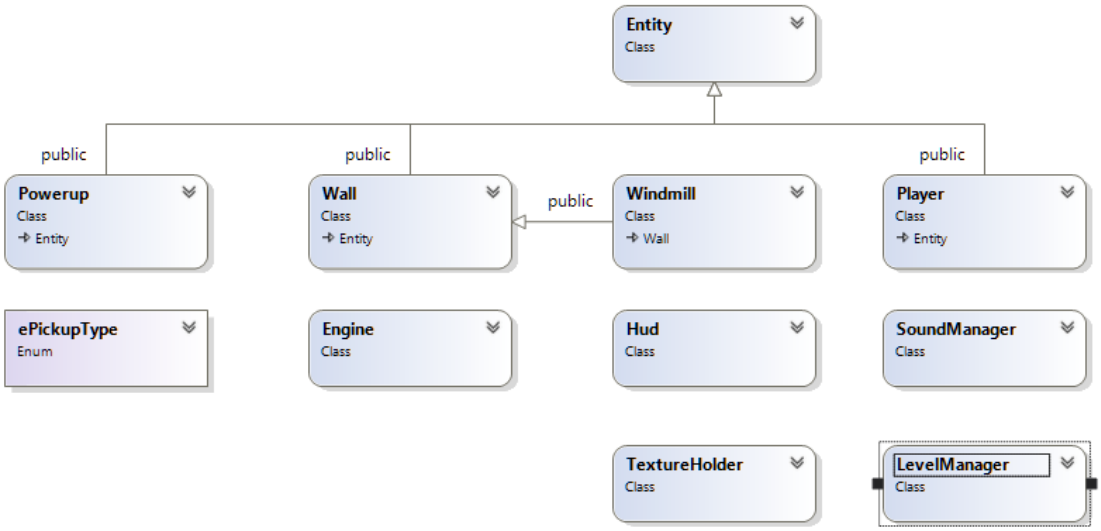

Fig. 4.2 – class diagram

4.2 Sequence Diagrams

From the sequence diagrams that the team produced, the diagrams give the visual representation of a user and how the user would interact with each step of the game. The diagrams show how the interaction or the suspected outcome of the user's input or interaction with the application will turn out. From the first diagram, the Main menu diagram, this shows how the user will interact with the menu options in the application and the suspected outcome of each input from the user. Within the first diagram, the user enters the application and is greeted by the Menu UI and from there the user must choose what the option would be. Typically., like any other application, if the user decides to quit the application, the application should be terminated if so. Otherwise, the user has the option to either; enter the game, view the leader boards, view the instructions video or to select a level of their choosing.

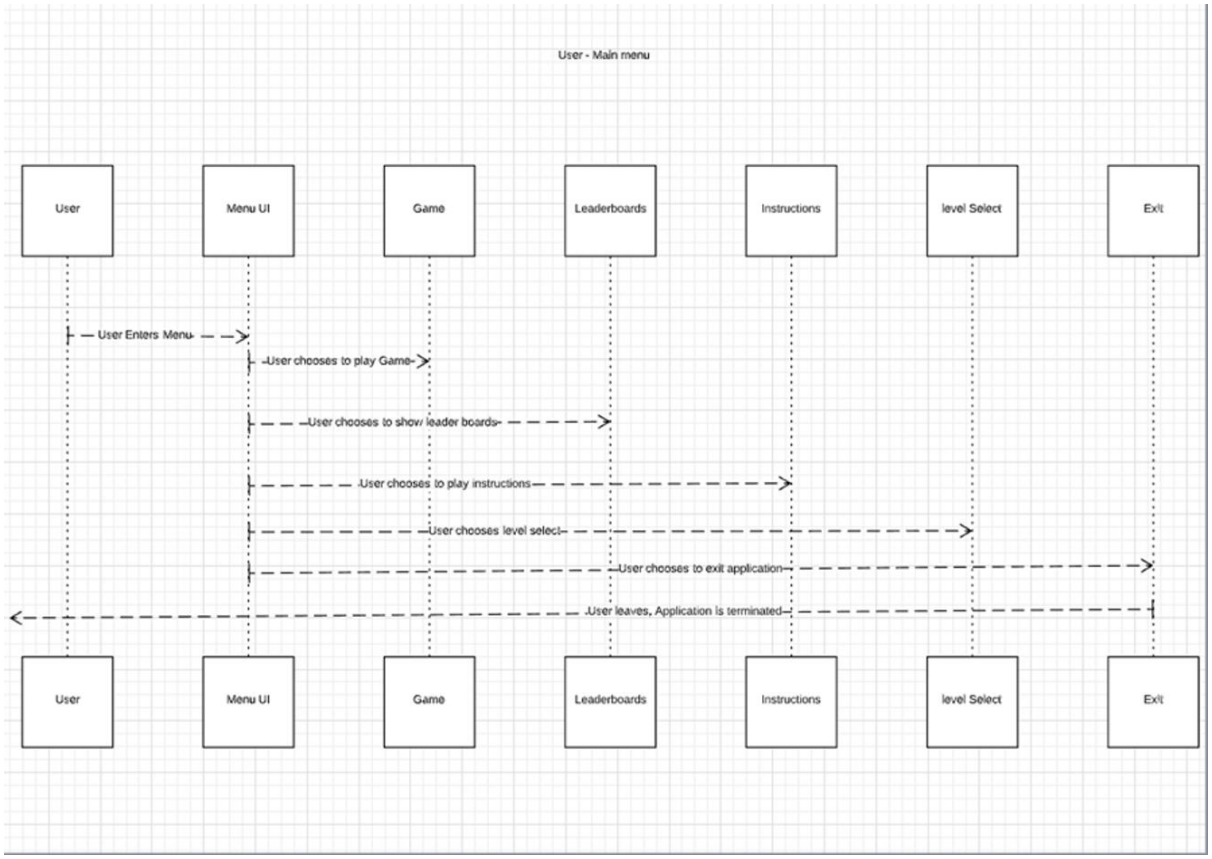

Fig. 4.3 – sequence diagram of game options

The next diagram is showing the game and how the user is interacting in the game state. The user plays the game according to the instructions and awaits the encounter by a pickup object or an obstacle in the game. If they do, they must avoid the obstacle or the retrieve the powerup. Then they would proceed as normal until the goal is reached. If the user has reached the goal, then the user is brought to the next level or when the game is finished.

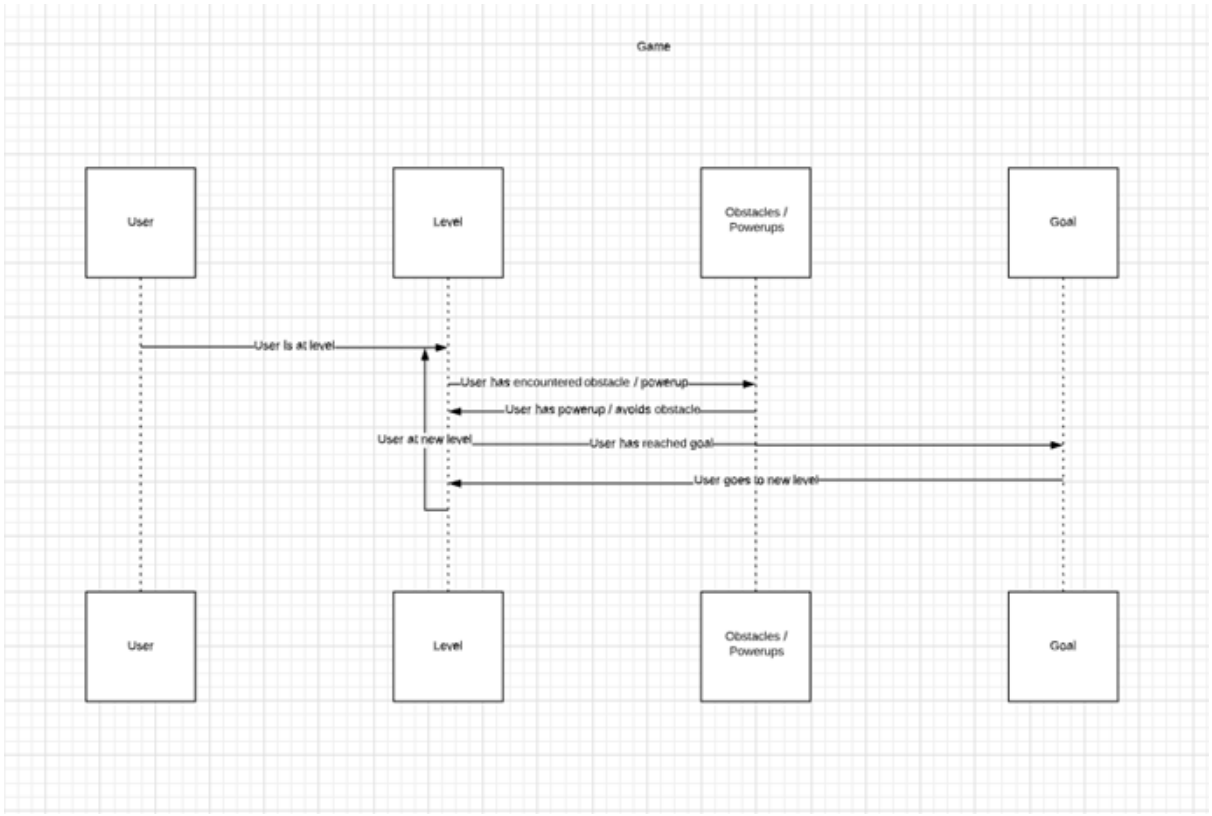

Fig. 4.4 – sequence of in game interaction

The next diagram shows the leader boards of the game. When the user has chosen to view the leader board, the application sends out a message to the database to retrieve the data to display back to the user. From here, the user can see who is in the lead or who is the competition, etc.

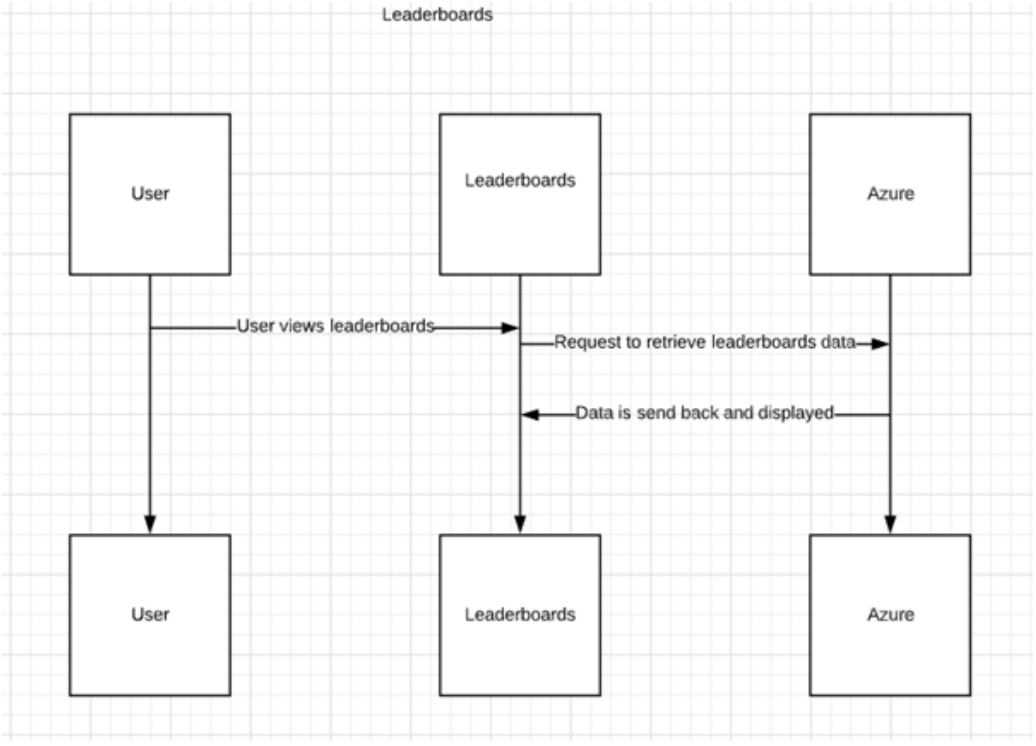

Fig. 4.5 – sequence of leader boards

The following diagram is showing what happens if the user had chosen the instruction video. The video will play for the user and the user will watch the video to the end, if the user wants to leave at any given time, they can by using the backspace key to do so.

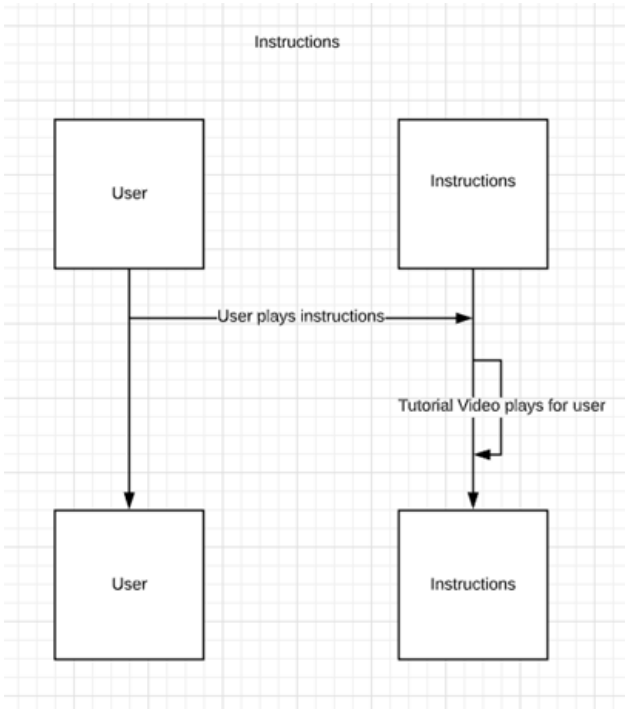

Fig. 4.6 –sequence diagram of instruction video

5 Example of use

5.5 Sound Manager

The sound manager class is a class that could be reused in most games. The sound manager is a singleton which means that this class can only be created once and then when creating another will return a pointer to its original creation. The sound manager also keeps track of all the sounds that are in the game making it a great piece of code that can be used in most any game that needs sound effects or music.

5.6 Texture Holder

The texture holder is an important class that helps improve the performance of the game. The advantage of using a texture holder other than having all the objects hold their own textures is that it saves memory by only requiring one reference that the game can point to and grab that sprite whenever it needs it. Since most games have some sort of sprite loading to the screen it can be used in most games that have some sort of drawing to the screen.

5.7 Player Physics

for the player physics of the game the method used to move the player can be implemented into multiple different games. In our game we use a three-click system to determine the direction, power and the accuracy of the game. This type of system could be used in games that require all these factors or even only a few of them. This feature can be used in multiple sport games since most sports require these three factors. Golf games, bowling games, and pool games can use this type of feature to add difficulty and skill to gameplay instead of basing it on pre-set calculations or luck. This feature can also be modified to be used in a pinball game when the player pulls on the plunger to determine the ball's power when entering the playfield or any time power needs to be determined in a game.

6 Testing & Implementation

○ 6.1 - Testing

As part of the project's development cycle, testing was used to find and resolve many emergent issues that have occurred during programming. Due to the nature of the project, it was key to eliminate as many of these issues as possible. To test the game, a multitude of different types of tests were utilized, ranging from using breakpoints, Visual Studio's debug feature, and playing each level of the game to test the player collision system and how it interacts with the environment and obstacles.

Debugging code is an important step to testing the code to make sure everything is going on correctly or not. There are many ways the programmer can do to debug code in this project some debugging methods were part of the integrated developer environment (IDE) Visual Studio 2019 but can be found in most IDE.

Debugging happens even when the programmer isn't in the code. The simplest way to debug code is by running it and determining if everything is working according to what the programmer is doing. If something inside of the game isn't working correctly the programmer can use the following methods to determine what the problem is and fix it accordingly. This method of debugging doesn't tell you what the problem is but can give clues about where in the code this is happening for example if the ball isn't picking up the powerup the problem might be where the powerup and the ball's collision detection are.

Another method of debugging is commenting out section of the code. This isn't the most efficient way of debugging since it will only let you know where the problem is and if forgotten can be left in the code as comments.

The most common way to debug your code is to have the program print out the value of the variables that you need by using the cout command (in the iostream library). This way of debugging is not the best way of debugging but is nonetheless used for the sake of convenience. Most of the time printing out variables to the screen or even just a simple line saying "I got here"

is a way to check to see if the programme has reached that part of the code or if the values for the variables are the correct values for that section of the code. Then depending on the outcome of the printed statement the programmer can find where the problem is occurring. The problem with printing out information to the command line is the fact that when you debug using this method it requires you to have access to the IOStream library inside of this specific section of the code and also requires you to write inline of the code. This is only a problem whenever you are done finding the problem and need to remove the section of the code that prints it to the screen. If forgotten this can be pushed to all of your teammates which is unnecessary code that shouldn't be there.

Breakpoints are one of the key features that help you determine what is happening in your code. In Visual Studio a simple left-click on the side of the code (on the line number) will add a breakpoint which allows the programmer to do multiple things. The main aspect of a break point is it will stop the programming whenever it reaches the line the breakpoint is on. This can be used for testing if statements to make sure that the programme is going through them whenever they are required to do so. Another aspect of the breakpoint that is useful while debugging is the fact that whenever the breakpoint is hit you can determine the value of each of the variables by hovering over them. This is especially useful because it allows you to keep track of the value of each variable and make sure that they are changing in the way they are supposed to or even figure out why the number is a different number than it's supposed to be. One of the common issues that were happening while trying to debug were that the variables were assigned junk values, therefore, they were not being assigned correctly or were not assigned in the first place. By using the breakpoints, we were able to determine the reason why the section of the code wasn't working correctly. The only problem with using the breakpoints would be for occasions where what you are trying to check is in a loop since you will have to constantly click the continue button to check everything in that loop. But even this can be prevented by using the breakpoint's features which allows you to set conditions to when exactly you want the programme to stop.

Besides the use of breakpoints and debugging, another method of testing utilized in this project was to play the game and experiment with its features. To ensure that each level of the game is playable, and most importantly, provides a fair but challenging experience to the player, the

game's level layouts and collision with walls and obstacles were tested extensively. Each level in the game was given approximately the same amount of attention when it came to test for collision and obstacle functionality. For the purpose of testing, the zoom level of the game was set to 2 in Engine.cpp, to allow for full map visibility.

Level one was the most used level when it came to examining ball physics and ensuring each powerup and obstacle works, as it was the smallest level and easiest to get around in, with a large open area right near the start. Any time a new game object was implemented, it was first placed in level one, which can be considered a testing ground for it. Once the object was confirmed to work, it was then added in other levels in the LoadLevel class.

The first two levels were straightforward when it comes to testing. Over several weeks, the tile layout has been changed significantly, as some of the walls left too little space for the player to accurately progress through. Being the early to mid-stages of the game, the player should be given time to learn how to efficiently navigate the mazes without punishing them as much as end game levels. This was not the case, as some wall tiles were either placed that massively inconvenienced the player, or where placed in diagonal patterns which made there was more surface area for the player to hit. Diagonally placed tiles were kept to a minimum as they could be a sizable hinderance.

Level 3 and 4 proved to be the most challenging and time-consuming regarding running tests. There were several emergent issues related to the windmill, door and weak wall obstacles. At one stage, whenever the player would collide with the door and weak wall while an appropriate powerup was enabled, the ball would pass through the object, with the powerup remaining in the item list. After some more testing, the wall objects were spawned one block away on the x axis from their original position, then moved back, only to work as intended. Level 4 features the windmill obstacle exclusively, which for some reason decided to occasionally drag the player along. Even after rigorous testing, there was no sufficient ground to formulate a solution for this and was assumed to be a chance occurrence.

After each level was successfully tested for a fair, playable layout, some of the obstacles and powerup placement was also tweaked, accordingly to the level and the player's convenience.

Thanks to regular collision tests, it was possible to regularly update the main movement algorithm to ensure than the ball collides walls in an appropriate manner. Over time, the code became more stable and allowed for more accurate collision scenarios whenever the player collided with other objects. The movement algorithm required a significant amount of testing and tweaking to get right, and the final version ended up being the most stable and reliable one.

○ 6.2 Implementation

The entire project was created using C++, with Visual Studio serving as the IDE. The assignment required the usage of the Simple and Fast Multimedia Library (SFML), which is a cross platform software development library that provides an application interface for multimedia components in games.

6.2.1 Movement Algorithm

The game uses an algorithm that is one of the major aspects of functionality. A major revision to how the game loop functioned, and a late change, was how collisions are handled. As figure 1 demonstrates, each wall tile near the ball is checked to see if they are colliding with the ball. The blocks in figure 1 are labelled, one through eight, in the order that the program checks their float rects' in. The rows are checked from the top downwards, and each tile in the row is checked from left to right.

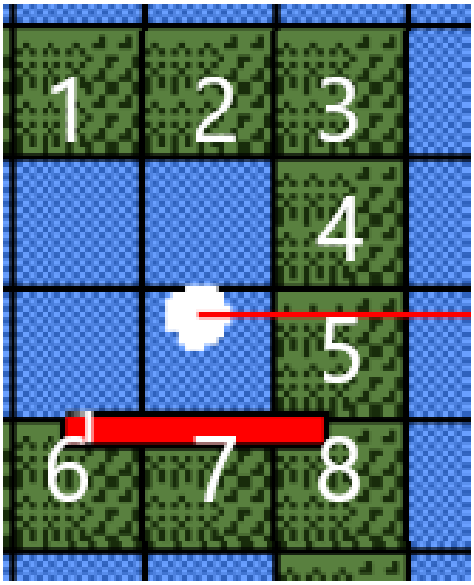

Figure 6.1: The order of the collision tests.

This system is still how the tiles are checked in the final game. What has changed is when the collisions are handled. Initially, the collision handling occurred at the same time the collision was detected. As seen in figure 2, line 126 determines if the ball has collided with a wall, and lines 132 through 135 handle the collision. If enough time has passed since the last collide, line 133 changes the direction and speed of the ball, line 134 plays the ball-wall collision sound effect, and line 136 resets the collide time. Most of the time this algorithm worked as intended; however, problems would arise when multiple wall tiles contained a section of the ball.

At its fastest, the ball moves approximately 10 pixels per frame. Because of this, it is not uncommon for a wall tile to contain multiple collision points on the ball. Measures were taken to ensure that the collision system prioritized cardinal (up, down, left, and right) detection points on the ball over the offset diagonal points, but this only applied to points within the same tile. Should the top-left collision point of the ball be contained within tile 1 from figure 1 and the top collision point be within tile 2 of the same figure, then both the top-left as well as the top collision point logic would be applied to the ball.

```
125   //Check the Top and Bottom of the Wall
126   if (tile.contains(fTop) && fSpeed.y < 0)
127   {
128
129           //Normal Wall
130           if (type == 1 || type == 3 || type == 4) {
131
132                   if (Collide(GametimeTotal, dt)) {
133                           fSpeed.y = -fSpeed.y / 1.1;
134                           m_SM->playWallKnock();
135                           ResetCollideTime(GametimeTotal);
136                   }
137           }
138           //Sticky Wall
139           if (type == 9) { // Make Ball Stick to Wall
140
141                   if (Collide(GametimeTotal, dt)) {
142                           if (fSpeed.y != 0 && fSpeed.x != 0) {
143
144                                   fSpeed.y = -fSpeed.y / fSpeed.y;
145                                   fSpeed.x = -fSpeed.x / fSpeed.x;
146                                   m_SM->playStickyWall();
147                                   ResetCollideTime(GametimeTotal);
```

Figure 6.2: The old collision system.

To fix this, the collision system was reworked to only change the ball properties after each tile has been tested for collision. The first step is shown in figure 3. In the new CheckCollision() function, a tile is checked for collision with the ball on line 13 of figure 3. If the ball and wall have collided, then the member Boolean in the ball object is set to true, and the type of wall is contained in either iCollideTypeCardinal or iCollideTypeDiagonal, depending on if the collide point on the ball was one of the cardinal or diagonal directions.

```
12      //Check the Top and Bottom of the Wall
13      if (tile.contains(fTop) && fSpeed.y < 0)
14      {
15          bTopHitbox = true;
16          iCollideTypeCardinal = type;
17      }
```

Figure 6.3: Setting the collision Booleans

Once each tile around the ball has been checked and the Boolean flags are set, the program moves to the HandleCollision function seen in figure 4. Depending on the type of wall the ball collided with, this function will call either HitWall or HitStickyWall which will apply the correct modifications to the ball's speed. HandleCollisions also resets the player's CollideTime, iCollideTypeCardinal, and iCollideTypeDiagonal values in lines 69 through 72 or 77 through 79. Finally, the Boolean hitbox flags — such as bTopHitbox—are reset on line 81.

Once either the HitWall or HitStickyWall functions are called, the collision logic is finally applied to the ball. Line 98 of figure 5 checks to see if the iCollideTypeCardinal is a value other than 0. If it is, then this indicates that the top, left, right, or bottom hitboxes have been hit. If this returns true, then the hitbox flags are analysed in lines 100 and 102 to determine the modifications to the ball's speed.

```
65  void Player::HandleCollision(Time GametimeTotal)
66  {
67      if (iCollideTypeDiagonal != 0 || iCollideTypeCardinal != 0)
68      {
69          HitWall();
70          ResetCollideTime(GametimeTotal);
71          iCollideTypeCardinal = 0;
72          iCollideTypeDiagonal = 0;
73      }
74      else if (iCollideTypeCardinal == 9 || iCollideTypeDiagonal == 9)
75      {
76          HitStickyWall();
77          ResetCollideTime(GametimeTotal);
78          iCollideTypeCardinal = 0;
79          iCollideTypeDiagonal = 0;
80      }
81      ResetHitboxFlags();
82  }
```

Figure 6.4: Handling the Booleans

```
96  void Player::HitWall()
97  {
98      if (iCollideTypeCardinal != 0)
99      {
100         if (bTopHitbox || bBottomHitbox)
101             fSpeed.y = -fSpeed.y / 1.1;
102         if (bLeftHitbox || bRightHitbox)
103             fSpeed.x = -fSpeed.x / 1.1;
104     }
105     //Only process diagonals if cardinal directions hav
106     else
107     {
108         if (bTopLeftHitbox || bBottomRightHitbox)
109         {
110             float fTemp = -fSpeed.x;
111             fSpeed.x = -fSpeed.y;
112             fSpeed.y = fTemp;
113         }
114         else if (bTopRightHitbox || bBottomLeftHitbox)
115         {
116             float fTemp = fSpeed.x;
117             fSpeed.x = fSpeed.y;
118             fSpeed.y = fTemp;
119         }
120     }
121     m_SM->playWallKnock();
122 }
123
```

Figure 6.5: The HitWall function

If iCollideTypeCardina does equal 0, then this indicates that a diagonal direction has been hit. No checking is necessary, because as seen in figure 4 line 67 this function will only be called if either iCollideTypeCardinal or iCollideTypeDiagonal are values other than 0. So, if iCollideTypeCardinal equals 0 then iCollideTypeDiagonal must equal a different number. I found this syntax more readable than the alternative: a very long if statement checking each cardinal hitbox Boolean.

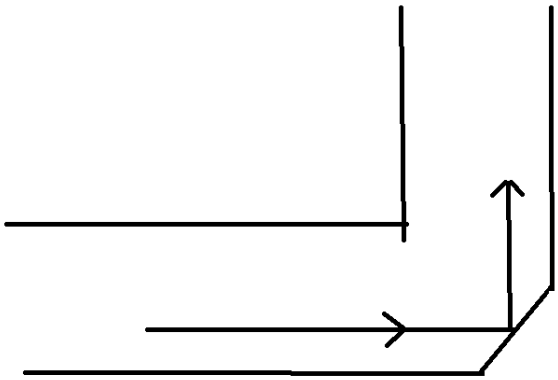

Figure 6.6: Diagonal bounce example

The logic for a cardinal direction hitbox is very straightforward. As seen in lines 101 and 103 of figure 5, the axis-speed for the ball is simply reversed if it hits the proper collide point. A collision with a vertical hitbox will reverse fSpeed.y, and a collision with a horizontal hitbox will reverse fSpeed.x. However, the changes made for diagonal collisions are more involved. As demonstrated in figure 6, a positive x speed will transfer to a negative (in SFML the top of the screen is y = 0) y speed, while a positive will become a negative x speed. To apply this, I create a temporary float value called fTemp on either line 110 or 116 of figure 5. This float stores the negative fSpeed.x value, fSpeed.x takes the negative fSpeed.y value, and fTemp is then transferred into fSpeed.y.

With this implementation the physics in Minigolf Dungeon behave in a realistic and immediately understandable manner. The ball no longer gets caught between walls or bounces erratically, which allows players to have a greater degree of control and plan their movement in advance.

6.2.2 Level creation

One of the dilemmas the team encountered involved how to create each level layout. Eventually, the idea was inspired by a tutorial on LinkedIn Learning, about how to program a game titled *Thomas Was Late*. This game is a 2D side scroller that utilizes file read-in to determine level layouts by assigning numerical values from the text file to images in a sprite sheet. We decided to use this approach as it seemed like the most efficient. It would mean, however, that the objects like powerups and obstacles would have to be placed individually in the code. *Fig. 6.8* shows an example of how a level was organized in a .txt file, to be then created in the game using tiles. *Fig. 6.9* features a complete level layout with a graphical representation of the numerical values from the file input. Blue tiles represent the floor, orange represents the walls and boundary of the level. Two of the obstacles, the sand pit and sticky were also included in the sprite sheet, and can be seen in the same image.

```
46      ifstream inputFile(levelToLoad);
47      string s;
48
49      // Count the number of rows in the file
50      while (getline(inputFile, s))
51      {
52          ++m_LevelSize.y;
53      }
54
55      // Store the length of the rows
56      m_LevelSize.x = s.length();
57
58      // Go back to the start of the file
59      inputFile.clear();
60      inputFile.seekg(0, ios::beg);
61
62      // Prepare the 2d array to hold the int values from the file
63      int** arrayLevel = new int* [m_LevelSize.y];
64      for (int i = 0; i < m_LevelSize.y; ++i)
65      {
66          // Add a new array into each array element
67          arrayLevel[i] = new int[m_LevelSize.x];
68      }
69
70      // Loop through the file and store all the values in the 2d array
71      string row;
72      int y = 0;
73      while (inputFile >> row)
74      {
75          for (int x = 0; x < row.length(); x++) {
76
77              const char val = row[x];
78              arrayLevel[y][x] = atoi(&val);
79          }
80
```

Fig. 6.7 - reading in data from file to determine tile placement

```
22222222222222222222222222
20000000000000000000000002
20333333333333333333333302
20300000000000030000500302
20300000000000030005050302
20300300333333339393553302
20300300000300030000050302
20333500000300000000000302
20305553333300030000300302
20305000300000033300300302
20300300300000000000300302
20300300000300000000308302
20333333333399933333333302
20000000000000000000000002
22222222222222222222222222
```

Fig. 6.8 -example of level layout as assigned in the Level1.txt file

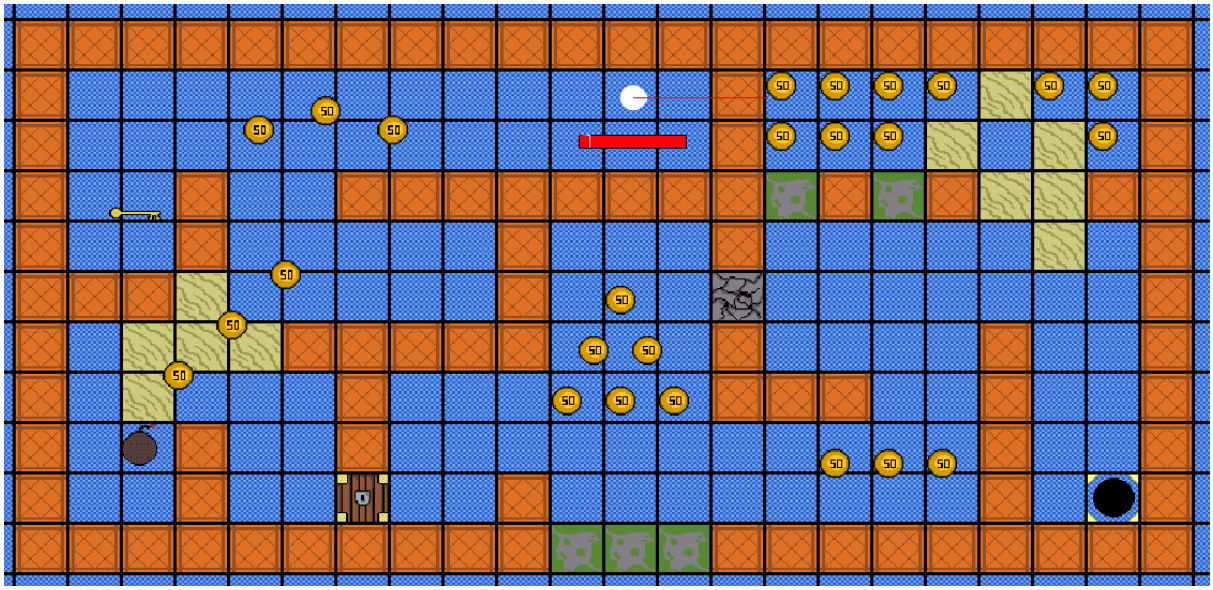

Fig. 6.9 - example of how a level is represented in game using the file read in

6.2.3 Media

Due to this project utilizing SFML to create an interface for visual and audio assets, two classes were created to handle loading in the media. These classes consist of TextureHolder.cpp and SoundManager.cpp, as well as their respective header files. The shared purpose of both these classes is to utilize their respective get functions to fetch the appropriate media file. In the case of the TextureHolder, it works in conjunction with LevelManager to determine the size of the level and to assign appropriate tiles to values which represent walls. As there were 10 different textures in the sprite sheet with game tiles, each was assigned to a different numerical value in the .txt files.

```
5     TextureHolder* TextureHolder::m_s_Instexture = nullptr;
6
7     TextureHolder::TextureHolder()
8     {
9         assert(m_s_Instexture == nullptr);
10        m_s_Instexture = this;
11    }
12
13    sf::Texture& TextureHolder::GetTexture(string const& filename)
14    {
15        auto& mem = m_s_Instexture->m_Textures;
16
17        auto keyValuePair = mem.find(filename);
18
19        if (keyValuePair != mem.end())
20        {
21
22            return keyValuePair->second;
23
24        }
25        else {
26
27            auto& texture = mem[filename];
28
29            texture.loadFromFile(filename);
30
31            return texture;
32        }
33    }
```

Fig. 6.10- TextureHolder class functionality

The SoundManager works similarly, although with extra steps. The process required the setup of SoundBuffer and sound objects, as well as respective function names that are called to play the sound in the correct situation. This was done in the SoundManager.h, (*Fig. 6.11)* whereas in the class file, the sounds were associated with their appropriate buffers, after loading them in directly from the .wav file.

```
    SoundBuffer gopherHitBuffer;
    Sound gopherHitSound;

    int next_sound = 1;

public:

    static SoundManager* getInstance();

    void playGopherHit();
```

Fig. 6.11 - declarations in SoundManager.h

Finally, there was a function created for each sound to enable them to be played (*Fig. 6.12)*. The function was then assigned to be called in appropriate sections of the code, such as for the ***ballknock.wav*** to play whenever the ball collides with any type of wall.

```
    gopherHitBuffer.loadFromFile("audio/gopher_squeak.wav");
    gopherHitSound.setBuffer(gopherHitBuffer);
}

void SoundManager::playGopherHit()
{
    gopherHitSound.setRelativeToListener(true);
    wallKnockSound.setPitch((rand() % 2) + 1);
    gopherHitSound.play();
}
```

Fig. 6.12 - sound settings in SoundManager.cpp

6.2.4 Object spawning

The objects are spawned whenever the level is loaded. depending on what level the player is on will determine what walls, powerups and enemies are spawned. every object other than static objects like basic walls, sand and the sticky wall all other objects are spawned by calling their constructor and creating them as an object. Each object when spawned is placed into a list where it can easily be accessed together and called in a loop for easy access.

```
33  void Engine::loadObjects() {
34
35      //Clears the powerup list
36      powerupList.clear();
37      wallList.clear();
38      GopherList.clear();
39      Powerup* PUP;
40      Wall* wall;
41      BreakableWall* BWall;
42      Gopher* pGopher;
43      Windmill* Wmill;
44
45      //
46      int iCurrentLevel = m_LM.getCurrentLevel();
47
48      switch (iCurrentLevel)
49      {
50      case 1: //1st Level
51          //powerup spawn
52          PUP = new Powerup(530, 600, "Powerup", Coin);
53          powerupList.push_back(PUP);
54          PUP = new Powerup(610, 600, "Powerup", Coin);
55          powerupList.push_back(PUP);
56          PUP = new Powerup(570, 560, "Powerup", Coin);
57          powerupList.push_back(PUP);
58          PUP = new Powerup(570, 640, "Powerup", Coin);
59          powerupList.push_back(PUP);
60          PUP = new Powerup(570, 600, "Powerup", Key);
61          powerupList.push_back(PUP);
```

Fig. 6.13 - object spawning within LoadLevel.cp

Each object is spawned using their corresponding constructor. Every single object has some parameters in common like their position. Their position is separated in the x and y coordinates

which determines where their position is on the screen. The powerups and the gopher both have an enum that can be passed in to determine their type (if the gopher moves up and down or left and right and the type of powerup spawned). Finally, the gopher has a speed parameter to determine how fast the gopher moves on the screen.

```
6   Wall::Wall(int iXPos, int iYPos, string sSpriteName)
7   {
8       //Constructor for the Wall
9       fCentre.x = iXPos;
10      fCentre.y = iYPos;
11      sSprite = sSpriteName;
12
13      fCentre.x = iXPos;
14      fCentre.y = iYPos;
15
16      m_Sprite.setPosition(fCentre);
17  }
```

Fig. 6.14 - Wall Creation in Wall.cpp

6.2.5 Engine

The engine class is what binds the functionality together in Mini-golf Dungeon. It takes care of calling the code for loading the level, the collision detection between the player and the walls and the gopher and the walls, drawing the game to the screen and taking in all the inputs that the user and updating each of the game objects.

The engine class is where the main game loop takes place on initialisation. The class creates the window and set's the game's state to be on the main menu. Once the window is opened and set on the main menu the main game loop starts. The game loop continues to go until the window closes and each frame the engine first keeps track of the time and the state of the game is playing. If the player is playing the game the update function is called which updates all the objects in the level and loads the correct level depending on what the user inputted and updates all those items as well. Next, the engine draws all objects to the screen. Depending on the state of the game this can be as simple as drawing the menu screen to drawing the entire game level. Finally, the last thing that happens in the game loop is the checks for the collision of all of the objects in the scene. Since all of the objects are in lists there is a for loop looping through all of the walls,

powerups and enemies calling all of their collision detection functions to determine what will happen to them.

```
37  void Engine::run()
38  {
39
40      while (m_Window.isOpen())
41      {
42
43          //m_Window.draw(menu_splash);
44          dt = clock.restart();
45
46          // Update the total game time
47          m_GameTimeTotal += dt;
48
49          // Make a decimal fraction from the delta time
50          float dtAsSeconds = dt.asSeconds();
51
52          //Determin if game is Playing
53          if (State == state::PLAYING)
54              update(dt);
55
56          //Draws all objects to the screen
57          draw();
58
59          //Detects the collision of all objects
60          checkAllCollisions();
61
62      }
63  }
```

Fig. 6.15 - run() function in the engine class

The engine class isn't only for functions but holds all of the classes and variables together from the player, powerups, enemies, walls, different views for the game, sprites, textures, and multiple different variables and lists of objects that are used to keep track of the state of the game. The engine class keeps track of all of this information with the exception of what the classes have. The engine class manages all this information and calls their corresponding functions in order to make the game run.

7 Critical analysis

The overall result of the assignment is a game that fulfilled the necessary requirements set out in the project brief. This section of the document will go over each aspect of the game in detail, discussing the end results of the project and a detailed description of each feature.

○ 7.1 Requirements

As stated in the team's Technical Proposal, the aim was to create a game that combines two separate genres: mini-golf and a dungeon crawler. Thus, the idea behind *Mini-golf Dungeon* was to be a game where the player controls a golf ball, being able to choose its accuracy and power, in order to traverse a series of mazes, while navigating through obstacles and collecting power-ups. Each level will contain a hole that will bring the player to the next stage when collided with.

○ 7.2 Game Progression

The game consists of four different levels, each one showing signs of progressions via maze layouts that become more intricate as the player goes further, as well as additional obstacles that appear over the course of the game. These obstacles include windmills, sandpits, sticky walls, doors that require a power-up to break/open, as well as malicious gophers that move back and forth at certain areas. The player can move around using a system of choosing strength and accuracy to determine the ball's travel path. Each level gives the player an opportunity to collect points in form of coins, as well as a point bonus given at the end of a level, depending on how many shots were taken to traverse it; the lower the shot count, the higher the score bonus will be. Originally, there was a plan to include a fifth level, however It was scrapped as the existing for stages of the game were more than challenging enough and took up an appropriate amount of time to play from start to finish. Another feature that was scrapped, was the possibility of losing the game should the player collide with an obstacle upon losing all of their points. There are two main reasons for this:

1. There is no way to lose a game of mini golf.

2. The player will get quite a lot of points during gameplay, and even if they were to collide frequently with obstacles, it would take quite a while to lose all of them.

As such, the player can collect as many points as they wish, without fear of possibly losing the game.

○ 7.3 Graphics

Visually, the game has been given a retro pixel style, to make it reminiscent of old school games that would have been playable on older consoles, such as the SNES. Each level shows a variation in colour and patterns shown on walls in the maze, to give a distinct feeling of progression. An example of this being the contrast between the first and final levels: the former is reminiscent of a hedge maze, whilst the latter has the feeling of an actual dungeon, reminiscent of the name of the project. Throughout the development process the art style went through two different iterations: a test version that simply consisted of coloured tiles, and the finished versions that introduced patterns to tiles and objects to make them stand out more. While the graphical assets were not a focus of the project, they did have a profound effect on the appearance, making the

○ 7.4 Technical Issues

The creation of the project came with its own challenges and difficulties, which were discussed and tackled on a regular basis via team meetings and implementation periods. While most issues were of a small, technical nature, the two major ones came in form of connecting a database to the project, as well as having to change the structure of classes. The latter issue came at an early stage of the development process. The team was inspired to use a class layout found in a SFML tutorial on how to create a 2D game, however the team soon realized that it is not an efficient example. Thus, time was taken to discuss the issue, even concluding that the class structure must be changed. This task was completed over two days, where new classes were made to accommodate future features, with the code being redistributed among them. This process caused a minor setback with the project, as time had to be taken to reimplement certain features in accordance with the new class structure. Another issue that occurred was a problem with collision that emerged during a late stage of testing, where some of the walls would be ignored by the player sprite as it went through them. The exact issue has been described in detail in the Testing section of this document.

Due to having an additional group member compared to others, the team was asked to implement an extra feature. It was decided that this would be present in the form of a database that stores the info of users who play and finish the game. While the actual database itself has been made, there have been numerous errors with the SQL and Visual Studio connection. Work was ceased on the database when it turned out that the queries would have to be done manually outside of the game, rather than when the player is still in game. As such, the feature was dropped due to time constraints, but some traces of the functionality were left in the project files.

There were several other, small issues and bugs that came up during the development process. On occasion, the windmill obstacles malfunction and seemingly drag the player along, rather than knock them away. Sticky walls in level 2 do not actually cause the player to get stuck to them. The function to press enter to pause the game causes the program to blink rapidly between the pause screen and game screen, followed by a crash. Finally, one of the finals issues to be spotted, was an occasional occurrence related to obstacles that have an associated power-up. During playtesting, the ball was shot into a door without having picked up the key. As a result, the ball glitched inside the door and further attempts to shoot it at full power resulted in it violently bouncing from side to side, occasionally passing the door boundary but not being able to free itself.

- **7.5 Time Management**

Given that the project had a timeline from the start of September till February 4th, there was a need for setting up a schedule that specifies dates for task delivery. The team ended up creating a Gantt chart to create said timeline, assigning periods of time for each major component of the project to be finished. Additionally, Trello was also used in management of the project, by allowing each team member to quickly check/edit current tasks or their deadlines if necessary, and to add new sprints on a weekly basis. Having a means to keep track of the project's timeline proved to be highly helpful, as it kept everyone motivated and aware of deadlines. The development cycle was kept well within the allotted time frame, as all the major parts of the project were essentially finished several days before the due date. This accounts for any setbacks that occurred along the way, such as due to the class rework, or the Christmas break, where most

of the group was mostly unavailable due to work or personal reasons. Considering all setbacks within the project's timeline, it is safe to say they had little effect on the overall outcome and success, as the coding and testing process was finished well before the official deadline.

○ 7.6 Summary

Overall, in contrast between the specifications laid out in the design stage of the project, and the finished result, the team managed to complete most requirements and criteria that were set out. About 90% of the desired functionality was achieved, with the main exception being the SQL database

8 Conclusions

The project has been completed successfully, with most features specified in the Technical Document being included over the development process. Thanks to the combined effort of all team members, it was possible to achieve all demands set out in the assignment. The time which the team was given to complete this task was utilized as efficiently as possible, which resulted in the implementation of nearly all the desired features.

Even though the project came with its own difficulties, most of them were successfully overcome via means of regular testing, team meetings and implementing the necessary changes. Very few issues have been left unresolved, with the most prominent one being the database being left unimplemented due to technical errors.

9 References

1. Battaglia, P. and Spelke, E. and Tenenbaum, J. and Ullman, T. (2017) 'Mind Games: Game Engines as an Architecture for Intuitive Physics', *Trends in Cognitive Sciences*, volume 21 p649-665. [online]. Available at: https://www-sciencedirect-com.ezproxy.lit.ie/science/article/pii/S1364661317301134?via%3Dihub (Accessed: 30 October 2019)

2. Carlo, F. (2007) 'Gameplay and Game Mechanic Design: A Key To Quality In Videogames Gameplay and Game Mechanics Design', [online]. Available at: http://www.oecd.org/education/ceri/39414829.pdf (Accessed: 30 October 2019)

3. Cubal J., Reyno E. (2008) *Model Driven Game Development 2D Platform Game Prototyping*. Conference Paper. Available at: https://www.researchgate.net/publication/221024315_Model_Driven_Game_Development_2D_Platform_Game_Prototyping (Accessed: 28 October 2019)

4. Engstrom, H. and Brusk, J. and Erlandsson, P. (2018) 'Prototyping tools for game writers', *The Computer Games Journal*. volume 7, p153-172. [online]. Available at: https://link.springer.com/content/pdf/10.1007%2Fs40869-018-0062-y.pdf (Accessed: 29 October 2019)

10 Ge, X. and Lee, J.H. and Renz, J. and Zhang, P. (2016) 'Hole in One: Using Qualitative Reasoning for Solving Hard Physical Puzzle Problems', *Frontiers in Artificial Intelligence and Applications*, volume 285, p1762-1763. [online]. Available at: https://openresearch-repository.anu.edu.au/bitstream/1885/154106/4/02_Ge_Hole_in_One%253A_Using_Qualitative_2016.pdf (Accessed: 28 October 2019)

11 Goszczynski, L. and Krahulik, C. (2014) *Alpha, Beta, Launch: A Newbie's Guide to Educational Video Game Development* [online]. Available at: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1007&context=jih

12 Henderson, J. (2006) *The Paper Chase: Saving Money via Prototyping* [online]. Available at: https://www.gamasutra.com/view/feature/131099/the_paper_chase_saving_money_via_.php (Accessed: 30 October 2019)

13 Kelly, C. (2012) *Programming 2D games.* 1st edn. Boca Raton, FL: CRC Press.

14 Keo, M. (2017) 'Graphical Art Style in Video Games', Bachelor's Thesis, Hame University of Applied Science, Riihimäki. [online]. Available at: https://www.theseus.fi/bitstream/handle/10024/133067/Keo_Mary.pdf?sequence=1&isAllowed=y

15 Lange-Nielsen, F. (2011) The Power-up Experience: A Study of Power-ups in Games and Their Effect on Player Experience. DiGRA Conference 2011. [online]. Available at: https://www.semanticscholar.org/paper/The-Power-up-Experience%3A-A-study-of-Power-ups-in-on-Lange-Nielsen/26e0cd1d53d3dd44c48b05bfb6aebcb7b59067de#citing-papers (Accessed 30 October 2019)

16 O'Donnell, C. (2008) 'Defining Game Mechanics', *The International Journal of Computer Game Research*. volume 8 (2). [online]. Available at: http://www.caseyodonnell.org/files/TC839/Defining_Game_Mechanics.pdf (Accessed: 28 October 2019)

17 Rogers, S. (2010) *Level Up! The Guide to Great Video Game Design*. 1st ed. The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom: John Wiley & sons, Ltd.

18 Schiffler, A. (2012) *New Game Physics: Added Value for Transdisciplinary Teams*. Doctorate Thesis. University of Plymouth. Available at: https://core.ac.uk/download/pdf/29816994.pdf (Accessed: 30 October 2019)

19 Solinski, C (2012) *Drawing Basics and Video Game Art* [online]. Available at: https://books.google.ie/books?hl=en&lr=&id=7FQiT76ECTMC&oi=fnd&pg=PA10&dq=how+to+make+video+game+art&ots=O9gXJOGU1c&sig=wP4NDplu0JqmxjnZI_ITpPw1RVg&redir_esc=y#v=onepage&q=how%20to%20make%20video%20game%20art&f=false (Accessed: 29 October 2019)

20 Totten, C. (2014) *Architectural Approach to Level Design*. 1st edn. New York: CRC Press. [online]. Available at: https://www.taylorfrancis.com/books/9781315270753

21 Appendices

21.1 Student 1 Reflection – Konrad Winkel

For the past few months, I worked in a group consisting of other students, some of whom were part of a foreign exchange between Ireland and Canada. Together, we designed, implemented, tested and released a unique maze game, titled *Mini-Golf Dungeon,* following the specifications required by the college.

Working as a team proved to be quite successful and enjoyable, as everyone involved in the team project shared the drive to put in their best effort to make the game to the highest standard we possibly could. The Canadian students, Zach and Rejean, were particularly fantastic to work with, due to having a much more advanced knowledge regarding C++ and game design. On the other hand, myself and James had more experience with testing and object-oriented programming. Thanks to the two vastly different backgrounds, we were able to combine the best of two worlds. During the development process, we got on very well as a team, often meeting up to discuss issues related to the project as well as tasks that we are currently working on. Throughout the development period of September – February, while may not have been a professional team, I still believe we performed well, and the group dynamic was a very strong one among us. The game, even though it was not perfect, was a result of hard work and learning that all four of us can be proud of to have achieved.

My contribution to the team consists of being the team leader, as well as working on several code-based features of the game, implementing the art style and regularly checking for errors in both the code and documentation. I put myself forward as team leader, as I felt confident that I would manage the project's timeline and tasks efficiently, and with the hope of gaining more experience in project management. I ensured that we had weekly meetings where we could discuss problems and the next steps to be taken with the project, giving everyone a chance to discuss any emergent issues or concerns related to the assignment. Every team meeting was recorded and posted on Google Drive to remind anyone of the discussions we had at any stage. Thanks to Rejean's idea, we also used Trello to manage tasks and sprints, which I ended up utilizing frequently to set deadlines for various elements of the project.

Having some previous experience in art and design, I offered to create an art style for the game that would give it a unique feel for anyone who would play it, particularly those who appreciate retro-style games. I used Aseprite, which is a graphics software dedicated to pixel art, to create each asset that is used in the game, including wall and floor tiles, pickups and obstacles. As expected from a programming-heavy assignment, creation of the graphic assets was not a priority, and I only set aside a minimal amount of time to make them. Regardless, I'm quite proud of the result and how every art asset came together, and I'm sure other people will appreciate it too.

Regarding code practices, I created the initial class structure that was used for our project proposal. As time went on, we decided as a team that we need to restructure the code to more efficient classes, as the version we were using had several issues. Later, it turned out the class structured required a rework, so I worked with James to restructure the classes into a system that works more efficiently and is more cohesive. I was also responsible for creating the SoundManager classes, which brought an audio element to the game, to provide more depth to the players' experience of it. All the sounds were used on Freesound.org, an open source audio sharing platform. I worked in collaboration with Zach on the collision to ensure the sounds were playing correctly. It was also with Zach's suggestion that we made a singleton for the SoundManager, which meant the object can be references in any class without passing in a SoundManager object in certain functions as a parameter.

My two most extensive duties involved creating the levels and testing them rigorously. I had a hand in pretty much any visual aspect of the project, such as helping James set up the render window, textures and HUD placement. As outlined in the Testing section of the document, there was a significant amount of time spent replaying each level to make sure the wall tiles and obstacles were placed to maximize the player's engagement with the game without punishing them too much. The goal was to create a fun maze game, not one where the player must immediately navigate tight spaces, approx. 40x40 px, at the very start of the game. When the layout of each level was finished, I added in obstacles and power-ups as separate objects in the LoadLevel class. Being knowledgeable about the level layouts, I was able to help the team when

it came to test for collision and object placement. Aside from that, I helped wherever I could if any of the guys needed me. This includes testing new obstacles and powerups whenever Zach made them, and helping Rejean with collision testing, for which I always included feedback, whether it worked well or not.

Finally, I was also in charge of making sure that each document related to the game's design and implementation process was edited and formatted to an acceptable standard. Especially during the final days/hours before the deadlines, I would regularly check in with the other guys to see how their work is coming along and guide them where necessary. Final edits on a document generally took me about one to 4 hours, depending on the size of the file. It gave me a chance to go through any grammar or punctuation errors, as well as edit some of the text that may have seemed out of place or needed a little extra work.

Unfortunately, I've had certain difficulties with C++ since the start of the year: getting the hang of it came at a slow pace for me, which has affected my ability to bring my best in this project. Granted, I have performed coding tasks as I have outlined above, but there is room for improvement regarding my ability to utilize C++.

What I learned in technical aspects)

The project was a fantastic learning opportunity for me, as I was required to heavily research C++ implementation techniques, I order to be able to contribute to the project. I was able to learn substantial amounts of coding practices from Zach and Rejean, thanks to their knowledge of the language, which was much more advanced in comparison to mine. This includes but is not limited to the player movement algorithm and the collision system that was used in the game.

Having studied the *Thomas Was Late* tutorial on LinkedIn Learning, I acquired the knowledge of loading in levels via reading numbers from a text file. As I enjoy making 2D games, this will certainly prove to be helpful in the future, when it comes to developing my skills as a game designer. The code that came along with the tutorial also helped me gain a better understanding of C++, as my only previous experience with it was back in first year. As such, when we first got the project, I had my fears about whether or not I would have any sort of impact on the code,

based on my inexperience. This however, disappeared over time, as I managed to improve my programming ability as time went by, either by implementing features myself, or discussing code fragments with other team members who were responsible for implementing them.

What I learned from the teamwork)

Thanks to this project, I had my first real taste of working together with other people on a long-term IT project. Additionally, I had a chance to show initiative to lead the team during the development process, which is something I would generally struggle with. As time went by, I began to feel more comfortable and confident in my position, regularly checking with the other team members to ensure they're doing the correct tasks and if they have any issues with them. Having to keep communicating constantly about tasks and issues helped improve my organisational skills, as due to leading the team, I was expected to prioritize certain objectives which required immediate attention.

I was also able to learn more about the theoretical aspect of teamwork. During our time in class, we were introduced to aspects such as the team performance curve, how to manage a project efficiently, and how to deal with conflict should it ever emerge in the group. The latter was thankfully not an issue, as there were never any arguments or fights within the team. Most of the issues were either design or code related, which were generally dealt with swiftly and within a good time frame.

Conclusion

Overall, I believe my contribution to the team was as significant as everyone else's that I worked with. Thanks to everyone's input and hard work, the project that came as a result of it is a well-functioning game that is also enjoyable and unique of aspects of gameplay and art. Granted, there could have been a few things done differently, but I am proud of the work I put and the result of everyone's time and effort that went into this assignment. It may not be perfect and include everything we wanted to do but is still a valiant result of our combined work effort, thought process and dedication to study and make video games.

21.2 Student 2 Reflection – Zachary Ayotte

For this project there are a couple of aspects of the game that I've worked on. I've worked on the interaction with the environment, the powerups that the player collects and their interaction and the creation of enemies and some other little things.

In the planning phase of our project I was tasked to research the interaction between the player and their environment and, I made a simple prototype in Unity which as simply just how the maze would look and the camera perspective and the.

In the beginning of our project my task was to research different ways that the user would be able to interact with different things in the scene. The main aspects of research that I was focused on was on how the player interacted with the environment and objects within the scene. for the environment the first method that we attempted to do was to have every single wall be part of the scene and be spawned in when the level will be loaded. In this method we can see if the ball is interacting with the specific wall or ground hazard and the ball would react accordingly. Although this was a great idea there was some problems that we ran into. The method we were using to spawn in the level was from a text file and spawning the specific objects according to what number was there in the next file. The problem is that while using this method did make creating the levels from a text file easier it was extremely difficult to set the location of each object which was required to spawn in the objects. This caused us to make a choice of what we had to keep. Either the reading from the text file or spawning in the objects individually. I had an idea about how we would be able to have the ball interacts with the specific sprites in the method that we needed it but that made it so the wall classes that were created to be not usable since instead of spawning in each of the objects and having the player interact with it we determine the bounds of the sprite that we were displaying and adjusted the ball accordingly to what we needed it to do. Inspired by the Thomas was Late code and the code we had for the collision with the objects I combined them both together to make the interaction just the way we wanted it to be. Although, the code wasn't perfect it was improved throughout the time we worked on it and it's finally colliding very well. Finally, the code for the walls were used elsewhere in the game for the special interactable walls like the door and the breakable walls since these objects would not

be able to be destroyed if they were a sprite and would be used easier if we were to spawn those walls instead because there are less of them so we can hard code them for the level layout that we needed .

Another aspect of the game that I worked on was the aspect of pickups for the player to interact with. The method used for the powerups were very similar to the first idea we had for spawning in the walls. The powerups were all a single class called powerups. To spawn the powerup there were specific parameters that were required to be passed in in order for them to be spawned the required items were their position in the game world and what type of powerup they are. These parameters were passed in and depending on which one of the powerups were created their sprite and functionality were added to them. Next we go to their collision with the player. Since the collision with the player and the environment is completely different there needed to be a check for the collision between each of the powerups and the player character. In this specific case whenever the player runs into the powerup the powerup disappears and gives the player one of 4 different powers. The first power that was created was the key and the bomb powerup. Then the next powerup was a coming powerup which increments the player's score by a specific amount and finally the last powerup that was created was the shield powerup this was the most difficult powerup to implement because of all the interactions that the shield has. The shield powerup had to be known in multiple different locations. Since the shield interacts with the Windmill, Gopher enemy, sticky walls all these different interactions happen in different sections of the code so the player needed to be passed in a lot to know if they held the value hasShield to make sure that they interacted correctly. Also, each object interacts differently with the object all of them make it so they don't lose points if they are hit but the sticky wall reacts like a normal wall with the powerup. The gopher will go down in its hole like you hit him but you won't lose the points for hitting it and the windmill as well but the windmill will still hit the player. This functionality of having one object do multiple things were difficult because there are multiple things that one object can do and this can cause some problems while coding since all of the functionality is spread across multiple files making it hard to keep track of where some of the code is for this specific powerup. Also, while making this reflection is how bad the way these powerups were made since they are all the same powerup and some of the functions aren't even used whenever the powerup is created making some of the code useless depending on what item

it is. The best way to have solved this problem is to have created subclasses for each of the powerups and making it easier to manage.

The final aspect of the game that I worked on was creating all the enemy objects. Most of the enemies are spawned in most enemies also reused some of the aspects of the old wall since some of the enemies required some type of collision inside of them for the windmill enemy this enemy interacts with the player in two different ways. It knocks back the player but not like a wall it knocks them a bit further and also makes the player lose points when they hit the windmill. There is also a gopher enemy that i created. This enemy was the most difficult enemy to create because of what it interacts with. The gopher interacts with both the player and all the walls that are in the game. And the gopher is an object as well. Since the method of interacting with the player and interacting with the walls are two completely different methods requiring both methods to be checked. The gopher requires knowledge of the layout of the level since it is able to move back and forth between two walls in a vertical or horizontal manor. For this interaction, the method that was used is very similar to the method the ball interacts with the wall but instead of the ball it's the gopher. Once the gopher hits the wall it will turn around and continue to move. The interaction where the gopher hits the player is the same method of how the powerups are collected but the enemies do not despawn when hit. These gophers make the player lose some of their score and go into a hole making them untouchable for a couple of seconds. The interaction with the player was difficult to make since there are multiple different interactions that the gopher has with the player. The easiest part was that the player loses their score but when it came to how the gopher reacts if the player hits them with or without the powerup or even after the player hits them needed to be determined by if the player has the powerup and what state the gopher was in. other than the gopher most enemies were simple copy of one other class that we created with slight changed to make them different.

There are also sections of the project that aren't in my territory but i did help somewhat improve. One of these sections is the sound section instead of having the sound class be implemented everywhere around the code I helped make the class a singleton so that whoever needs to have the code can simply add the singleton to the class making it accessible everywhere in the code with ease.

The teamwork aspect of the project was very organised everyone put all their effort into what section we were doing and made sure to get it done as soon as possible the only aspect where our teamwork didn't work out to well is inside of the code. Since some aspects of the code were being worked on at the same time it was difficult to change something without making someone else redo or merge the two pieces of work together. Although, by the end of the project this was happening less and less because there were multiple and each of us were working on individual ones.

In Conclusion, this project that we work hard on ended up not where we wanted it to be since some of the features that we wanted didn't end up in the end result but i'm still proud of what the finale project ended up. It's a functional golf maze game that is fun to play. I am proud of my team for making this great game and putting all their effort into creating it.

21.3 Student 3 Reflection – James O'Neill

In the beginning of the group project, we all examined each other in the first couple weeks to see what state we were in and the skills that we have obtained over the course of the past years of our education. For myself and my teammate Konrad we both had reasonably the same education and skills given the fact that we both attend the same college, but for our Canadian teammates Zackary and Rejean they had a little more experience in the game programming from the start of their education.

In the weeks that followed, we had some creative ideas about what we expected the project was going to be like and the problems or complexities that would hinder our progress. Mainly, we were on the same page with each other and we had some ideas for the game, but not all of us could agree with them, but in the end we had to complete the decision making process as soon as possible, so we had to make some compromises along the way.

With the implementation of the project and the team performance over the course of a couple months towards December, I think that the progress we have made over the course of the project module has been great and very productive. For the sprints, we had our meetings on a regular basis and decided the tasks at hand each of us had to complete. For our meetings and supervisor meetings, they were quite short if I am going to be honest with the expectation of the supervisor meetings. Generally, we have most of the tasks brought up and discussed on our Discord group chat. When it comes to our face to face meetings, there are very quick and just a generalised overview on the tasks ahead for this week. The problem that I have with this is the way we communicate online and then we know what to do, but when we have a meeting it's not very interactive and then it is up to presumption that we all know the tasks that are at hand. The timing of how long the meetings where was very short since we had the meeting mostly done online. What I think that we needed to do more of this to brainstorm more ideas during our design phase, because we kind of came up with new ideas in the game every week. For the management of the project we used the application Trello and this allowed for the project to manage more easily and has each week lasted out.

Our team performance overall is really good and I think where I see our position on the curve is around the potential team and real team but I think that there was some fluctuations in the course of a few months where it would hit somewhere down near the pseudo team. Mostly this was brought on by the problems or ideas not going to plan which hinder the progress being made when it came to our coding week in the project, that was where the team had as a group excelled a past our abilities and had a reasonably functional game. Our commitment to use the applications that we used for project management was very sceptical since the only thing I see working is our GitHub repository or the Google drive, but the Trello board is completely gone dead from the last month and a half. From my point of view I think that I would rate the rest of the team around where I am and I think that we all have put effort into creating something that we like to have presented at the end of the module deadline. Each member of the team put work into getting tasks done but sometimes there might be some confusion over time roles that each of us had.

Since the Christmas period our plan was to have a communication plan setup with the Canadians over the Christmas to work on the project. Ever since the plan was in place there was an issue with time zone differences as one country's time might be very early in the morning or late at night, so there was much we could do about it everyone on different schedules. In terms of the plan to have some work done over the course of the Christmas period it is very minimal since each of us had increasing difficulty finding time to do work since our personal lives, family and other issues had stepped in our way. The second coding week we had didn't have much of an impact on the project since some of us were experiencing travel sickness, so that week kind had a significant impact towards the project progress, but in my case, in the interest of the team, I was willing to continue and have something done for that week.

Generally, I think we are some form of a team but probably not the best but these errors can be simply fixed and there is always room for improvements to ensure that the team performance can grow. Over the course of the project there were some issues that have arisen from the progress being made. Some of the issues that we faced caused us to have concerns about how the project would, but I think that it went ok in the end, but not to the certain standard we chose to have it at.

22

My own individual contribution to the project was on my behalf but because the other members of the team put more of a performance in the game rather than I did. What I am trying to say is that I think that where a problem occurs, I think that it affects my ability and belief in fixing the problem. From the group, I don't think I am the strongest programmer out of the groups, and I think that I was putting in my full potential to get the project completed.

23

I was tasked with the design phase of the assignment and what the structure of the project was going to be like. There were some problems in which the project layout would be like. From the LinkedIn learning courses, I spent the entire first month in finding what sort of layout to do, but in the end, the Thomas Was Late game, because there was a better structural layout of the game and this worked better with the game we created. I think that we can use the tutorial as a base ground to build up the project on and there was a better structure of inheritance for the game.

The other tasks I did for the game was to setup the engine class, handle the user inputs to the game or do some of the drawing to the screen in the draw class. The windows or the types of screens that were contained in the splash screen commands which allows the player to play the game, view the instructions, rules or exit the game. From this, I initially set up the windows for the team to use. What I think that I did the best work at was the Hud class and that it was something I could not mess up that much, but at the moment, the HUD is looking ok and I think I proved that I did something right. Primarily, I spent most of the time doing the database and research on the use of the Azure database, but also from time and time again, I went through the code making minor changes where necessary or adding in some beneficial code for the game. I did help out a teammate in the creation of the levels for some time, but when on then to do other various tasks through the game, mostly with the game states and getting the meu to run or have some functionality in the menu.

My primary task and main responsibility that I was tasked with was creating the database connectivity with the game. I had high hopes for this one and the biggest problem I faced throughout the last month of the game was to get the database to do two simple things that I thought it would be easy to do. Finding out the correct method to connect the database to the

c++ application was tricky to get done and it proved to be the worst thing I could have done. I was using the ODBC connector to do the connection from c++ application to the Azure portal. In the end, I could not get it done and there was no time left to get the backup plan to have the data going into a file and the controller connected to the game. If i did have more time to figure out the database connectivity, I think the game would have been in a reasonably higher position of completion, probably not completed to our expectations, but at further than it is now.

Overall, I think that I did ok for this project and it is standing in a reasonable position, but quite not the one we expected it to be. The primary functionality of the game works and as it stands, we are happy with it. From myself, I think I let the team down with my contribution to the game and it would be better if I was better at programming, but I am generally pleased with everything and everyone, but I wished that it could have done better on my side. With everyone who was on the team I was happy to be around them, but at certain times, i think that i was probably hard to be around, but i think that we have pulled together as a team to get the project done.

21.1 Student 4 Reflection – Rejean Lagasse

At the beginning of the school year our team decided on each member's workload. Because of my understanding of math and geometry, it was decided that I would oversee collision detection and player movement. As such, the Player object comprises the bulk of my contributions to the project. This includes the files Player.h, Player.cpp, BallParticles.cpp, and Collision.cpp. I also spent a lot of time debugging these systems.

i. Creating the Physics

My main goal in developing the collision system was to create a consistent physics. Minigolf Dungeon is heavily reliant on players being able adjust the power and angle of their shots to avoid obstacles and reach the goal quickly, so if a player was unable to foresee how the ball will move it would undermine the entire experience.

I started by developing the systems to shoot the ball with the space key and rotate with the arrow keys. After these were implemented, I added the ability to bounce the ball off walls. Next, I added the timed button press that determined the ball's power and accuracy. Once the core movement was in place, the rest was mostly a matter of refinement and player feedback. I added different types of terrain and walls that would influence the balls movement such as sandpits. Using stock SFML/OpenGL shape creation, I implemented the power meter and accuracy line seen in the final game. I also implemented the ability to use pickups as well as the particle effects that display when a pickup is obtained.

These systems worked well, but I did make a few mistakes. The ball would occasionally stop entirely, no matter what its speed was in the previous frame. To find the cause I used a black box testing method to narrow down the possibilities. I had the program output the ball's variables to the console while it was in motion and would repeatedly shoot the ball until the bug occurred. I would write down the balls last known properties, and eventually a pattern emerged. The ball would stop when its x and y speed values were within 10 frames per second of each other.

I was using the fabs function to find the absolute value of x and y and using their combined value to determine when the ball should stop. I determined that fabs was not working properly and switched it out to four if statements that covered all possible x and y speed values. The ball performed as intended after this.

While debugging I discovered another issue with the collision system very close to the end of the project. Minigolf Dungeon is tile base, and I noticed that when the ball collided with where the tiles intersected it would sometimes get stuck or bounce in an unexpected angle.

I set up a console output that would tell me the ball's position when it collided with a wall, and my suspicions were confirmed. Because our collision system checked from the top-left tile to the bottom-right, if the ball hit two different walls at the same time it would prioritize the top-left most one. This caused the ball to bounce erratically. The bug was a logical error, so the game was still playable, but it undermined a crucial aspect of gameplay, so I decided that the collision detection system needed to be retooled.

After analysing the problem, I developed a system where collisions were detected, and only once the program had gathered all the data would it handle the collision events. I then wrote a function that would check if one of the ball's cardinal direction hitboxes (top, left, right, bottom) was hit and prioritize those hitboxes over the diagonal ones. This removed the problem where the ball would interact with multiple walls and created a vastly more realistic and consistent physics system.

ii. Creating the Tutorial

I also created the tutorial for the game. The final version of the tutorial is a slideshow wherein the golf ball tells the player in a jovial manner how to play the game. I worried that the golf ball was so friendly and helpful that players would be hesitant to hit them in the actual game, so I made sure to explain that golf balls don't mind being hit with golf clubs.

Initially, I wanted the tutorial to be a video, so that the filling in and depleting of the power/accuracy bar would be easily read by the player. However, SFML offers no support for such functionality on its own, so I attempted to use the library sfeMovie (http://sfemovie.yalir.org/latest/start.php). Sadly, there is not much of a community around this library, so I had trouble finding resources on how to properly install and configure it. I tried several times to get the library working, but to no avail.

As a backup, I created a slideshow. In the tutorial class, I store each tutorial picture in a queue. Then, in the Tutorial:Update function I pass the front of the queue to the draw function. Update keeps track of the time and will pop the front slide off the queue every 3.5 seconds, at which

point a new image is drawn to the screen. I also use the soundmanager singleton to play a free use for non-commercial products from Windows 10 slideshows.

iii. Team Management

The team met every Tuesday to discuss what each individual member had accomplished and what would be done in the next sprint. This was initially based on the Gannt chart created at the beginning of the semester, but the team quickly found out that we had frontloaded the chart with the more difficult tasks, so adjustments were made on a weekly basis. These meetings were typically 15 to 30 minutes long, and points talked about were written down, and later put on Google Drive or Discord by Konrad.

My role in planning would be to create a Trello task board that broke these topics down into tasks. These tasks had to be doable in the given time and outlined into steps so there was no miscommunication. If anyone had any concerns about the tasks, they could make a comment on the virtual kanban, or change the steps to their preference and tell the group through some other means. This ensured that each team member knew what the others were doing, and guaranteed that if anyone forgot their task, they could easily look it up.

Despite these precautions, there were occasional missteps. Early in the project the team decided to create levels using a .txt document. This sped up level creation and testing; however, the implementation of the obstacles no longer worked with the collision system I had developed. The team member who designed the level creation, and, everyone else on the team, failed to look at the program holistically. Oversights such as this subsided entirely as the core features were added to the game and the team became more familiar with the codebase, but it remains an important lesson for future projects.

There is also the matter of how teams were formed. Due to every person in our class taking the same course and having similar interests, most groups were homogenous. While our team was not exempt from this, we did have the unique advantage of having a mixture of Canadian and Irish students as our members. Despite being from different countries, our education and backgrounds were largely similar, except that the Canadian students were better versed in C++, and the Irish students better understood the language around planning and design. We used this to our advantage where we could, with the Canadian students taking on the jobs that required a more in-depth knowledge of SFML and the Irish students creating architecture diagrams.

iv. Document Deliverables

Our team split writing duties into what we felt were even amounts of work. For the final project report, I wrote the introduction, part of the implementation section for collision, part of the debugging section for the same, and edited the literature review.

I have a bachelors in English, so I am very confident in my writing and editing skills. Even so, I had some trouble revising the literature review for this final document. Many parts of it were done, I assume, under a strict time crunch, and as a result read like nonsense. It was a struggle putting the content into coherent paragraphs that flowed together and remained focused on their chosen topic. I probably put more time into editing the literature review than on any other section of the document. I also helped edit some other parts, though Konrad did more editing than myself.

v. What I Learned

During this course I superficially learned about topics such as the team performance curve. I am able to describe what the curve is and the basic idea behind it but am unable to go into any real depth on what exactly differentiates a pseudo-team from a potential team. I understand that in a workplace employee will specialize in different areas but given that students mostly have the same learning experience I have a difficult time understanding how it applies to our team.

I also have a difficult time differentiating some concepts in documentation. I've had it explained to me numerous times, but I still struggle with the difference between scope and deliverables.

Despite my tenuous grasp on team dynamics and documentation, I was pleased that this course gave me the opportunity to reinforce my understanding on C++, SFML, and working in a group. Because I dislike not being in control and having influence over the work of others in equal measure, it can be difficult for me to work on a team. It was nice to work on my interpersonal relationship skills.

I also learned that on the Belbin Team Roles my highest score is a plant. This means that I thrive on creative solutions to problems and embrace new ideas. My second highest however, is an Implementer, which means I am very practical and resistant to change. These may appear contradictory, but in hindsight they make sense. I enjoy coming up with new ideas, but dislike

when my plans are disrupted. I also tend to think that changes made to my ideas are for the worse. Finally, I am occasionally paralyzed by indecision. I will attempt to focus on one or the other in the future, though I do not actually know if personality is something that can be learned or is innate to one's self.

vi. Conclusion

I did a lot of good work on this project that was integral to the success of the team. I helped my team members when they needed it and accomplished nearly every task that I set out to do. There are a few areas where the game can be improved, but I am pleased with the product thus far, and I hope to continue refining Minigolf Dungeon for Games Fleadh.

24 User Manual

The player controls the ball in a dungeon-like maze. The player can rotate clockwise or counter clockwise, by pressing the right and left directional buttons respectively. To move the ball, the user must press the space bar twice: once to set the strength of the shot, the second time to determine the accuracy. Through the maze, the player must navigate their way through obstacles to reach the golf hole that will transport them to the next level. Along the way they will be able to pick up additional coins and power-ups to guide them through and help deal with obstacles. There is an in-game instruction video that explains these features with the help of visual guides.

The in-game menu can be navigated with number buttons 1, 2, 3, 4 and 0. Each number corresponds to a different function. 1 launches the game, 2 lets the player watch an instructions video, 3 shows the database, 4 quits the game and 0 goes to level select for the game, allowing the player to jump straight into any level of their choice.