Game Project Report

Team #12 Members: Jason Gates, Zhong Cai, Leonel Paredes Barrera and Stanley Yang

Date: 11/28/2022

CSE 165 Fall 2022

Introduction:

In this project, we created a level from the game called “The World's Hardest Game”. This game is a simple 2D flash game where you avoid obstacles or blocks using movements. To win, you need to reach the end goal or reach multiple locations to unlock the end goal. This game is notorious for being difficult and requires you to be precise with movement. In our version of it, we have the player, represented by a yellow square object, that needs to avoid the autonomous moving green square objects, reach the cyan square parts of the map and then reach the blue square object which is the goal. The cyan square parts represent prerequisites that need to be reached before going for the end goal. The end goal will not unlock if they aren’t reached by the player. When a player makes contact with the autonomous object, this indicates death for the player, and they are simply sent back to their original spawn point and the death counter is updated. This game is built in C++ and uses the Simple and Fast Multimedia Library(SFML) library that lets you use 2D Graphics with OpenGL and can use different types of media such as fonts, audio, and images.
The fonts, audio, and images from this project are from SFML. The game is compatible and can be playable for both Windows and Linux systems.

Motivation:

We had two motivations for this project. Our first motivation was to make a fun and simple game that doesn't have complicated or complex rules. We wanted any person regardless of skill or experience with gaming to be able to play and have fun with a game. One of the downsides of games like chess or backgammon are that they aren't beginner friendly and players need to learn and invest their time into learning a multitude of rules. In our game, we planned on making fewer rules and simple rules to make it easier for newcomers to learn our game. Additionally, making our game simple made it easier for us to explain our game to others during the presentation. If we had an abundance of rules, it would've made it harder for others to understand our game and most likely left them confused. We were also constrained to 5 minutes for our presentation and making a simple game with fewer rules made it easier for us to explain the game and not go over the 5 minutes range. Unlike "The World's Hardest Game", the difficulty of our game is easier, rather than being extremely hard. We designed the movement of the autonomous objects to be fast, but not unfairly fast to appeal to casual players. This was to make the game more fun for a broader audience. However, the difficulty can still be enjoyable to skilled players who seek a challenge, since the game can be hard. Overall, we tried to design the game for casual and skilled players to have fun.

Our second motivation was to get experience in game development with C++ and other libraries. At UC Merced, CSE 165 is one of the few classes that let us develop a game as a project and allow us to have creative freedom in how we design and develop our project. We were enthusiastic to develop a game since little to no classes at UC Merced have game development. Each of our members was interested in how game development works and

doing this project helped us learn more about C++ and how C++ and its libraries related to game development. We were exposed to many different C++ game development libraries like GLUT, SOIL, SFML, and irrKlang, which we had to learn how to install and how to use. Without this project, we wouldn't have learned about how game development and C++ work together and the many different libraries that C++ has.

Methodology and Implementation:

When we first started this project, we originally used the OpenGL C++ library called GLUT. For our group, we struggled to understand and implement a game using GLUT. This is because there was a huge learning curve for understanding GLUT and the documentation was often unhelpful or hard to understand. After slow progress on our game using GLUT, we decided to use the C++ library called Simple and Fast Multimedia Library(SFML). Using this library helped speed up the development and progress of the project because of its easy-to-understand documentation and library functions. We started the setup process for SFML by downloading the SFML library on the SFML website. Next, we dynamically linked the files and Visual Studio. This makes sure that Visual Studio can find the library and use a specific path to find the location of specific library files. This was done by editing the properties of our Visual Studio project. Lastly, we copied the library files from the SFML website to the project folder. To complete this game project, we needed to decide what each group member would do. In our group, Stanley, Leonel, and Zhong focused on wall collision and object collision. Jason and Stanley worked on player and object movements. Jason worked on features like the death counter, spawn point, music, and end area. In order to achieve the goal of completing this project, we used time management skills to organize times when each of us were available and used those times to work on the project. We also

used group chats, and calls and sent updated files to each other. We often met at the library to discuss and work on the project. This shows our different methods of communicating with each other and how we finished the project.

In our project, we created a square player object, a square goal object, three square checkpoint objects, and eight square bot objects. We used SFML's various functions to adjust their color, position, and size like setPosition, setSize, and setFillColor. We also used classes and OOP to help adjust positions for the player and bot positions and shape sizes for the bots and checkpoints. The OOP that was specifically used was inheritance and polymorphism. This is shown in the object.h file. We also created music objects for the built-in music functions like music.play() to play background music. The music was designed to loop so that there isn't any awkward silence while playing the game. Once the player wins the game, the music is stopped and then another music object plays victory music for the victory/game over the screen. We also created a texture object that is used to help load an image to the background of our game. We also do this process for the victory/game over screen. The game over/victory screen is shown when the player touches the cyan squares, which act as checkpoints or keys to unlock the goal, and then reach the end goal square. Whenever a player touches the cyan squares, it makes certain boolean statements in our code true. This indicates that the player has touched the square and can proceed to the others. The goal becoming available is an if statement that only executes if three certain boolean statements are outputting true. The game over/victory screen automatically closes after 10 seconds to indicate the player has beaten the game. The game over screen, music, texture, and death counter is a part of the main() code which is in Source.cpp. However, the death counter also uses a class and OOP in the object.h file to update and print the number of deaths. OOP objects similar to the object that gives access to the functionality of the death counter are used

widely across the main() code and are essential for making the game work. If a player touches the moving square bot object, they are sent back to spawn and have a death updated to their death counter. The class also has a constructor and a destructor. We also have calls to classes that have functions that correspond to both bot and player movement. We have a normalization function that uses vector calculus through normalizing the vector and is used primarily for bot movement. The normalization process uses the non-zero vector and divides it by the norm or magnitude of the vector. This process is used to guide the direction of the moving object toward the top or toward the bottom of the boundaries. Additionally, we altered the velocity for a vector in the normalization function to make the bot object movement faster. This was in order to make the game more difficult for players and encourage players to play carefully. However, the velocity value was tuned to fit the playstyles of both casual players and skilled players. The class with player movement uses the Keyboard::isKeyPressed to tell if a certain key was pressed. The keys correspond to WASD and are used to move up, down, left, and right. The velocity of each directional key press can be adjusted through this function. We decided to make the velocity slow to appeal to casual players so that they can take their time and make precise movements to reach the goal. By editing velocity values, you can make the player object go faster or slower. The movements are present in the movement.h and movement.cpp codes. We also implemented a pause function and escape function. If a player pressed the escape key on their keyboard, it would close. If a player pressed the letter p key on their keyboard, it would freeze all objects in the game for 10 seconds. We used the Sleep() function to do this. We also use window.draw() to render all the objects like bots, checkpoints, boundaries, background,text, and the player to the screen. Without this, nothing would appear on the screen which is why it's very important. The last essential SFML functions we used were getGlobalBounds(), intersects(), and getPosition(). getPosition() and getGlobalBounds() helped us track the vector

coordinates of all the objects in the game. This is used for movement, object collision, wall collision, and boundaries. Intersects() is used for finding if a player touches a cyan square or touches the goal point. It is also used in collision calculations.

The key component of the project was object collision, as all objects in the game involve collision. The way collision worked for the walls was that the wall would check in all directions if an object was in the position right next to the wall. If an object such as the player came next to the wall, the object's position would be set as the position right next to the wall. The bot collision involved a similar concept, in that the bot would check if it had another object next to it. If the bot did indeed have an object next to it, the bot would change directions. This is the reason why the objects move in a vertical motion once collision occurs. Player-to-wall collision and bot-to-wall collision is essential to our game because it restricts the gameplay area and prevents players from going out of bounds and breaking the game. If there were no wall collisions, the player can easily avoid the moving objects because there is more space for the player to traverse. This results in the game being substantially easier and hinders our goal of wanting to make a game that both casual and skilled players can enjoy.

Outcome and Evaluation:

The outcome of the code shows that the game perfectly works throughout the entirety of the level, as shown in the screenshots below. The movement and collision work as intended. However, we came across a problem where the player may be able to leave the box by moving in a certain method but are unsure how to fix it. Although there may be some issues with the game, we have achieved the basic intended outcome of this project by creating an object that is controlled by the player, an object controlled automatically through code (for our case, we had multiple objects), and implementing concepts learned throughout the course which includes classes, inheritance, and polymorphism. The bugs in this game don't hinder the experience of the player and are hard to find. Our end result was a working level from "The World's Hardest Game" with complex collisions, wall boundaries, player movement, autonomous moving bot objects, creation of objects and object interactions, music, death counter, pause and escape function, and a game over screen. We made a game that appeals to both skilled players and casual players and aims to reach the goal of being fun and simple. If we had more time, we would've been enthusiastic to improve our game and add more features that were scrapped like more levels, more enemy bot variety, different backgrounds, different music, sprites instead of squares, sound effects, map variety, timer with a scoreboard and player customizability.

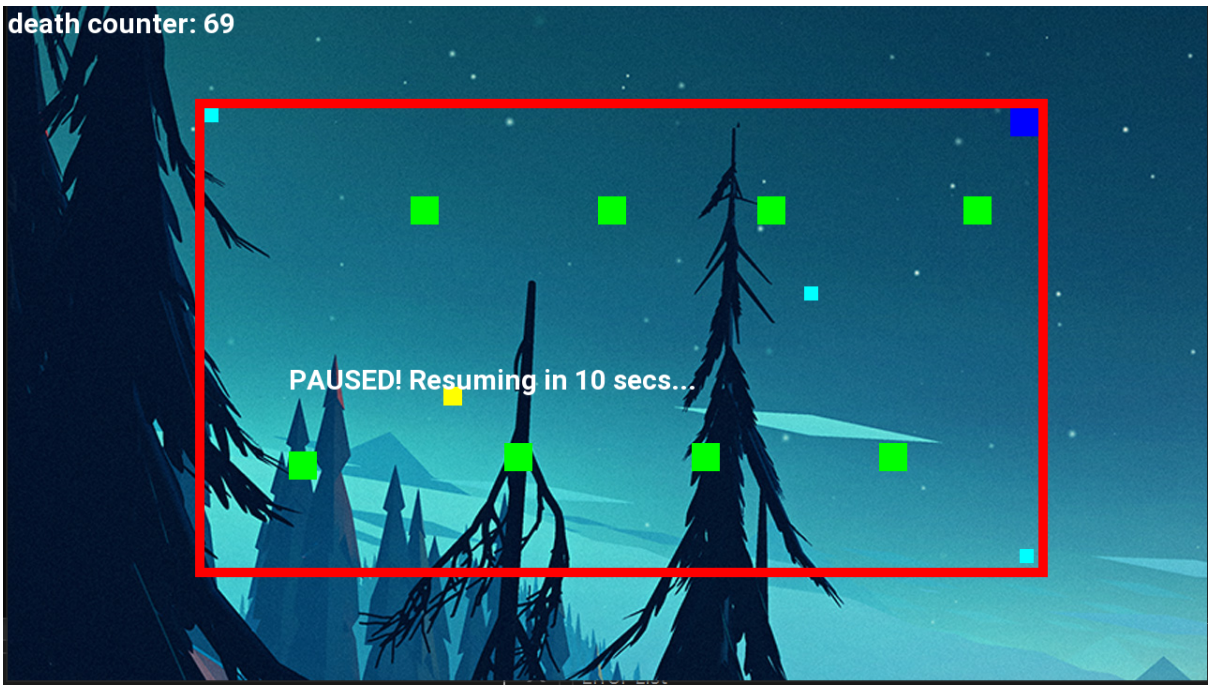
Conclusion:

The idea for this project was to create a level from "The World's Hardest Game", which is a simple game that utilizes the movement of the player to avoid moving blocks to reach the end goal. Some of the features implemented mimic the game with player movement, autonomous moving objects, wall collision, and a goal area. This project was originally started using GLUT but was eventually moved to SFML due to the complexity of GLUT. Throughout the development cycle, we implemented concepts learned in CSE 165 such as classes, inheritance, and polymorphism, which meet the goal of the project. Overall, we feel like this project has made us learn more about C++ and object-oriented programming. This project also helped improve our ability to work as a team and improved our ability to adapt and learn things we aren't very knowledgeable about. This project was an excellent introduction to the game development field and in the future, some of our group members might consider using the experience and knowledge required from this project to make a better and greater game.

Supporting Screenshots:

Screenshot 1: *A screenshot depicting the start of the game. The green objects are moving and the yellow object is the player. The cyan objects are checkpoints or keys that need to be reached by the player in order to progress. The blue is the goal point and is the end area of the game. Currently, the bottom left of the boundary is the spawn point for the player.*

Screenshot 2: *A screenshot depicting the player interacting with the last checkpoint or key and can now progress to the goal area. Text on the top left indicates that the player can go to the goal.*



Screenshot 3: *A screenshot depicting the player pausing the game. The player cannot move for 10 seconds and everything on the screen is frozen.*

Screenshot 4*: A screenshot depicting the game over/victory screen that occurs once the player gets all three checkpoints or keys for the goal and reaches the goal point. This screen is accompanied by victory music.*