Escape to Space

Team 13

Aaron Morris, Omkar Chekuri, Yonathan Hendrawan

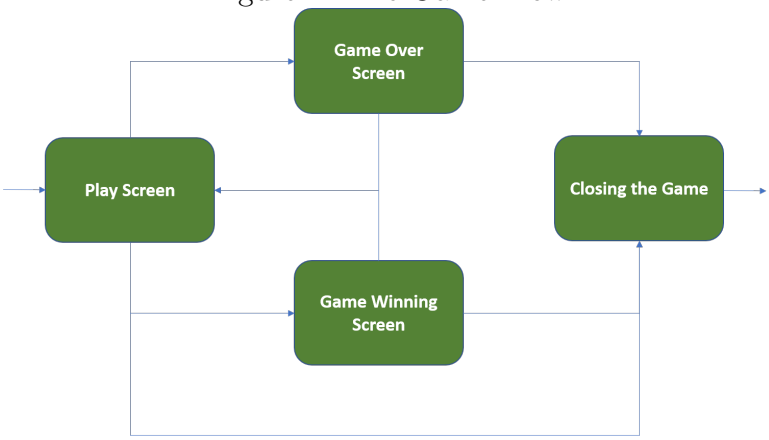
Introduction

In today's world most people have more free time than previous generations. For some people this amount of free time can be frustrating because they become bored without having something to do. Boredom is a problem that can exacerbate mental health problems or even cause some such as depression. People often turn to products such as games, movies, TV shows, cooking, sports, etc... to fight off boredom. As long as people have free time, there will always be a need for products such as games to help prevent boredom. Our product is a 2.5D platformer game that is meant to provide entertainment and help prevent boredom. In our product, a user controls a person and must make it to the end of a level by jumping from platform to platform while avoiding dangerous objects. We have successfully created an Alpha version of this product. We were not able to implement all of the features we wanted to sadly. For example, we were unable to add weather. We did however complete a fully functional 2.5D platformer game with animation, collision detection, shaders, a particle system, and some simple physics. We currently enjoy playing ourselves. We believe that it can already help fight boredom in its current state but could be much more successful if fully completed.

Example: Game Walk Through

This section describes how our game works. It is divided into four states as shown in Figure 1. We will explain on how to reach each of them in the following sections.

Figure 1: The Game Flow

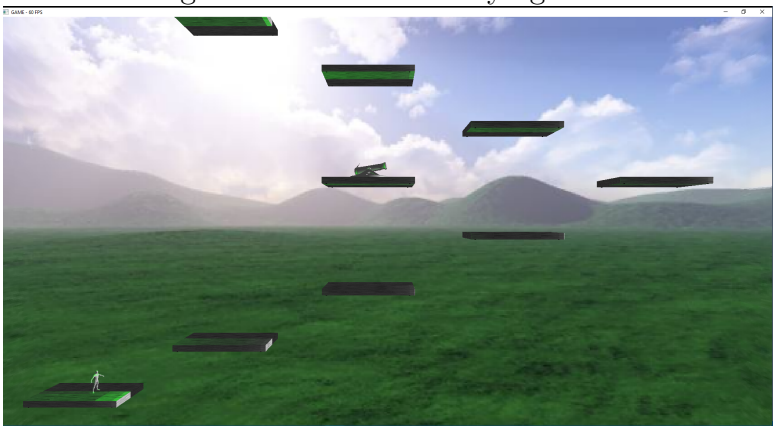


Play Screen

This screen is the entry screen of our game. It shows a human model standing on the first platform at the bottom left corner of the view, as shown in Figure 2. The platforms are arranged at different heights and positions. There are obstructions in the form of cannons that fires projectiles. The game has three levels.

The player uses the keys "A" or "Left Arrow" to move to the left direction, and the keys "D" or "Right Arrow" to move to the right direction. Pressing the "space" button lets the player jump in the air. The direction buttons can be pressed in combination with the "space" button to get the combined effect of travelling in air in either left direction

Figure 2: The Game Playing View

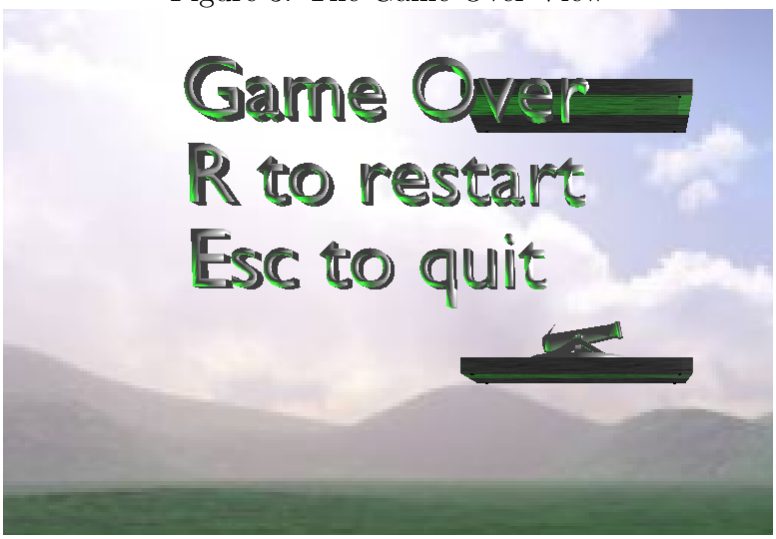


or the right direction. Pressing the “R” will restart the game and pressing the “Escape” button will end the game and close the terminal.

Game Play

The objective of the player is to jump on different platforms and move in the upward direction to reach different levels. The player should avoid the projectiles from the cannon.

Figure 3: The Game Over View



Being hit by the projectile will kill the player and end the game. In such event, the game will show the Game Over Screen with a message prompt: “Game Over. R to restart and Esc to quit ”, as shown in Figure 3. Another cause for ending the game is by falling from the lowest platform. It is still fine if the player does not land on a platform while jumping, depending on where its current location is. If the player is not on the lowest platform, he still has chances to use the control buttons and control the trajectory of the player falling to land on the platforms below him. If there are no available platforms below him or he is unable to land on the platform before crossing the first platform from the bottom, the player will be dead and the game ends.

If the player manages to successfully reach the top most platform while successfully avoiding the obstacles, the game ends with showing the Game Winning Screen. It has a

Figure 4: The Game Winning View

message prompt "Victory! r to restart esc to quit", as shown in Figure 4. On that screen, the player can either choose to restart or quit the game.

Process

We developed our game by dividing it into several tasks based on software engineering processes: Requirement gathering and analysis, Design, Implementation, Testing, Analysis, and Reporting. Instead of using the processes closely, we split and grouped them into six milestones: building the game framework, creating an alpha version, completion of individual facets, integrating the individual facets, testing, and finally reporting.

The milestones followed a linear process, except for the third milestone, which could have been worked on without needing the completion of the first and second milestones. Our first milestone was building the game framework that we would use as the foundation of our game. Without the framework, the game could not have been developed. The Requirement gathering and analysis process and some of the Design process were performed in this in this milestone. Our second milestone was creating an alpha version of our game. This alpha version was built on top of game framework we developed with the help of a free online book called *3D Game Development with LWJGL 3* (it can be found here `https://lwjglgamedev.gitbooks.io/3d-game-development-with-lwjgl/content/`). This milestone included part of the Design and Implementation processes. The third milestone was completing our individual components: weather system, firing cannon, and confetti particles. This milestone was apart of the Implementation process. The fourth milestone was integrating the individual game components with the game. This milestone was the final step in the Implementation process. The fifth milestone was the testing of the game and its performance. This milestone completed the Testing process. The sixth and final milestone was creating the project report and presentation for wrapping up the project. This milestone completed the Reporting process.

Planned Vs Actual Timeline

Milestone	Planned Completion time	Actual Completion time	Description
1	Week 8	Week 13	Game Framework
2	Week 9	Week 14	Alpha Version
3	Week 12	Week 15	Individual Facets
4	Week 13	Week 15	Integration
5	Week 14	Week 15	Testing
6	Week 15	Week 15	Report and Presentation

As mentioned in our progress report, we had not been able to finish the first three milestones on week 12. Our actual completion time is shown on the table.

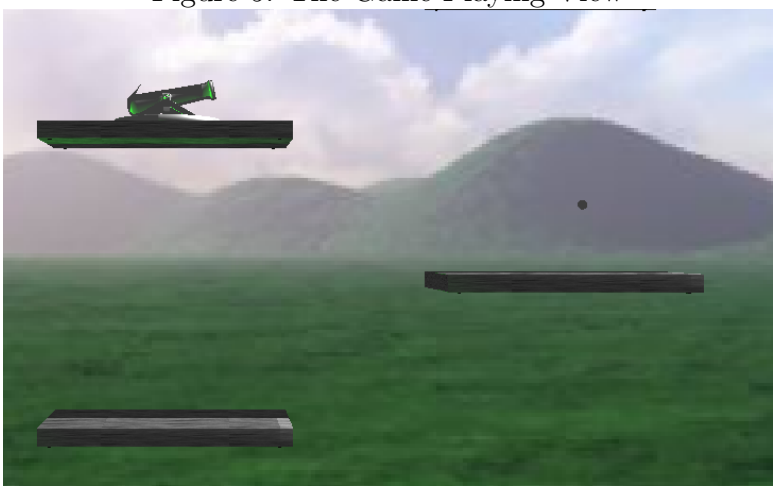
Major changes

We made a switch from "JOGL" library to "Lwjgl" library as it is more tailored towards building games. The Framework was very challenging and took longer than expected as a result we did not implement some parts of the facet like dynamic weather, explosion, and shader-side confetti. We implemented some things that we did not talk about in the original project plan, such as animation effect while the player is walking and collision detection.

Detailed Results

Our product is a 2.5D platformer. All of the models are 3D and the camera is positioned to provide a side view of the models. This creates the feel of a 2D game made with 3D models. Below is an example screenshot of this feature from our product.

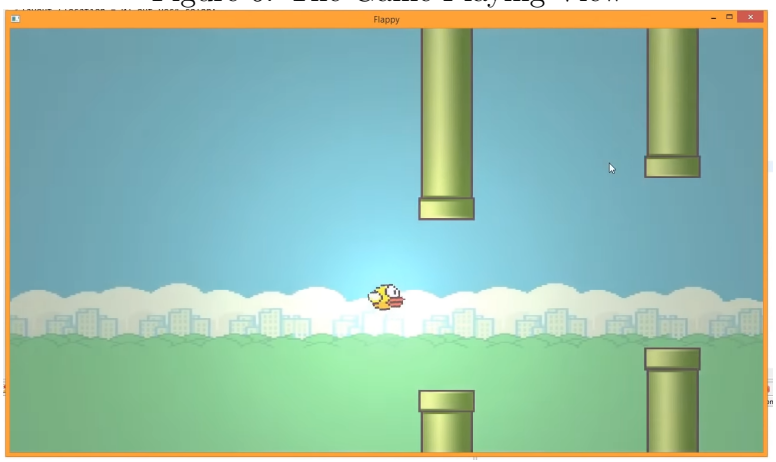
Figure 5: The Game Playing View



For comparison, compare the above image with the image of the 2D flappy bird clone below.

Creating our product as a 2.5D platformer allowed us to provide more features than a traditional 2D platformer. One of the features gained was the ability to create and use more realistic models. The use of these models adds a more realistic feeling to the game. This in turn will allow for better immersion. Another feature we gained from our the

Figure 6: The Game Playing View



2.5D approach was more accurate lighting and shadows. Using a 3D world allowed for the lighting and shadows to be more similar to real life, and also allows for them to be more accurately fine tuned. A 3D world also provides the feature of more realistic physics. While we do not have a high amount of physics or extremely complicated physics in the Alpha version, the physics we do have were still enhanced. This feature will be extremely important and helpful for future versions of the game.

Our product has more features than those provided by using a 3D world. Our product is a game that has collision detection, simple physics, shading, lighting, shadows, animation, and a particle system. Collision detection is a feature that is used to determine when the player is standing on a platform or if a player has been hit by a cannon ball. Bezier curves were used to provide the physics of cannons shooting cannon balls . Shading is used for many things ranging from applying textures to shadows. Lighting and shadows are added to all of the models in the game by the game engine itself. This provides a more realistic feeling. Animation was used to simulate bouncing like physics and velocity for the cannon balls, and to make the human model look as though it is running. A particle system was used to create confetti that is shot out of the final platform when the player wins the game.

Currently our product is functional but not the most appealing to the eye. This is because it is in Alpha. Our goal was to design the world and game play mechanics with basic models that textures and shaders could be easily applied to later. For this reason the majority of our models do not have any textures and the lighting needs some work. With this design, we were able to spend more time polishing the game engine, mechanics, and features. The appearance of our product can easily be improved by adding textures and adjusting the lighting.

Our product behaves very similar to other platforming games. The player must jump from platform to platform while dodging projectiles until the end of the level without missing. If the player misses a jump or is hit by a projectile the game is over. Our product also creates realistic lighting and shadows.

All team members had access to a Windows 10 machine, so Windows 10 was used for the development of our product. In order to construct our game we used a couple

of libraries and tools. Our game relies on two libraries, a Java OpenGL wrapper library known as LWJGL (Light Weight Java Gaming Library) and a Java OpenGL math library known as JOML (Java OpenGL Math Library). The LWJGL library was used to create all graphical aspects of the game. JOML was used to perform translations, bounding sphere calculations, matrix multiplication, and many other linear algebra operations often used in computer graphics. We used several tools for this project. We used the Eclipse IDE to edit, debug, and run our code. We used Gradle to manage our project's dependencies. We used Blender to create 3D models and a blender add on called Riggify to edit 3D models. The Windows 10 performance tools were used to measure the performance of our game. We used this information to improve our code's performance. Git was used to manage our source code and versions. GitHub was used as our remote repository. The GitHub desktop application was used by two of the team members for Git, and the Git bash from `https://gitforwindows.org/` was used by the other member.

In order to learn the LWJGL and how to build a game engine we used a free online book called *3D Game Development with LWJGL 3* (it can be found here `https://lwjglgamedev.gitbooks.io/3d-game-development-with-lwjgl/content/`). This book and the examples we implemented from this book expanded our understanding of computer graphics. It also made us realize how much work and how difficult it is to actually implement a game engine from scratch. Due to the time constraints of the semester, we decided to build on top of the example code we had been working with. For these reasons our code structure is pretty much the same as the code structure found in the book's example code repository. Our code consists of two packages. The game package and the engine package. All other packages present are strictly resource packages that contain models, fonts, sounds, and shaders. The game package consists of two files, the main file to launch the application and the game file called by main. The game file has all of the game and rendering logic. The engine package contains everything required to create the graphics of the game, and the logic required to create the game loop. Inside the engine package at the top level are the files needed to create the game loop and render the graphics. The IGameLogic file is an abstract file that ensures any game created using the engine can be ran by the game engine and has all the necessary pieces such as user input. The GameEngine file itself contains the game loop. The game loop controls when the input is checked, the scene is updated, when everything is rendered, and tries to maintain a minimum target frame rate. The engine.graphics package contains the java classes required to load the models, the terrain, and the sky meshes from their object files along with their related textures and shaders. The engine.graph.anim package contains all of the java classes needed to perform animation for models. The engine.items package is used to create game, skybox, terrain, and a text objects that can be used to create games. The engine loader packages are used to help shaders and models. The rest of the packages (i.e. engine.graph.weather) are used to create the graphics described by the last word of the package.

In order to play our game users must have access to a computer with Windows 10 that has Java installed and a working keyboard. Users must also be able to interact with the keyboard in some way. Once the game is launched only keyboard keys are used. The keyboard must have working a and d or left and right arrow keys, and a working space bar to play the game. If none of these keys are available then the user will not be able to move and jump in the game. It is also recommended to have a working escape key. The

escape key is used to close the game, but if a working mouse is available it can be used to close the game as well by clicking on the x icon in the top right corner.

Future Directions

Due to the time constraints of the semester, the complexity of building a game engine from scratch, and the complications caused by COVID-19 we were not able to implement all of our planned features. One feature we planned to have implemented but did not fully complete was the application of textures for our models. The majority of our game models are just basic models with one solid color. Another feature we planned to implement but failed to was the weather system. The weather system was supposed to be capable of creating sandstorms, snowstorms, sunny days, and thunderstorms. The idea behind this was that each storm would change the visibility and lightning of the map to increase or decrease the difficulty. We also did not complete all of the performance measuring features. We also did not manage to implement the shader-side confetti effect as planned. Next, we did complete the FPS measurement, but did not implement features to measure main memory, CPU, and GPU usage for our game. We were originally planning on these features but discovered that Windows 10 has some built in tools that can pretty accurately measure this performance for us. The final feature we failed to implement was explosions for the firing of the cannons. This feature was meant to give the cannons more realistic behavior and animation. Therefore, we deem these unimplemented features as our future works.

A fun and interesting feature to add would be a race mode that allows 2 or more players to race to the end of the level. Who ever reaches the end first wins. This would be a challenge to implement but it would bring a new level of play-ability and fun to our game.

The potential directions for expansion of this project are endless. New levels, projectiles, models, interactive objects, and story modes could be added. An online multiplayer or leader board could be created for the game. We could develop a way to for users to create custom levels. These are just some examples of directions the game could go by themselves or in some sort of combination.

Conclusion

In the modern era, people have more free time than ever and for many this leads to boredom. This feeling of boredom can be overwhelming for some people, and they need products that can alleviate this boredom. The goal of our game is to help these people escape their endless cycle of gloom and find enjoyment. Even though we only have the alpha version of our game completed, we believe that our game can successfully meet this goal. We are entertained by it ourselves and it is not even completed yet. Once this game is complete it will have even more entertaining parts and will have no problem helping people stave off boredom.

By building this game we also gained considerable experience in game development and computer graphics by understanding the pitfalls in both fields. We can carry this

experience over to build much interesting games and computer graphic applications in the future.

Team Summary

Although we did not have many face-to-face meeting throughout the semester, we worked together helped by online tools. As the repository and versioning system, we used Github to facilitate our development and collaboration. We use "WhatsApp" as our daily communication tool. When we held several meetings throughout the semester, we used Skype and Zoom.

In terms of collaboration, we did not encounter any problem. Each team member is responsible toward his work and willing to share any challenges faced.

Aaron

Aaron's role was to create the game engine, dynamic weather, and integrate code changes made by Omkar and Yonathan. Aaron with the help of the book mentioned previously, was able to create the game engine and the very first alpha version of the game (Alpha 1.0, we are on Alpha 3.0 now). Due to the complexity of creating a game engine from scratch Aaron was not able to complete the dynamic weather portion, which was his individual facet. Aaron did however have to implement collision detection and animation from scratch for this game. This was a unique and somewhat difficult part of the project that Aaron would like to make his individual facet instead. Aaron spent several days alone on creating successful collision detection using bounding spheres and rays and fine tuning it. Aaron was also able to successfully merge all facets, branches, and code changes together with the use of Git. Having one person manage this made it a lot easier. Aaron volunteered because he created the engine and felt very confident in his version control skills.

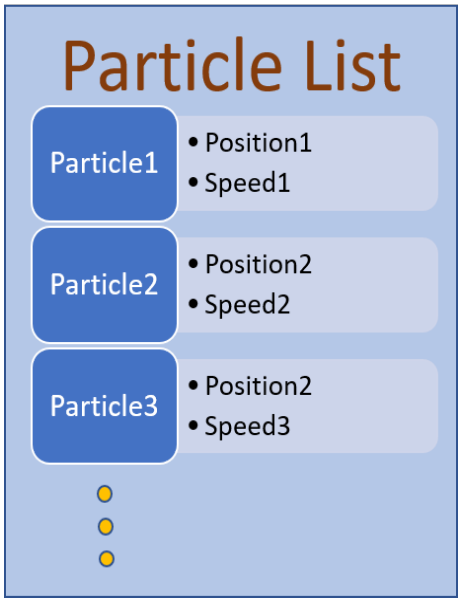
Yonathan

As stated in the project plan, Yonathan worked on confetti particles. The architecture is illustrated in Figure 7 and described as follows. Our game has an array list to maintain a group of particles. Each particle has its own attributes, such as position, speed, size, etc. Using this method, our group can control the number of particles, its individual speed (with a random number added), and its unique cubic Bezier path generated randomly in its initialization.

Compared to the particle system built in the progress report, integrating the particle system into our Alpha version involved a significant work, starting from managing the resource files, folder structures, and integrating different framework parts: coordinate system and animation methods.

The confetti effect is mainly done in Java. Although the program uses shader capabilities, they are of the simple ones: multiplying the vertices with the MVP matrix, passing the texture coordinates, and doing simple lighting.

Figure 7: The particle system architecture



Omkar

As per the project plan, Omkar modelled the cannon and projectile as two game items using Blender software. The Cannon is being rendered using a method called instance rendering which involves creating instance of a game item and rendering it in multiple locations. The path of the projectile is calculated using Bezier curves for all the cannons. We began implementing the cannon projectile as a particle system in the alpha version where we generated multiple instances of the projectiles but later changed to implementing it as a regular object in the Alpha 3.0 version without the particle system due to issues occurred while integrating the facet. Omkar learned about programmable pipeline and shader programming to implement vertex and fragment shaders in the Alpha Version of the game where the projectiles are implemented as a particle system. Since the game is very less CPU intensive and does not require much CPU and GPU utilization, no decline in performance is noticed using the windows based performance measurement tools. Omkar implemented a fps calculator that measures the frames per second and displays it at the top of the game.

Since we started building the models at the beginning of the semester where we did not learn the concepts of building 3D models. To get our game going, Omkar decided to learn how to use open source tool called Blender and created cannon, human walking poses (using method called rigging where the base human object is aligned to skeleton and create the necessary poses by manipulating the skeleton), and extruded text objects that display Game Over and Victory in the blender.