**Leg it! Games Project Report**

****

[**https://github.com/leyla2/Leg\_It\_GITHUB.git**](https://github.com/leyla2/Leg_It_GITHUB.git)

**<https://www.tumblr.com/blog/leg-it-game>**

**Contents**

**Page 2: Abstract**

**Page 3, 4: Introduction**

**Page 5, 6: Background Survey**

**Page 7: Specification**

**Page 8, 9, 10: Design & Implementation**

**Page 11, 12, 13, 14: Testing and Evaluation**

**Page 15: Conclusion**

**Page 16: Bibliography**

**Abstract:**

**Leg it! Is a finite endless runner game designed in mind for those looking to think on their feet and put their reaction times to the test.**

**The game isn't complete to the desired standard of the developer and its uniqueness is slacking compared to other finite runners on the market. Many different strategies were used in an attempt to allow this game to stand out, such as using different characters and multiple levels. The end result is a game with some sense of addictiveness due to it’s increasing difficulty and varied aesthetically-pleasing graphics.**

**Introduction:**

**Unity was used during the making of this game due to it being one of the most popular game engines at this time, generally thought to be very user-friendly and easy to master. It is thought that Unity is so easy to use that no coding knowledge is necessary during the process of completing a playable game. I ran into many problems during the use of Unity, including but not limited to, instances of objects were removed from public variables, external scenes were saved over one another and different versions of Unity wouldn’t run the exact same game once pulled from GitHub.**

**There are two types of endless runner games, an infinite runner and a finite runner… an infinite runner is that whereby the platforms are randomly created as the player moves across the map and each time the game is played, the platforms may be in a different location. A finite runner is a map where the objects (excluding the player movements) are already pre-set, this may cause more memory to be used, however I believe that it’s fine for more memory usage during my project as the map isn’t tremendously huge to cause a system crash and it allows the player to master the game instead of being irritated they constantly die after every level reset. I initially wanted to create an *infinite* runner game. My idea of an endless runner game was derived because of huge inspirations from the likes of Fun Run and Sonic which were both games that I’ve spent a lot of hours playing and never got bored of. At first, I was using object pooling for the infinite generation of platforms, however this quickly caused major delays and didn’t go to plan, therefore, I quickly decided I should change the concept of my project to a finite runner, whereby the platforms are already pre-set. The finite runner type of endless runner would need an increased difficulty of gameplay, if the difficulty is not high enough, then the player will easily speed though each levels, especially after playing the game a few times. Players will become aware of the upcoming platforms and will have already trained their minds to successfully pass them quickly.**

**There are many different aspects of Unity I have learned during the creation of this game project such as different methods in C# [OnTriggers, IEnumerators], how to use sprites in different ways such as for repeating, organisation skills, prefab usage and using animation.**

**Before the start of this project I had no idea that IEnumerators even existed. When I wanted to make a trap whereby the player freezes for a few seconds, of the top of my head I first though that I should set the velocity of the character to zero or disable the script altogether, however, I soon figured out I was very wrong as this wouldn’t allow me to restart the script or increase the velocity after a few seconds – which was the plan. After intense research I came to a realisation that many people were using IEnumerators to imitate similar characteristics in their projects. After researching the coroutines available to use in the IEnumerator functions, I decided to play around with a few myself, including yield WaitForSecondsRealtime which let me create a frozen-like delay effect for the character during an OnTrigger.**

**I was unaware that OnTrigger2D had an OnTriggerEnter2D and a OnTriggerStay2D configuration, both enabled me to do different tasks during gameplay such as removing health and instantiated blood splatter particles… it wasn’t until later on I found out that I don’t need to have my player constantly ‘bleeding’ during damage taken, as I believe this looked amateur and very unrealistic. Although the game is constructed of cartoon-like graphics, I believe some sense of reality always entices a person to keep playing.**

**During this project I also learned that sprites/spritesheets don’t only have to be used for cutting up and immediately setting onto the scene. It is during some tutorials I watched that I caught onto the fact that they may be set to different textures and different wrap modes. One example of where I used this was during creating the repeating backgrounds for each level, I turned the sprites into default modes and set the clamp mode to ‘repeat’ which allowed the image to be repeated over and over again [image the image constantly wrapping itself around] inside a quad texture. This gave the character a sense of physical movement, which is also another sense of reality being portrayed here.**

**Usually I am not a very organised person, albeit at times I wasn’t during my project construction time for the past few months. However, I have improved vastly in terms of organisational skills, so far I have grouped my scenes into a few parts, including platforms, camera, starting platforms, decorations and in some scenes there are other miscellaneous groups such as ground – which consists of the brown area of the platforms… this is because I didn’t want to set these to ground as they caused the player to have unwanted collisions with them and possibly walk inside them.**

**I had to do some online research to know exactly what prefabs were used for, I didn’t know that they were able to be instantiated although they may not currently be in the scene… knowing this broadened my horizons for the uses of them. I then immediately started to create trap prefabs – which initially I was going to use for the object pooling system that I first created for the infinite runner game, however after my plans changed to have a finite runner game, I used the prefabs for duplicating traps across the maps, this saved me much time and effort instead of manually adding the same scripts and configuring the same settings for every time. The bullet was also used as a prefab, this is an example of use whereby the object wasn’t already in the scene and had to be instantiated in the script in accordance to the player’s transform numeri’s.**

**One problem I did encounter the prefabs was the centre positioning to the master/parent prefab. I started to become frustrated that whenever I would place a prefab in the scene and play the game, during the game view they would appear somewhere else. Such examples were during the rolling saw trap, it was damaging the player because it’s OnTrigger collider was there, however, the saw was out of game view. After trying to research about this problem, I quickly came to a realisation that it must be a problem with the parent prefab because this same situation was occurring on all the prefabs I extracted onto the scene. After playing around with the parent prefab’s transform numeric, I solved all the in-scene prefab problems – I was very proud of myself and this is definitely something I kept in mind for all future prefab use.**

**Working on my final year project was a huge learning curve for me, because in previous projects given to me I had to work in groups whereby I was told not to do much at all or the task was to only script. Never have I been able to work in Unity so independently, have the responsibility of all tasks that was produce the final outcome of my game. I like to think I’ve learned a lot, during this report you will be able to gain better knowledge of my journey of 100% making my first Unity Endless runner game.**

**Background Survey:**

**The development of Leg It! was heavily influenced by Fun Run and Sonic games.**

**Fun Run is a real-time multiplayer mobile platform game developed for those that are competitive in nature and have little time on their hands. The graphics are very comical and captivating with little detail as compared to other finite runners. The game has no story line, unlike Sonic. Fun Run characters all have the same starting speed, this may be increased with temporarily with power-ups that are collected across the maps, players rely on throwing out obstacles such as lightning and bear traps in the game to slow everybody else down, which would give them an advantage of being ahead of everybody else.**

**Sonic is a game sequel created by SEGA in 1991 with old-school, low graphical settings based highly on story lines, with pauses in-between some scenes to explain the story to those fans/players that were very interested in understanding so. It is not a real-time game and may be played solo. Power-ups are also a part of Sonic, this is where Fun Run and Sonic have similarities.**

**Power ups and having a solo game was part of the plan, as described in the project proposal report. I definitely wanted to make a cartoon graphical theme such as both aforementioned games, both of which are uplifting to watch and play due to their graphics. Many obstacles came in the way of a few other ideas I wanted to implement such as time and lack of creativity for myself.**

**Originally, I wanted to make the game a multiplayer through a network, allowing players in the local network to play against each other, possibly competing for the best time. A local network was preferred rather than a server-based network as this would have a high cost and electronically demanding input. The main problem I and others realised was that I would be unable to make a multiplayer game due to lack of knowledge in the programming aspects of this. It would have been possible to make a multiplayer game using the same keyboard, however, that in my opinion would cause discomfort to the players and is not how I intended on making my game. It was all or nothing with making a local-network multiplayer game or not doing multiplayer at all.**

**My game needed to have a different aspect than other current games on the market, I haven’t seen an incoming killer wall from behind so I decided to give this a go, I soon realised that I would have preferred an object to follow the player such as a saw… this would enable me to duplicate this into all the scenes and would not have me faffing about trying to configure the ‘killer wall’ to mirror the speed of the background or create individual colliders for each spike on the wall. Instead I was able to create a moving saw which approaches the player using only a circle collider and an OnTrigger. The speed of the saw is adjustable, also allowing increased difficulties as you run through every scene. It’s a win/win situation in my opinion and in the opinion of others’ that play tested my game.**

**Sonic is a 3D game, although I would have wanted to make a 3D game, I have far more experience in assisting in the production of 2D games to be able to have enough confidence to make a 3D game without many setbacks that would slow me down a lot. I just don’t think that four months is enough time for somebody to be able to heavily research and smoothly implement regarding something they haven’t done before.**

**I wanted to make customisable character selection, such as that in Fun Run, alternatively I ended up creating two different characters for different scenes. Both characters are able to do different animations. The reason I didn’t create a player selection on load is because both of the players needed to have different scripts, as the abilities of both characters differed, causing different scripts to be relied upon.**

**Specification:**

**The idea of the game initially was set out to be a time-consuming (approximately 10 minutes) infinite endless runner game. I wanted to make it for the person who has time to sit by their computer and put effort and passion into wanting to play the game. The whole game was supposed to be one scene with the same theme, but it may last as long as you want it to, the player would literally be able to play infinitely and there would be no pause, no breaks, it may be called a stealth-mode implementation. A high-score system was supposed to be fitted in to be able to determine who ran the farthest out of everybody who has played the game.**

**The idea of a stealth-mode like infinite runner was derived from both Fun Run in its stealth abilities, whereby moving backwards was not an option and high score system, also Sonic was admired for it’s long duration of game play.**

**A wall closing in from behind was supposed to be implemented, similar to those from the Saw movies.**

**Sprite sheets were to be made by me from scratch, I have done this in previous years group projects using Adobe Photoshop… although this can be very time-consuming, I think it looks very high quality, smooth and adds innovativeness to a game.**

**PC, linux and Mac users were the target audience of this game and it still is. Although implementations on a different platform would be easy, I don’t think that it would be a very enjoyable game on other platforms. For iPad/touchscreen users there wouldn’t be a comfortable way of smoothly jumping and shooting simultaneously or sliding whilst crouched would cause issues, the user may experience finger burns or just plain be unable to use the game’s mechanics 100% to their advantage unlike users with keys to press.**

**Users with controllers such as the PS4 and Xbox have many different buttons on their controls, therefore, I think it would be too basic of a game to enable it onto their platform, whereby only the joystick, a jump button and a slide button would be used.**

**Users with a laptop or a computer seems that they would have the best experience compared to others. Also, many friends and family that I know own a laptop or computer, in comparison, not many own or have access to a PS4 or Xbox for example. I believe that if this game was made for any other device other than a laptop or computer that the target audience would be significantly reduced, and I wouldn’t be able to playtest my game with as many friends and family members.**

**I wanted the game graphics to be physically very appealing, because whenever I play a game, if the graphical side of things are low and seems uncared for, it usually makes me uninterested in the game very quickly. Cartoon-like graphics were expected with a low-render rate requirement to avoid downfalls during gameplay if some users may not have a high-spec PC/laptop.**

**Many traps were at first thought to be implemented, this is to add time-consuming obstacles in the game, instead of the user just flying through the maps within seconds. I didn’t have too many ideas with which traps would be used as I’m not very imaginative, however, once I saw some assets that I wanted to use in the game, I started to imagine the traps and their capabilities to slow down the player. I initially knew that I wanted to insert the classic bear trap and falling platforms.**

**Design & Implementation:**

**I was considering whether to make a 2D or 3D game within Unity. The 2D aspect appealed to me more, not only because it is easier as I wouldn’t have had to work with the Z-axis but the fact that all animations and objects are able to be viewed from only one angle. I also admire the retro feel of having a 2D game. Upon research, I gathered that the colliders may work better during a 2D layout. Also online there were plenty more tutorials on how to do basic tasks in 2D rather than 3D Unity. As I didn’t have a great amount of experience or knowledge in regards to Unity, I decided to stick with the 2D layout. I still feel that 2D is as enjoyable, if not more, to play due to the fact it’s more possible for me to create a bug-free game as compared to working in 3D.**

**I first started to create the basic map with only platforms and aesthetics, without any animations or traps yet. I then tried to implement the closing-in spikey wall from behind the player, at first I tried to make this a child of the player but then I quickly realised that the damaging wall should be a child of the background to be able to follow along exactly as the background moves and not when the player moves as this would defeat the whole purpose of trying to harm the player.**

**I then quickly decided that I don’t want a spikey wall from behind to close in as this may be too easy because the player can rapidly and easily overtake it for the entire game. Instead I decided that a rolling saw would be a better idea as the speed can be fluctuated during each level and the character’s transform location may be accounted for, for the use of following to try and damage the player. In my opinion this was a better option as it increased difficulty for the user and causes fear, encourages the user to complete the level faster.**

**Now it was time to implement the player’s movement. A basic character was added to the game from a royalty-free sprite sheet in the asset store. After some research I realised that Unity had it’s own built-in controllers for ‘Horizontal’ and ‘Vertical’ inputs from the keyboard, which then in turn gives an output of 0 or 1. I used this to be able to move the player but the problem was that it was too slow for my liking. Therefore I used this output variable as the ‘basespeed’ which I then decided to multiply by a certain number to create quicker movements during key presses. I already knew about GetKeyDown and KeyCode inputs from previous C++ projects, so it was easy to convert this in C#, just a little bit of research on Unity’s API from the website and I had it all figured.**

**The jumping implementation was my next step, so I decided to keep a separate script to implement this to keep things more organised and cleaner… or so I thought. The new jumping script I used started to cause the entire game to slow down upon start, it took approximately 10 seconds to load the scene to be able to play. I couldn’t figure out exactly why (I assumed it was due to the foreach loop within the script, which possibly kept looping over the ground checking game object I dragged onto it). So I decided that I will no longer use an entire script to implement the jumping moving and it even turned out to be simpler to add it into the whole movement script than I previous thought it would be. A simple change of the rigid body’s velocity was needed upon GetKeyDown! At first I made is so that the grounded would be deemed true if the player’s collider was touching the ground layer, however, this was changed very soon due to it causing issues of ‘jumping’ on the side of the platforms, although the players feet weren’t in contact with the platform itself. A gameobject was created with a small radius to determine the player’s feet location and whether or not it came into contact with the ground of the game, enable a better and a more realistic approach to the player being grounded.**

**I had an issue with determining why the player got physically stuck on a specific part of my map. After hours of examining my game and with the help of a few others to no avail, I finally realised that I had a box collider enabled in one of the children on my platform prefabs, I’m not sure how this occurred as I didn’t myself attach a box collider randomly to any prefabs. This issue caused me hours of wasting time to figure out why my player couldn’t’ move forward once I hit a certain area in my map. This is a good lesson learned as from then on I never forgot to check for colliders if my player is unable to move, which coincidentally saved me a lot of time in the future of this game making process.**

**Next, the running animations were taken into consideration, as without these animations the movement would look flimsy and poorly cared for. Unity enabled me to build animations very easily using Booleans, floats and other variables. The animator is very simple to use, as making transitions and enabling or disabling animations are very easy to configure in Unity’s interface. Also referring to the animations are very simple to do in scripting. The running animation was used as a float variable and was set to whatever the velocity of the character is. If the velocity of the character on the X-axis was more than 0.1, than the animation for running would start, giving a running effect to the player.**

**As of yet, I didn’t know what instantiating was in coding. I spent some time, especially on Unity’s website, researching how to instantiate objects. This was because I already thought too ahead of my time, wanting to instantiate blood, even before I created traps. This is because I thought the traps would be easy to create, so I wanted to get the harder and more exciting (I found it exciting because it was new) part over with. I then found a good video online which demonstrated a blood splatter being created and instantiated using the particle system. I didn’t learn of the particle system until I saw this video, the whole time I was thinking that I will create an image to act as the blood particles. Having found this new found knowledge was exciting for me and would mean the quality of the blood splattering would include movement of particles and be more realistic than an image.**

**Some traps that I added were basic, just to get an idea of adding traps and allowing them to hurt the player. I added a rolling saw which would activate upon the scene being played and a mace which was placed in between platforms, in case the player would fall off of a platform this would cause them to die. The traps were both given tags as ‘damager’, and upon entering the colliders of objects with the tag ‘damager’ the player would instantiate a blood particle in the same transform positions as the character, this would give an interesting bleed effect upon being damaged.**

**My grounded was causing me issues whereby the player would start jumping after it collided with anything that had a collider attached to it. This is because I used a box collider for my player to determine whether it was on the ground or not and I didn’t have any layers being used in the game. I resolved this issue by creating a game object to determine whether the player was grounded or not, this empty game object was attached to the players feet location and would be dragged onto a public game object for use in my script. Grounded was now determined in my script if the ground checker game object collided with the layer ground (which was set to a public LayerMask) using the OverLapCircle method.**

**At this stage of my game I started to create the infinite platforms, which would consist of platforms to start being created from a certain point (starting point using an empty game object) and being destroyed behind the camera view (using an empty game object again). The infinite platforms were at first created using the PlatformGeneration script which intended to randomly generate platforms selected from an array of prefabs (platform prefabs). Their spacing was also considerate, using the previous platforms length to determine the gap needed for in between each prefab.**

**I then tried to create an ‘object pooling’ method, which is highly looked upon in the Unity community, according to the forums. I had great difficulty with the object pooling script, it kept demanding that I didn’t have an object set to an instance of an object, after solving this issue I ran into more issues. I remember having great difficulty with this method or randomly pooling platforms in the scene and this cost me a lot of time, approximately 2 weeks. The object pooling method didn’t go to plan because within the first 15 seconds of the game being played it would crash, this was due to the method destroying the array index of the platform and being unable to generate it again when its time came. After failing to do this, I decided to completely change plan and manually insert the platforms instead.**

**Making an infinite runner game versus a finite runner meant that I had to randomly spawn traps and obstacles, which also meant I had to try and determine their starting positions on the Y-axis is accordance to the platform that has been randomly generated. After struggling to search for a solution to this problem and confiding in friends with programming experience, I determined that this would be of a far too much time-consuming effort and would delay me in creating a satisfyingly playable game.**

**Inserting the platforms manually on each scene allowed me to have much more variety within the game. This allowed me to have each and every platform different to another if I wanted to do so as compared to having prefabs placed into an array with an automated platform generation system. Each individual platform was able to have it’s only mechanics and physics, this prevented the user from being able to pre-determine the abilities of each one, being shocked and challenged with every step of the way.**

**The animator sometimes gave me a hard time only due to the fact I was unfamiliar with some key details. Such examples were when I landed the player on the ground that the player was still in the previous animation state. After trying to determine the problem via the scripts, I came to a realisation that the scripts weren’t the problem and in fact the animator was. The animator has some keys features and one of them is the ‘exit time’ that only after awhile I noticed to be the issue. I didn’t want a delay in the animation transitions. After searching and asking on forums what my issue was, I was suggested to change the exit time to zero. As soon as I did this, I realised that this was the solution for all of my unwanted delays in the game animations, therefore, I went back on every animation (jump, walk and idle at the time) and set the exit times to zero.**

**I have a few failures in my game, the main one being that the restart button upon death is not working although I have tried a few different methods of re-loading the scene. Update, the start, exit and restart buttons are now working. The problem I came across is that I was unable to simply add a script to the buttons in the UI’s. I had to create a game object that held the scripts, I would have to drag this game object onto the button and then decide which method I would like the button to run OnClick().**

**Testing and evaluation:**

**I basically had two different games to test, the first & the original one-scene basic infinite runner and the second version, which was a more detailed and difficult to complete finite runner.**

**Feedback for the first infinite runner:**

* **Game was too plain/boring.**
* **Character lacked animation.**
* **No desire to want to play the game.**
* **Game had no story line.**
* **The colours of the game were nice, very vibrant.**
* **Was too linear in terms of movement/running.**
* **There was no theme to the game, the graphics were just inserted because of the asset store assets.**
* **Mechanics were of a poor quality.**
* **No flow to the game.**
* **Inconsistent game.**

**The first version of the game I created seemed to [in my opinion and everybody else’s] lack overall enthusiasm in gameplay structure, I think this was reflected upon my passion towards the game. I usually lack imagination in everything so starting this game was very difficult for me. I was unable to see the ‘big picture’ and was only adding little bits at a time to the game, which resulted in a very linear and repetitive gameplay to be produced.**

**It was commented by one member of my family that the game has no structure, no backbone because it lacked originality. I think what was meant by this was the fact that it seemed that I ‘threw’ the objects onto the scene very generally, in terms of that if I needed a platform, a platform would be placed… there was no before or after thought in terms of style or gameplay structure.**

**A few users stated that they were a bit disappointed because there was lack of animation in the character. Jumping, running and staying idle was a bit too boring for them, I understand this as different games such as Sonic have a constantly moving character. Therefore, I decided that it would be best for my character to be constantly moving to create a more difficult and exciting gameplay structure, however, instead of constantly moving forward, I wanted to make my character constantly move backwards which meant that there’s a higher chance of hitting the saw behind the player if the user remains idle or doesn’t move forward in the game as fast as they should be, this is to make it ever so difficult.**

**After the overwhelming amount of negative feedback by honest friend and family members, I decided I should re-evaluate my concept of the game and possibly restart. After intensive research online to try and find some inspiration with other endless runner games, I was quick to pick little ideas here and there to allow me to be comfortable in completely restarting from scratch.**

**Feedback for the finite runner (restarted version):**

* **The gravity/player mechanics didn’t feel authentic (this has been fixed now).**
* **Each scene has a different theme – very enjoyable.**
* **Two different characters give excitement to the game.**
* **Animations are better than the previous version (infinite runner).**
* **More objects add variety and prevents boredom.**
* **Comparable to current games on the market.**
* **The health UI is very accurate and realistic, the amount of damage received by certain objects seems very adequate.**
* **‘The floor is lava’ is very humorous to some individuals.**
* **No audio caused extreme dullness to the game.**
* **Character animation is slightly off in comparison to speed of the movement.**

**After examining vigorously into other endless runner games, I came to the conclusion that the map would be better generated by myself if I were to manually place the platforms in the scene.**

**I thoroughly searched through the asset store to find good objects and sprite sheets for the map creation. After inserting the sprite sheets into the scene, I soon started to find myself lacking in ideas again, however, this, I realised was due to not many PNG objects in the sprite sheets that I downloaded from the asset store. I was given advice from a friend to search elsewhere other than the asset store and to simply Google ‘Royalty free Unity assets’. This worked to my advantage because I was able to come across a website with many free assets that were also royalty-free, even for commercial use. After finding many sprite sheets I was still slightly unhappy because they all seemed completely different to one another and I didn’t want to yet again to create an unorganised looking game, that is, until I was suggested by a friend to create many different themed scenes. This was the best advice I was given thus far. I immediately began to generate a few scenes using the multiple sprite sheets I downloaded from online. Once most of the images were in the scenes, everything seemed a bit more organised and I started to get the real-feel of the final outcome of my desired game. As time went on I generally built up better scenes because I could easily add platforms and decorations where I felt was best needed, sometimes this was extremely apparent during play testing the game myself and with others, for example, the user was unable to reach another platform because of the distance, therefore, another platform was added as the in-between hop-on hop-off helper platform.**

**Another great suggestion from many people was to add sound/audio into the game, as this gives the game a better sense of real-time play because of the reactions to the user’s actions on the keyboards, such as an audio being played when the player become grounded after being airborne.**

**One detail that I seemed to have been oblivious to and two friends noticed was the fact that the character in-game was moving faster than the animation seemed to have been playing. What is meant by this is that the animation for the running should show that the character’s legs should be moving faster in relation to it’s current moving play speed. I believe that this is a high attention to detail discovery by two friends who have noticed it, as I or others didn’t notice this key aspect. I understand that this may be as important as anything else in the game is because my end vision is to make a smooth game which the user may be indulged into. I solved this by setting the frames of the animation to a higher rate in the animator, fixing this visual error created a better running effect and added to better realistic animations in my game, which improves overall gameplay.**

**As I was inputting different obstacles in the game, I started to realise that many of them included just running through or jumping over. I noticed this during watching others play my game and it made me bored to watch the user only playing with the arrow keys over and over again. Therefore, it came to my initiative to create objects which are interactable with the user such as shooting at to destroy it and sliding under. I got this idea from thinking that the player is constantly being damaged, pricked and prodded at, yet the scene isn’t, it wasn’t fair that it wasn’t vice versa with the damage… so this is where the idea of destroying objects in the scene and avoiding them in a more interactive way should be achieved to add further enjoyment into my game. Luckily for me, the character sprite sheet that I already had downloaded included sliding and shooting sprite images to be sliced up and used readily in my scene. All that I was to do was create animations in my animator and marry it together with my scripts, setting keys during KeyDown to enable the animations to become true.**

**My initial decision was to create two different characters so that the user can decide which character to choose to play with throughout the whole game. I realised that this could be set up in the menu options however, this proved too difficult of a task for me in the current moment. I made a wise decision to just manually set the character skin for each scene and possibly come back to this situation of choosing a character at a later date… this is because I didn’t want to cause any delays in the production of my game. Unfortunately, I was unable to incorporate this although I really desired to make this option to choose the character at the start of the game a reality because many of the best games out there in the market have this option and I feel that my users would enjoy to decide. However, I also feel that enabling multiple characters during the game throughout different scenes gives a nice surprise to the users, even though this may be distracting to some because they may be used to looking for the character skin that they were playing in the last scene… this may cause users to have accidental deaths upon the scene load, which is not something I wanted to see happen during gameplay.**

**I did have slight difficulty with the addition of enable bullets to be shot from my character once the key is pressed. I already understood that the bullet needed to be set to a prefab so that it can be instantiated when the player shoots, and then destroyed when the bullet hits an object just as it would in reality. Once I implemented the bullet, I realised that it was not how I intended because they bullet would appear from the top of the player’s transform, giving a very silly and unrealistic look. I solved this problem by creating an offset of the bullet to appear from a certain point in the coordinate system in relation to the player’s transform numeri. The bullet was then destroyed within the script once the bullet comes into contact with anything tagged ‘shootable’, this caused the bullet prefab to disappear when it comes into contact with the shootable object and the object is then also destroyed.**

**I was told that the health UI system was very classy and good looking (this was agreed upon everybody that I asked, following this comment), therefore, I duplicated it into every scene. I decided that this was the best option as it really gives some sort of smooth flow to the whole game. Since every scene is different, at least something is similar which makes it identifiable as the same game and gives a little bit of a theme to the entire game as a whole.**

**Many different traps and obstacles were included in the game, which I felt gave the users a sense of variety and surprise. One of the most commented on traps was the frozen icicle which caused the player to freeze for a few seconds in time. At first the icicle was a single small image and the player was frozen for 5 seconds, however, the character was still moving on the scene – users felt as though this seemed like a bug in the game and didn’t seem very visually satisfying. I had to play around with the seconds ‘frozen’ and then I also realised it would be a smart idea to make the trap look visually bigger, as though the character has been frozen inside for some time. This resulted in a more realistic icicle look and users commented that this is a much better system for the trap. I did have one difficulty during the finalising of this trap and that was the fact that I failed to get the trap to be slightly transparent in nature, allowing for the view of the character being frozen inside the icicle – I believe this would have been a better outcome if I was able to succeed in have doing so… now the character only seems to disappear being the icicle whilst being frozen, I’m disappointed in myself for not completing a task that seems to be so easy to achieve – as I have seen this in many other games. Even after intense research, I was unable to make the icicle seem slightly transparent. Some users also told me that the icicle should completely stop the player movement, freezing the x-coordinates, and some users admired the current state whereby the player appears frozen in nature, but still slightly gets moved along the x-axis as if sliding from being frozen. The movement across the x-axis wasn’t intentional. I couldn’t seem to have figured out the problem as to why the character still slid across the platform once being frozen, so I decided to not waste too much time on this particular problem and possibly come back to it later if I had the time. I noted this down for future reference because I already had a few minor problems that I would have hoped to solve. Update… I have now figured out a way to turn the icicle into a slightly transparent image using the ‘colour’ option in the unity inspector, I was unaware of this until today and I am glad I’ve figured this out because it means I can use transparency and colour changing in other objects as well. In particular I have changed the colour of all of the objects and platforms in scene one to allow for a darker theme and for the theme to be more of a similar colour to the character – pumpkin man! Also all of the saws in each and every level/scene have been coloured to create a more balanced theme colour, this also increases difficulty and the visibility of the saw is significantly reduced, as compared to it being of an obvious silver colour.**

**At first the character death was very dull and basically invisible because the character would ‘die’ when out of the camera view, falling way below to it’s death and then the saw followed the character to proceed to ‘kill’ it. I and every single user that play tested my game didn’t like this idea at all and it was compared to other games on the market, because other games on the market have visible deaths for the characters. I must have been slacking on the game because this is a crucial concept within the game – the death of the character. After viewing a few different games to examine how the characters die, particularly in Sonic, I wanted to create something that’s not been done (at least not commonly executed). The floor is lava was completely my idea and it was derived from a childhood game, that I’m sure everybody knows about. The lava was a PNG from an already downloaded asset, I just combined this with two box colliders, one OnTrigger (trigger to enable the collision of a tag called ‘hell’, which causes immediate death upon contact) and one not triggered (not triggered to enable to player to be held unto it). This should have been something that I should’ve completed from the beginning, as it’s one of the most simplest tasks to do, however, at least it’s done and dealt with now – me and my play testing users are happy about this and I’ve gained some giggles from ‘the floor is lava’ approach.**

**An idea I was thrown by a user who play tested my game, was to have the health be the same for every character throughout all of the scenes. What was meant by this was that if on the first scene the player ended with a health of 70, that the second scene would start with the health at 70 and so forth for all of the scenes. I thought and still think that this is a good idea as well, as it caused the player to be more careful in case complete death would occur and result in starting the game over entirely. After researching into the occurrence of shared health in between levels/scenes, I was constantly bombarded with the use of the word ‘static’ and that it would apparently solve all of my problems… This was very wrong. I tried to use the word static for the CurrentHealth value and/or the HealthBar value from the Health script, but neither situation resulted in the solution I was after. After constant battling to try and figure a way to use a shared health script and spending an entire day dedicated to solving this matter, I decided to give up and move on.**

**Conclusion:**

**Successes: The success of my overall game is that it’s playable and compiles. There aren’t many errors, if any that I’ve noticed… I have taken off ‘pause on error’ so excuse me if I haven’t noticed an error.**

**The game runs smoothly until a certain point, which the characters mechanics and animations in good order. The overall game is enjoyable for all users and high interactivity is involved, enabling fun for everybody who is willing to play it!**

**Overall, the first two playable scenes are a success in my opinion because they set out to do exactly as I desired from my original plans. I would like to have solved some errors/bugs in the game, but the hardest part of my games are complete and I’m satisfied with the outcome. The play testers were also mostly happy to play with the end result.**

**Failures: From many of my failures I have come back to them and taken this as a lesson learned, for example with the prefabs not being centred, I remembered to centre all of the prefabs from then on to avoid errors in visibility of the objects.**

**On build and run however, this is a different situation, from the third scene there seems to be major problems, whereby the character is invisible, yet it’s still able to be moved with the controls. I have tried to search upon this problem in forums and with others have a similar situation happen to them, however, I cannot seem to resolve my issue at all.**

**Trying to add audio was a complete failure me as I tried many times but even to add background audio in my games cause issues such as compile errors or distorted noise.**

**Others vs me: I haven’t seen any executable files of others so I cannot compare if they have any errors in their work. From playing in the play testing sessions, I can determine that others may be ahead of me in terms of overall gameplay. I have vastly improved my game since the play testing session, however, I still believe that others may have done more than me till date.**

**I know somebody that has a very interactable menu section, where they can adjust the volume, to me this adds an aspect of higher quality and thought to their game.**

**What I could have improved with more time: With more time I could have added more to the menus sections in terms of decorations and compatibility. I could have fixed the build and run issues if I was able to have more time. I am slightly disappointed with myself in terms of the game not being fully complete. Unfortunately I ran into problems that caused major delays in both aesthetics and game mechanics, causing a complete and total restart of the whole project, when I should have been a third way complete.**

**Bibliography**

**Start menu tutorial:** [**https://www.youtube.com/watch?v=zc8ac\_qUXQY**](https://www.youtube.com/watch?v=zc8ac_qUXQY)

**Particle systems for the blood splattering and snow:** [**https://www.youtube.com/watch?v=vwUahWrY9Jg&t=0s&list=LLlBeKUr6\_hkc6wQw5Lp0Mmw&index=1**](https://www.youtube.com/watch?v=vwUahWrY9Jg&t=0s&list=LLlBeKUr6_hkc6wQw5Lp0Mmw&index=1)

**Moving background:** [**https://www.youtube.com/watch?v=HrDxnMI7pCc**](https://www.youtube.com/watch?v=HrDxnMI7pCc)

**Unity scripting, major resource for the use of IEnumerators:** [**https://docs.unity3d.com/ScriptReference/**](https://docs.unity3d.com/ScriptReference/)

**Character sprite sheets:** [**https://www.gameart2d.com/freebies.html**](https://www.gameart2d.com/freebies.html)