5-9-2023

The Process of Using Unity to Create a 2D Video Game

Sean Tammelleo

Follow this and additional works at: https://vc.bridgew.edu/honors_proj

Part of the Computer Sciences Commons

Recommended Citation
Tammelleo, Sean. (2023). The Process of Using Unity to Create a 2D Video Game. In BSU Honors Program Theses and Projects. Item 635. Available at: https://vc.bridgew.edu/honors_proj/635
Copyright © 2023 Sean Tammelleo

This item is available as part of Virtual Commons, the open-access institutional repository of Bridgewater State University, Bridgewater, Massachusetts.
The Process of Using Unity to Create a 2D Video Game

Sean Tammelleo

Submitted in Partial Completion of the
Requirements for Commonwealth Honors in Computer Science

Bridgewater State University

May 9, 2023

Dr. John F. Santore, Thesis Advisor Date: 5/9/2023

Dr. Paul Kim, Committee Member Date: 4/27/2023

Prof. Sean Stanley, Committee Member Date: 5/4/2023
Introduction

Ever since I was a kid, I have always been fascinated with video games. I would play games any chance I could get, and it did not even matter what it was. Of course, I had my favorites throughout the years. It started out with simple games like Plants Vs Zombies (PopCap 2009) and Insaniquarium (PopCap 2001). Then as I got older, I moved on to games with more depth which allowed the player to complete longer, much richer, storylines. Sometimes these stories would even span multiple games which could take tens or hundreds of hours to fully explore the backstory and experience everything the developers had to offer. Games like Call of Duty (Activision Blizzard 2003-2022), Assassin’s Creed (Ubisoft 2007-2020), and Bioshock (Irrational Games 2007-2013) all created amazing worlds to explore with storyline details so minute that they completely immerse you into the universe the story takes place in. I loved being able to explore these other worlds and control my own person in these worlds. I loved playing with my family and friends and working together with them or working to try and beat them. And most importantly, I loved that the developers were able to create whatever they wanted and could even add in little secrets, or easter eggs, that showed that they worked on the game and contributed to making something great. From the very beginning, it was obvious to me that game development is what I wanted to do with my life.

By the time I started my journey at Bridgewater State University, my taste in games had broadened even more and the experiences I continued to have with them even further motivated me to want to be an actual game designer. However, it would not be an easy journey. Going into freshman year, I had extremely limited experience with programming and no experience at all in game design. As my college career continued, we had a couple of projects here and there that gave me a taste of what I was really after. A text-based Pokémon (The Pokémon Company 1996-
2023) game here, a recreation of Snake (Nokia 1997) and Hangman there. But nothing truly gave
me that feeling of actually designing my own game, it felt more like completing a school project
because really that’s all it was. There was no freedom to express creativity or to make my own
world like I had been chasing, just complete the assignment and get the A. During my last
semester at Bridgewater, however, I knew I wanted it to be different than before. The
culmination of all my learning was about to come together and even more learning was to come
as I had decided I wanted to do an honors thesis. And for this thesis, I was going to actually
create my own game and my own world as I had dreamed of my whole life. This game would
give me experience in game development before even making it to the industry and that would
challenge me to reevaluate what I thought I knew about games and what I thought I was capable
of.

The first thing I had to decide was what kind of game I wanted to make. Making the
choice not only what I thought would make the best game, but also what I would have the most
fun doing. Maybe a puzzle game that would challenge the player to think in unusual and creative
ways in order to win. Or perhaps a 3D first-person shooter that surrounds the player in their
environment and forces them to fight their way to victory. The only thing I knew is I wanted it to
be a challenge and to force the player to be good at the game in order to succeed. I did not want a
game that would be too easy, because it can feel extremely unengaging and boring. I was taking
the developer approach and mindset from immensely popular games such as Dark Souls
(FromSoftware 2011-2016), Elden Ring (FromSoftware 2022), and, most directly applicable to
my game, Super Meat Boy (Team Meat 2010). These developers believe if a player cannot do
what is put in front of them, they must simply get better at it until they can. This does not mean
making it impossible, because that would not be fun for the player. But instead, the developers
make it difficult enough that they struggle and when they finally beat it, it will be that much more rewarding. In the end, only one style made sense to me that would allow me to simultaneously learn what Unity (Unity Technology 2005) had to offer and push myself to make an engaging and challenging game. I decided that it would be a 2D platformer.

Tools and Software

For this project, there are several pieces of software that are vital to the creation process. The most important of which is Unity. Unity is a game development platform that allows the user to create whatever they would like using C# as the programming language to do it. Of course, Unity does not provide me with everything I need, rather it provides a framework to put the pieces that I create into. It works very similar to a puzzle, where I know what I want the puzzle to look like, so I create the puzzle pieces to make up the whole picture. Then, Unity allows me to assemble the pieces into this bigger picture that I created. In addition to being a container for all the pieces to get placed into, Unity also assists in organizing the pieces into different smaller containers, called an Empty, so that as the project gets bigger it stays neat and these Empty containers fill up with similar objects instead of having the pieces all mixed up and randomly sorted. This helps not only visually but especially when there is a problem with something, the ability to know exactly where that problem object is located is invaluable to the debugging process.

Unity has many built-in features and basic assets that I can take full advantage of by building my more complicated components from these prebuilt products. Specifically, Unity has many user interface (UI) elements that allow the player to interact with things on the screen such as buttons and adjustable sliders. With these are several built-in values that are associated with
their respective components. For example, a volume slider has a number that represents how high the volume is while the button can perform an action the user creates when the button is clicked. Unity also provides basic shapes and colors to place these UI elements on top of. In addition, Unity also has built-in libraries of code that I can reference from within my own code. This not only saves a large amount of time and effort for me to not have to make functions that interact with Unity’s components, but it also allows me to access information I would not normally be able to access such as the amount of time since the level started using Unity’s Time functions.

Unity has a specific kind of game object, called a Prefab, that allows the game designer to create an object and reuse that object again and again without having to remake it every time. It could be as simple as one object sized perfectly or many objects and pieces working together to make a complex product. If the object is going to be used more than a couple of times, especially in more than one level, it is best to make that object into its own prefab. This has huge benefits to efficiency during the creation process. More accurately, the longer the game development process goes on, the more of a time benefit you get from the creation and utilization of prefabs. The reason for this is that, at first, you must take the time to create the game object that the prefab is based on. Also, early on in the development process is when many of the prefabs have yet to be made so it is most likely necessary to make a large number of prefabs. However, once this arduous and time-consuming process is complete, the time-saving benefits will rapidly outweigh the time spent making it. At this point, the process of adding an instance of the prefab is as simple as dragging and dropping it into the desired spot using Unity’s GUI. The time saved by using the prefab is equal to the difference in the amount of time it takes to remake the object from scratch compared to the drag-and-drop method of the prefab for each instance of the object.
This difference is anywhere from a couple of minutes to a couple of hours per use of the prefab depending on how involved the creation of the object was.

One of the best things about Unity is not what it does, but rather what it does not do. It allows me to completely express myself and make what I want to make and how I want to make it. If I want a bat to fly around on the screen, as I have in my game, Unity has many helpful things to assist me in that. Firstly, it allows me to place an image of my choice on top of where the game object is in the virtual space. It even has the ability to scroll through different images on repeat to simulate the look of flying. Then by utilizing multiple-sized colliders, which track collisions between game objects, the collision area changes for each picture on the bat’s animation. This makes navigating around the bat more realistic for the player. I can add sound clips to the bats that are proximity triggered so that they only play when the player gets close enough to them. Finally, I can write the code to move around the game object that has now become a bat. But the most amazing thing is that Unity did not tell me how to make a bat or what steps to take or what parts to use. It allowed me to create the bat exactly how I wanted it to be and in the way I thought was best and most faithful to my vision while Unity was there to put it all together for me.

One of the most evident components of a video game is the graphics. However, what many people do not realize is just how much the graphics of a game can affect the style, gameplay, and even how it feels to play for the player. The last one is the hardest to gauge, as often small details can cause major changes to how the game feels. As a game developer, part of my job is to not only build the game in a certain way but also to formulate and understand the reasons why it should be one way and not another. And this knowledge only comes one way. It comes from years, or in my case almost two decades, of experience playing games of many
different genres and seeing what makes them similar and especially what makes them unique.

And armed with this knowledge I can start to utilize GIMP (Spencer Kimball & Peter Mattis 1995); a software for editing drawings and other art.

The only problem is I am not a very good artist at all. And it is rather unfortunate and frustrating to have these fantastical ideas that I do not have the artistic ability to create myself. That is where places like the Unity Asset Store and itch.io come in. These are online marketplaces where game developers can obtain premade art and animations which they can then change and use in whatever way fits their needs. For me, all of the art and animations seen in my game started as a part of one of these asset packs. I knew the feel of the game I wanted, which is unforgiving arcade style. I also knew I wanted a 2D platformer as the gameplay. So, what was left was finding art that is similar enough to what I wanted, yet not so specific it cannot be used with assets from other packs and other artists. For instance, if the art style is too pigeonholed into one genre such as 2D medieval cave crawler, even if I like the rock and cave assets, it is unlikely to fit with the look of the player sprite so I cannot use it. However, once I found art that I liked, using GIMP I was able to open these cookie cutter-esque asset sheets (See picture above) then pick and choose what I wanted to use. I was also
able to modify these pieces in any way I wanted including adding or removing sections or modifying an existing part.

Throughout the game, there are 3 hidden collectibles to gather. These collectibles make up the sword of the main character for the player to escape the cave in the end. Interestingly, this was not one of the assets in the character pack. To start creating it, I pulled up the character’s attacking animation. An animation that goes unused since it did not fit the style and gameplay I was aiming for in my game. The reason is that the ability to fight back gives the player both an easy way out of platforming around enemies and a sense of empowerment over their environment. However, removing this component forces the player to think defensively and play intelligently to overcome the challenges before them. Midway through this animation, the character has the sword extended out in front of them (See picture right). Then I went through and trimmed out the pixels around the sword so that I could more easily work with it. I proceeded to carefully define sections of the sword and split it into 3 parts, the hilt, the middle of the blade, and the tip. Together they formed the complete sword but alone they could be hidden in the level to form the collectibles. At the end of this process, however, I realized that when separated, the individual pieces no longer look like parts of a sword. So, I had to go back to GIMP and stretch my artistic capabilities to their limits. I started simple and took the time to look at what I already had and to ascertain why I thought it was not working once broken up, then I could come up with a plan on how to fix it. What I realized is the reason the sword works as a whole and not in parts is that the sword was originally drawn to have the player holding the sword while swinging it. This allows the awkward shape of the sword to be offset with the player’s proximity and the rest of the animation to follow so it looks completely normal. However, taking these away makes the color choices of the sword
awkward and ruins the sword. By using light colors for the top pixels and dark for the bottom ones to simulate an overhead light source and some depth, I was able to redesign the sword almost entirely. This resulted in a wider-looking sword with much more clearly defined lines that can be recognized as a sword both as part of a whole and as individual pieces.

When I needed to work with the sound effects and music throughout my game, I used a program called Audacity (The Audacity Team 2000) that allowed me to make any edits I needed to. As with the art, my base music and sounds came in prepackaged packs. Unlike the art, however, many of these songs and sounds were generic enough that I was able to pick the ones that I liked without worrying as much if they fit into the style of the game. The audio editing mostly came in handy when working with the bat audio clips. There are numerous bat audio clips but finding one that fits the style I was trying to create proved to be a tedious endeavor. Eventually, I found a clip of a bat making several different chirps that I thought sounded good. I then took this full audio clip and selected the one section that I thought fit the most and exported that section as its own audio clip, allowing me complete control over what the bats in the game sounded like without having to compromise with clips that were far too long to put into the game.

The Complete Product

Pre-Production

Something I have discovered during this long process is that everything is a work in progress. At the beginning of the development process, I wanted my game to be a platformer where the main character was a mango. This mango had his girlfriend stolen by bats while they were out having a picnic in the forest. Then he would venture into the cave, save his girlfriend
from the bats, and climb back to the top to escape. However, in a twist on the classic save the princess trope, at the end of your journey, she would get grabbed by the bats and you would lose her forever. This story echoes the ancient Greek story of *Orpheus and Eurydice* (Ibycus 530 B.C.E). Wherein the wife of Orpheus, Eurydice, dies and gets transported to the underworld. Orpheus then travels to the underworld and gets the opportunity to bring her back. However, right at the exit to the underworld, she would be taken from him again, never to return. This default ending was, for obvious reasons, the bad ending. However, the player had a chance to save her and would experience a more traditional save the princess style ending. By collecting three collectibles throughout the game, you would be able to defeat the bats at the end and escape with your mango girlfriend. Clearly not all of my original plans would make it to the final product.

One of the first things to get left behind was the main character. Without being an artist myself, I am forced to rely on what other people have created. And as I searched page after page of assets it started to become clear that no one has ever drawn a 2D mango character for me to use in my game. This is when I expanded what I wanted and found my current main character. A character who I think effectively conveys the games guiding themes of adventure and exploration. With me finding a new main character, that also meant I needed a new character to save. Now the plan was to search for a princess for my metaphorical prince. However, I realized that my storyline of bats being the main enemy that stole the character’s girlfriend did not make as much sense now that my characters were human. So instead of forcing an idea into a hole that it could no longer fit, I decided to redesign the idea to fit into this new hole. And the more I thought about a potential story and tried to figure out what could work, the more I realized that it
was more important to move on and come back when I had more to base the story on. So, with this decision, I moved into my notebook to sketch out the environment.

Just like with the storyline, my ideas for the environment during the beginning of the process were different than what I had by the end. What I ended up with is so much better than what I originally thought I wanted. When I set out on my design process my idea was to have a single room where the blocks and obstacles would come up through the floor and try and drag you to the ceiling (See picture top right). The ceiling would have had spikes on it that would kill the player if they got too high up. This would also be accompanied by various spawning power-ups and coins to truly embrace the old-school arcade vibe created by the 16x16 pixel art style. But instead, I decided to redesign the room to have the same gameplay as my original idea while feeling much more
realistic and open to exploration than a closed room. To get the gameplay of dodging objects and spikes as the player tries to travel down, I extended the floor down to create a large rectangle the player would be inside to act as the level (See picture above). Instead of making the obstacles move up, I instead would make the blocks and other objects below the player so that as gravity pulls the player down the level, they meet these obstacles and must work around them. I also knew that as the player descends, I wanted the gameplay to get much harder by adding objects like spikes and, although not the main focus of the game anymore, bats that would make an appearance as pesky moving obstacles the player must avoid at all costs. Instead of spikes on the ceiling being the persistent threat to keep the player on edge, I would add automatic downward scrolling to the camera so if the player was too slow, it would leave them behind and reset the level.

The process of sketching out level two was very similar to the first except almost all of my original level two ideas made it to the final version. My idea was to take the gameplay I just created and rotate it so that instead of going down, the player would travel right and face the same style of platforming gameplay and face the same enemies as level one. I also had the idea at this point to create stalactites that fall from the ceiling as you go under them that could end the player’s run if they were not careful or quick enough (See picture right). Once level two was completed, the player would then have a section of gameplay that would be a
reverse level one. Where instead of trying to race down as fast as possible, they now had to race up to escape. I was aiming for gameplay akin to popular games such as DoodleJump (Lima Sky 2009) or Icy Tower (Free Lunch Design 2001). Once the basic designs and mechanics were established, it was time to start working in Unity.

**Start Screen**

Once I had the basic outline for the game and each individual level, it was time to start creating it. To make the final product easier to understand and describe, I will mostly go through my work in the order it is encountered in my game instead of the order it was created unless it makes more sense to talk about an entire topic that spans the whole game. One example of this is the music that plays as soon as the player starts the game. Having been deeply involved in music and music theory for a decade, I have a solid understanding of both the importance of music to the player’s emotions and what kind of songs would get the player into the proper headspace to immerse themselves in the game. To start, I knew the title screen would be inside the cave with the underground lake shimmering in the background, So I wanted the player to be welcomed to a slow serene song that slowly builds throughout the storyline screens to enter into level one. Then level one would be a medium-intensity song to put the player into the adventuring mindset that would kick off their journey. This song is very base drum heavy which provides the song with both intensity and a drive that pushes the player to begin their journey and conveys the feeling of taking on the great challenge to come. Level two has slightly faster-paced music to indicate the player is in the middle of their perilous journey while still having a driving bass line like the level one song that conveyed a sense of adventure and motivation to keep pushing through the challenges that the player is facing. The final song is more upbeat but equally fast-paced as the
song from level two. The reoccurring portion voiced in the alto range gives off the feeling of hope and a sense of nearing the end of the journey. While the lower voiced sections layered under remind the player their death can come at any moment and their journey, while close to completion, is not yet done and they must finish this last fight.

Going back to the start screen, the player is greeted with a welcome message across the top of the screen in a custom font that fits the art style of the game (See picture above). The background of the start screen is multiple different pictures that, when layered on top of each other, form one cohesive picture of an underground lake. There are three buttons labeled options, start, and exit. With the exit button being self-explanatory, I will move on to the options button and how it works. The button, once clicked, activates a panel with audio sliders that adjust both the sound effect volume and the music volume for the game (See picture right). Those settings also persist throughout the entire game due to a Unity library function called playerPref() which allows me to store values, in this case, the volume, then access those values at any time I require. The player may press the red square to exit out of the options menu. Once the start button is hit, a black box begins to fade onto the screen followed by lines of text fading in and out telling the story of the main character and his goal of finding the lost treasure in the
caves below (See picture below).

Finally, after the final line has faded out, instructions on how to control the character and how to pause the game fades in along with a continue button that brings the player into level one.

**Level One**

Level one starts the player at the top of a vertical shaft. The camera will then start automatically scrolling downwards, forcing the player to follow suit. The camera scrolling also increases in speed the longer the player spends on that attempt at the level. I made the decision to speed up the camera scrolling based on time for two reasons. Firstly, this makes the game reward quick early-level gameplay by giving the player more time on the challenging parts later on in the level because the screen is moving slower. Secondly, near the end of the level is the first collectible of the game. It is tucked in a corner guarded by a bat that flies across the screen. Then, once grabbed, the player must jump through a hole in the ground between lines of spikes. However, if the screen is scrolling too quickly by the time the player arrives at that section, it is impossible to grab the collectible and still make it to the next part of the level. I wanted this collectible to be one only obtainable through skill and fast high-level gameplay and there is no
better way to ensure that than to make it impossible if the player is not able to make it through the level quickly enough.

The camera script I created also allows for what is known as edge scrolling (See picture right). Edge scrolling is when the camera gets dragged through the world space based on the position of an object within that camera’s view, in my case the player. This was monumentally difficult to implement because of the additional constant auto-scrolling. This means that not only do I have to make a process for moving the camera in during each method, but I also must define when to switch between the two and how to make the transition seamless for the player.

The logic of moving the camera breaks down into two components, the distance and process of the camera moving on its own, and the distance and process of moving the camera as a result of edge scrolling. The camera has a start position and an end position defined by coordinates in the world space. Using a built-in Unity C# library function called Lerp allows me to move the camera smoothly from the start to the endpoint. While traveling along this path, the fraction of the total distance covered is calculated and used
to ensure a steady movement along the defined line from start to finish. It is calculated using the speed of the camera’s automatic movement and the amount of time spent in the level. The complicated portion of the calculation occurs when the distance covered is no longer equal to this simple equation but must now consider the forced camera movement as well. Now, when the camera is auto-scrolling, the distance covered is equal to the relationship between the time, the speed of the camera, and the camera’s movement caused by the player even if the camera is not actively edge-scrolling. When the camera is moving due to edge-scrolling, the distance required to move to keep up with the falling player is equal to the camera’s position in space minus the distance the player is in relation to the center of the camera and the player’s non-camera related position in the world space. It must then add that to the total distance covered and recalculate the fraction of the total distance while simultaneously moving the camera the required distance defined by the edge scrolling formula outlined above. This results in a camera that has dual auto and edge scrolling capabilities to allow the player to accelerate the pace of their gameplay if they so choose.

As the player descends downwards in the level, The environment reflects their proximity to the surface in multiple ways. Firstly, the early platforms that the player jumps on along with the walls on either side are covered in grass or have plants hanging from them (See picture right). The
background, which is a repeated image layered behind everything else to give the illusion of depth, gives off the appearance of being well-maintained with structural reinforcements in the form of wooden beams. However, the platforms that are deeper down slowly start to mix in more solid rock platforms and fewer grass platforms until there are only rock platforms left. While this obvious transition is going on in the foreground, the background undergoes a much more subtle transition where the walls begin to lose their support structures and turn into bare rock walls giving the player a sense of journeying in areas that have not been upkept or visited in a long amount of time. These two effects combine to help give validity to the storyline of your character being a hunter of forgotten treasures that is going to go farther than anyone ever has before.

Early in the game, the player will face their first challenge and opportunity to die. Spikes are one of the two major ways that the player can die, and they have to face a copious amount of them throughout the game. The spikes contain a collider that acts as a trigger for the player’s death. Once the player enters the collider of the spikes, the game immediately stops taking the player’s inputs so that the player can not move the character after death. Then, a timer starts playing and once this timer is up, the level gets reset and the player is back at the top. The reason that I decided to add a timer instead of a reset as soon as the player dies is that I wanted to have a character death animation to indicate to the player that they died instead of them being surprised by being instantly sent back to the start. This is also more realistic for the player and can help immerse them into the game more by having them watch their character fall over and die based on their actions instead of an instant unengaging reset.
In the last quarter of the level, the player is faced with a new challenge, bats (See picture right). These bats consist of multiple components such as an audio clip, an animation, and a movement script. The bats are programmed to pick a random speed value in a pre-defined range. I decided to make their individual speeds random to make it more engaging for the player so they cannot do the same thing every time to get through the level. Because of this small design feature, my game gets a major boost in its replayability.

The bats then travel back and forth along the screen at their randomly defined speeds and act as moving obstacles for the player to dodge as they traverse the level. When the player gets close enough to the bats, the bats begin picking numbers at random using a random number generator. If the number is in a certain range, their associated audio clip plays. The reason for this seemingly overcomplicated process is realism. When the player is at the top of the level, it would make no sense to be able to hear the bats all the way at the bottom of the level. But then once they get to the bat sections, there is the problem of how often the bats should make sounds. If they make too much noise, then it is not only annoying but also drowns out the music of the level. However, on the other end of the spectrum, if the chance of the audio clip playing is too rare, then it will be as if they do not make any noise at all. It is a very delicate balance to maintain. In addition to taking into consideration the chance of each individual bat’s audio clip playing, the density of bats in one area also affects the audio balance in the level. This is because
each bat has their own number generator. So the more bats I add, the more chances there are for the clip to play during gameplay without directly changing the chance of the clip playing for any given bat. For example, the chance of the clip playing for one bat may sound very balanced, but then adding in another five bats makes the audio play five times as often, which would throw the audio balance off significantly.

As previously mentioned, in the last section of level one is the first collectible of the game. With it, a major, unforeseen, problem arose with the way I implemented my collider detection with the player. Before this point, anything the player interacted with that had a collider that was a trigger, namely the spikes and the bats, would send the player into their death sequence and would restart the level. However now I wanted to implement a collider that left the player alive and did other things behind the scenes from the player’s view. First, I need a counter to increment indicating that the collectible was grabbed. Then a pickup sound needed to play accompanied with the object disappearing so it can only be picked up once. After adding these actions into a script and attaching it to the collectible I expected it to work as I intended. However, during testing the collectible collider would instead kill my player and reset the level exactly like the other ones. I found the reason for this was that there was no distinction between good and bad colliders. There was just one kind of collider and if the player touched a collider, it would kill them. What I then needed to do was add a built-in Unity feature, called a tag, that would allow me to make that distinction. So, I added a deadly tag to the bat and spikes prefab which changes all occurrences of those objects saving me the time of adjusting them one by one. I then added a collectible tag to the sword piece. Now in the player’s collision detection, it has the ability to check the tag of the collider that triggered it. If the object that has the collider had a deadly tag it would start the death sequence of the player. And if the object had the collectible
tag, it would do the collection sequence instead of the death sequence. And with that figured out, all that was left was to change the level the player was in once they reached the exit and they would be onto level two.

**Level Two**

While the challenge of level one came from creating all new objects and formulas to make my game function and figuring out how to put these creations together, level two was a mix of different steps in the game design process. This includes utilizing assets I already made for level one, modifying some of them accordingly to fit level two’s play style, and creating a few new objects and movement formulas. And while level one starts very easy and moves into more difficult gameplay, level two would start at an increased difficulty and end at an even harder difficulty than level one. Right from the beginning, the play style of level two is different than level one. Where level one goes downwards, level two goes right to progress through the level. This changes how the player thinks and forces them to adapt what they have learned in level one to make it through just as I did when designing the game.

One of the first things the player will notice about level two that differs from level one is the background (See picture right). Additionally, for those observant enough, it is possible to notice that the background to level two is a variant of the title screen background and further into the level it is the exact same. However, the background in level two is significantly more complex and as a
result is much more realistic to look at. This is due to the background layers of level two moving at different speeds in relation to the camera’s movement. To explain why I decided it was better to move the entire background and camera instead of just the camera, first imagine a big open area such as a forest or a canyon. When a viewpoint moves in real life, such as a person’s head, the perceived distance the objects around the viewpoint move is based on how far away those objects are. If the object is right in front of the viewpoint, it appears to move a significant amount more than an object way off in the distance. This is what is known as parallax movement. And I fully intended to utilize this real-world phenomenon in my game. To start I had to modify the camera movement script to now move right instead of downwards. Once that was complete, I began working on the background itself. Each layer of the background is assigned a number based on how far away the object is supposed to appear (See picture right). This number is calculated as a fraction of the distance the camera moves between two frames. For objects that are supposed to appear far away from the camera, I move them to the right almost as much as the camera’s movement, effectively slowing down the image’s travel time across the screen. And for objects to appear closer, I simply move them a tiny amount right in comparison to the camera’s movement, so they go across the screen very quickly. With this
small detail that most players may not fully grasp the intricacies of, my 2D world suddenly gains what appears to be miles of depth in the background and that perceived depth makes the game so much more realistic for the player to experience.

Right at the start of level two, the player is immediately met with consecutive spike jumps and platforming over a sea of spikes (See picture right). In this beginning platforming section, I play tested for over an hour to find the perfect balance between the distance of the spikes and platforms to each other to allow for the player to perform quick fluid movement throughout this section that resembles a clean speed run. Then the player is met with more bats. These bats are modified from the first level where they now travel farther on each way of their patrol cycle. This is also the point where the player gets the option to either take the high-ground route or the low-ground route. Giving the player a choice on how they want to play the game adds even more to the replayability.

Regardless of the route chosen, they end up at a section I refer to as the spike run. For early 2010s Flash gamers, this might sound very familiar. This section is partially inspired by the bottle run levels of the hit internet game Happy Wheels (Fancy Force 2010). The objective of these bottle runs was to travel as fast as possible on a ground that collapses underneath you as you touched it. So, if you moved too slow you would fail and must restart. The objective of my spike run would be slightly more dynamic and would instead see stalactites fall from the ceiling as the motivator to keep moving instead of a falling floor. The stalactites were a new asset I had
to create. The stalactites would calculate the distance between them and the player and, once they were close enough, they would begin to fall toward the ground. I accomplished this by essentially turning off their gravity while the player is not near them and then once the player comes close enough, I turn it back on allowing them to fall to the ground. These stalactites however would just fall straight through the floor which ruined the realistic immersion I was aiming for throughout the level and the entire game. To fix this, once the stalactites fell low enough in the world space, I would turn their friction up to max and as a result, they would stop moving and appear to stick out of the ground instead of falling through it (See picture right). This worked incredibly well and moving into the end of the spike run section the player has the opportunity to grab the second collectible in the game. The end of the run breaks into another upper and lower section. The upper section, being the easier section, requires the player to position themselves between two falling spikes (See picture right). Then, they may move on to the last section. However, if the player wants the collectible, then they must traverse the lower path. This lower path requires pinpoint precision to dodge not only spikes and the same falling stalactites as the upper section but also a bat that flies by that will end their run if they are not careful. The player must then maneuver around the stalactites sticking out of the
ground and finish out another, more precise, platforming section over another sea of spikes where they can easily fall into the spikes below (See picture below).

Looking Back on the Process

As I near the end of the semester, and as a result the end of my thesis project, it is important to reflect on the process and think about what I learned and what I would do differently. To start, one of the biggest and most important things I learned was all the different parts and processes involved with designing a game. From the sketchbook to the final product, I was involved with designing, assembling, and troubleshooting everything. I also learned that what I originally wanted is not what I ended up with, but what I ended up with is much better than what I could have imagined. Even though I had an idea of what I wanted, I decided what was best as I went and because of that, everything worked seamlessly off each other to form a cohesive final product. In addition, a large part of my journey was discovering what the Unity program was capable of and figuring out how to make my ideas come to life using their provided
functionality and libraries. And once I started understanding how everything was supposed to fit together, there was nothing I could not achieve. The other tools I used, namely GIMP and Audacity, played smaller roles in the design process. Therefore, I did not learn as much about their software as I did for Unity. However, I still was able to utilize them effectively enough to achieve my goals without running into any major problems while using them. As for what I would do differently, the only thing that I would change is making my main character a mango and going with my original storyline. I thought that the original storyline had the potential to have the player be more emotionally connected to the characters rather than a straightforward treasure hunt and had a certain charm about it that is unmatched by the alternative storyline. I also had an amazing idea for part of the mango’s movement to be a roll mechanic and the mango would be able to roll under bats and objects which I would have loved to have implemented and integrated into the level designs.

**Future Work**

The future of my game is one filled with many new ideas and some old ones that I did not have the time to make during the semester. For starters, the third level of the game and the climax of the player’s journey is not yet complete. Of course, at this point, I have some time put into it and the foundation for what it is going to be. However, many of the gameplay elements that are present in the rest of the game are not yet implemented in the level yet. There are also no new encounters to experience in the third level. This is not necessarily a bad thing, however when I have put in so much care and attention to detail when it comes to the game’s replayability and making the player always adapt to new challenges, it seems like a major shortcoming especially so close to the end. To remedy this, I intend to add spiders as an enemy in the third
level. These spiders would hang under the platforms that the player can jump on, potentially killing them as they try to ascend if they are not focused. Once reaching the top, the player will be met by a massive bat. And if the player did not collect all of the pieces of the sword, they will be unable to defeat it and will get the bad ending. This leads to my next feature which is yet to be fully implemented, the sword collectibles themselves.

Collectibles have two steps to completing them. One is the actual creation of the collectible. This includes photo editing and the manufacturing and assembly of the different physical parts. At this point in the process, where I am currently, I have a collectible that can float on the screen which disappears and makes a noise when the player walks into it. But it does not actually do anything beyond that. That comes in step two which is the scripting of the results of that collision. For this, I intend to have a counter that I store in the PlayerPref() which increments every time a collectible is grabbed. However, there is a problem with doing this naïve approach. To understand why this solution, in its current state, would not work suppose the player grabs the collectible on level one, so their counter goes up to 1. Then they die, and on their way down they grab the collectible again. It would then go up to 2 erroneously. The process for getting around this is fairly simple. At the start of each level, take note of what the player’s collectible count is. Then increment as normal but if the player dies, set the number back to the original before the level resets itself. To see if this will work, we must think about all the possible ways the player could interact with the counter. Firstly, if they die before getting to the collectible, the counter will remain unchanged and will be accurate. Scenario two is if they grab it then die, which works because we built the solution specifically for that scenario. Finally, if they grab the piece and make it all the way through, the counter will carry over to the next level and the process will repeat.
Not everything left on my to-do list is as vital as the secret ending or as large as an entire level. However, if I have demonstrated anything it is that the smallest details can make or break a game. Those additions can be the difference between a good game and a game where every aspect of gameplay is tailored to the player’s gaming experience and their immersion in the universe. One idea I have for this is cleaning up the pause menu. As is, the menu is very minimalist and does not fit the style of the game at all. Adding some personality to the art and making it look like the menu belongs with everything else on the screen can do wonders for a player’s experience. Another idea was to add images of a glint that fades in and out on the start screen to give the illusion of glistening water. The final feature I will discuss is a feature I was less sure I wanted in the beginning, but the more work I put into the game the more I want it. I want to add a player animation that follows the mouse while the game is paused and does the attacking animation when the player clicks on anything. So, it looks as if the character is breaking the fourth wall and interacting with the menu of the game he is in.

References

530 BCE. *Orpheus and Eurydice*. Ibycus

1995. *GIMP*. Spencer Kimball & Peter Mattis


2000. *Audacity*. The Audacity Team


2011-2016, *Dark Souls*, Tokyo. FromSoftware

2022, *Elden Ring*, Tokyo. FromSoftware