Project Report

Project Name: An SDL 2D Side-scroller Game

Team Members:

1. Muhammed Tasnim Bin Anwar (SH-05)
2. M Muztoba Hasan Sinha (AE-15)

Course:

Fundamentals of Computer Programming Lab(CSE-1211)

Introduction

The game puts the player in a red plane against the evil boss “Hilda Berg” in the countryside. Hilda Berg has different transformations and to defeat her one has to beat all of her forms.The player has to control the plane and simultaneously avoid different projectiles and obstacles. The player accumulates points based on time alive and the damage he dealt to the boss. The plane is constantly firing bullets to damage the enemy. The goal is to destroy Hilda Berg completely.

Objective

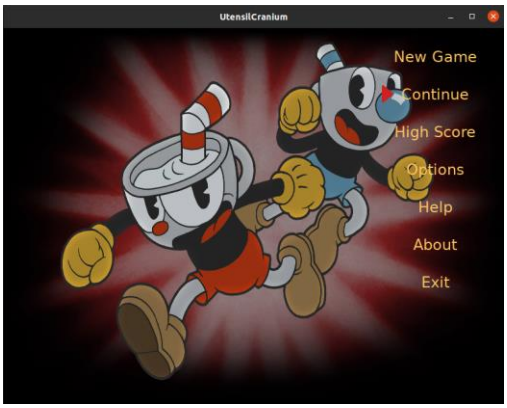
The main objective of this project is to correctly implement the c/c++ language that we learned this semester in some practical field and learn more about the graphical section of c/c++. The game is developed using SDL2.0, thus it also served as a purpose to learn more about SDL. Writing easily understandable code with scope for easy further development by making the code modular was also one of the intended objectives of this project.

In terms of objectives that we set out for the game, our main focus was to make an engaging game. It is proven that games hold your attention when there is minimum friction between the controls and the games systems are fair even if it’s difficult, as long as it’s cohesive and properly communicates with the player it should have ample replay value. So in short we set out to make a game that while limited in its scope and high in challenge, gave the player necessary tools and works in a way that the user intends it to work. So every game action had to be clearly communicated and it should reward skill.

Game Features

The game has general game features like a main-menu, instruction page, options menu to change between different settings, basic movement features, collision detection, high score calculation, score presentation etc. The game also has different power ups, diagonal movement, a skill based doge move, resuming the last uncompleted game, full

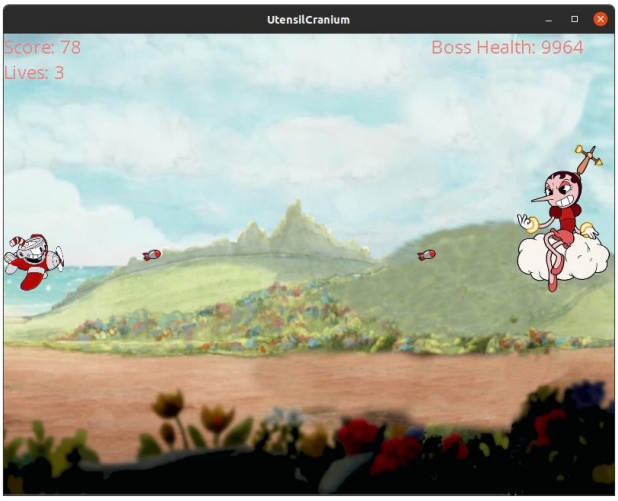
screen, functional screen resizing, mouse-mode gameplay, high score reset options as special features. Below the features are described in short.



Main-Menu Features:

The game starts with a title screen and after pressing any key takes the player to the main-menu which has 7 buttons. The menu is highlighted by a cursor which can be moved with the directional keys and the mouse cursor. The Main menu buttons are described as follows:

New game: Clicking on this button will take you to the game directly with your past progress being overwritten. The player will start with 3 lives and score zero and boss health at the maximum. All three are shown on the screen. At first the difficulty is at minimum and as your screen increases difficulty will increase. The boss can fire two types of projectiles, one is a bomb-like projectile thrown in an arbitrary direction that bounces back from corners. The second one is a homing projectile that tracks the plane and adjusts it's path to the planes every time the plane moves. There are also obstacles in the form of logs of different sizes that obstruct the plane's path. As difficulty increases, the number of obstacles and speed of the projectiles increases. The plane shoots bullets constantly to damage the boss.

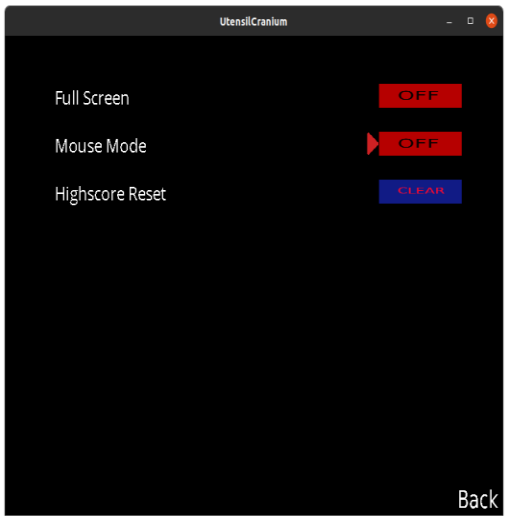


Continue: The “Continue” button allows players to start from where they left off last time. Takes players directly to the environment of the last time they played the game.

High Score: This button allows players to view the high scores of the game to raise up their inner competitiveness. One can also clear the high score table by going to options and clicking the clear button.

Options: This button gives a user opportunity to change the game mode and clear the high score table.

1. Full-Screen Mode: this allows the game to be played in full screen. Everything in the game is scaled to the window to allow gameplay to be the same.
2. Mouse-Mode: In this mode players can control their in-game characters by their mouse. This allows more flexible and fast gameplay.
3. High-Score Clear: As previously said, using this one can reset the high score table and make all the high scores effectively 0.
4. Sound Toggle: It toggles game audio on or off.



Help: This button shows users a page on how to navigate and play the game.

About: Gives general information about the project.

Exit: With pressing the Exit button, the user can quit the game.

In-Game Special Features

Power-Ups: There are two types of power-ups available to the players.

1. Life: There will be a heart on the screen. If the player is able to collide with it then he will get an extra life.
2. Invisibility: If this type of power-up is collected then the character will be invisible from all types of harmful objects like projectiles and obstacles for 10 seconds. It is shown by changing the player texture and blinking before the powerup runs out to communicate that it is wearing off and the player should remain vigilant once again.
3. Time Stop: This power up freezes everything except the plane, so the player can hit the boss easily or escape from projectiles and obstacles during this time effortlessly.

Enemy Attacks: Initial enemy boss can only use two types of attacks. But as the boss changes, the attack variety increases.

1. Bouncing: This attack is a sphere fireball that bounces when it hits the edge of the screen.
2. Homing: In this attack, a magical ball follows the character everywhere until it is dodged or life is lost.
3. Super Bomb: This attack is exclusive to the second boss. It is a high speed bomb with three times more velocity than the normal projectiles.

Invisible Frames On hit: After losing a life, the player will be invisible for some time. This is done to stop collision detection from working more than once per hit and giving the player a small chance to recover.

Skill based doge/dash: The doge or dash move is always available to the player so that they can get themselves out of dangerous situations with sheer reflexes. This provides a fun challenge to the game and makes it so that even in the worst rng luck the player has an alternative. It is signalled by a puff of smoke appearing from the player and the place where the player was before dashing. The dash smoke visibility corresponds to the IFrames awarded to the player on a successful dash. While the player texture smoke corresponds to the time before the dash can be used again. These times have a 200 millisecond delay between them. This is done to prevent spamming the dash button and make it so that only a successful doge is rewarded. We believe this elevates the gameplay to focus on skill even more.

Diagonal movement: In addition to the 4 directional movement our game supports diagonal movement keeping with the philosophy of reducing friction between the controls and the player.

Pause Menu: Once in the game, users can pause the game and see the pause menu. From there you can either resume the game or go to the main menu. Also users can exit the game from the pause menu also by pressing the Exit button.

Project Modules

The project has been divided into several header files each serving some specified function. These modules are discussed below.

Preprocessor.hpp

This contains the libraries used to create the project and some preprocessor constants which can be manipulated to change various aspects of the game for easier access. Unfortunately most UI and game elements are tied to the screen height and width since they are variable due to the introduction of window events. As such we could not use many preprocessor constants.

Globalvars.hpp

This contains the variables frequently accessed in all parts of the program. Notably the renderer, the window (a struct in this case), the screen variables, scores, health, screen scaling variables, various boolean variables to direct the program on a larger level. It also contains the function for the core gameplay loop and an enumerated screen which is used to keep track of the various screens in the game in a readable way. If one wanted to add another screen this is where they should begin.

textureMusic.hpp

This file contains all of the textures and music to be used in the game. It is isolated for easier loading and access as textures are accessed very regularly.

loadMedia.hpp

This handles all the loading of textures in the game. I/O operations are some of the slowest. That is why we load all textures and music at the start while it hits the memory hard and the initial loading is a bit long, the size of the entire program is rather small and ultimately this results in improved responsiveness of the program. To make this easier we have implemented a loadTex function which takes the string of the file path as argument and uses SDL_Image's IMG_Load function to load the texture in a surface and create a texture from that surface. Music loading is done using library functions. Finally this also contains the close function which closes the initialized systems and frees memory using library functions.

Utills.hpp

This contains all the necessary functions which are either essential to the basic framework or are widely used. This contains init which initializes systems and libraries used in the project, SDL itself, linear texture filtering, SDL_Image, true type fonts and SDL Mixer also initializes the window, and the renderer via the window struct. It is the first function called in the main program loop. New libraries are to be initialized here.

Next are some functions which scale variables according to screen size on window events. Updatescreen fetches screen resolution and the fraction of screen resolution by which it has been scaled and the functions scaleIntX/Y and scaleRect can be used to scale various elements. Highscoreclear function clears the high score, logError logs SDL errors, printText returns a texture with text, play fetches the current playing environment from an external file save game saves the game.OptimizeFPS saves the frametimes from the previous frame so as to delay exactly the correct amount to lock FPS. This is a fallback feature for devices lacking vsync.

Menu.hpp

Menus were decided to not be abstracted too much because there were many variations from menu to menu. Instead a cohesive style is used for the menus. The menus contain rects corresponding to the buttons on them, variables to draw these rects and a few functions to run said menu. The constructor first initializes the 1st menu element with the yVal which is also the menumin value, next the dimensions and the x offset is also initialized and for all subsequent elements the yval is increased by the padding between the buttons to make them align perfectly. The rects of all buttons are thus initialized. The menumax is found by multiplying menu elements with padding. The cursor dimensions are also initialized in terms of the button dimensions.

The updateUI just reinitializes the menu in terms of the new screen resolution. The cursorJump moves the cursor to the pointed rect. The cursorUpdate moves the cursor up or down according to the step.

Handle event, polls events and does a number of tasks. Firstly it closes the system on quit. It updates the ui on any window event. Handle event then polls keyboard events to move the cursor on navkey press. On enter key press it checks which rect the cursor points and can execute tasks accordingly. It also has a mouse handler which updates the cursor position only on change in mouse position and on left click checks if the mouse is inside a button and executes functions accordingly

Finally the run function of each menu, calls the event handler and then presents the textures one by one. Notable exception being highscore where the highscores are retrieved from a file rendered.

gameUtills.cpp

This contains utility functions used in the core game loop. Notably the upTimer which is an object capable of counting up and that can be started and stopped with functions also it can report its state and the time since it's been initialized. Then there is a collision checker which checks for collisions between 2 rects. Then there are 2 functions to calculate and print highscore. Difficulty is modulated by a function so is boss change. ScaleGame scales the game elements in a window event and initGame initializes a game. The variable diffthreshold and diffstep is used to modulate difficulty.

gameElements.cpp

The game elements have different attributes; a general introduction to elements is given. In general init initializes the elements, scale scales and move moves the elements according to its velocity, render functions render the element. The player element has a render rect and a hitbox rect and rects for the player bullets. The player has a handleEvent function which handles the controls for the player. It takes a snapshot of the keyboard states and checks which keys are pressed and moves the player accordingly. This allows for diagonal movement. In mouse mode movement is tied to the mouse cursor. The player bullet sub routine fires the player bullet from the player position straight forward. Col checks for collisions with the player.

Incase of other elements the hitboxes and render rects are the same except for the enemy attacks. Walls use the wall object to move an array of walls in tandem. The change in powerup is that it checks collision with the player render quad thus being more lenient. The types of power ups are enumerated for better readability and the choose function uses a random number generator to choose the next powerup. Spawning of these powerups are tied to timers. The powerups activate on hit and the run function handles the behaviour of powerups.

The boss attacks are another struct which also uses choose and run to select one attack and a subroutine to execute the features.

Team Members Responsibilities

1. **Muhammed Tasnim Bin Anwar, Roll: SH-05 (Team Leader):**
    a. basic UI.
    b. UI assets.
    c. All Works related to backgrounds.
    d. Showing scores, lives, etc. to screen.
    e. Works related to true type fonts.
    f. High score calculation, storing and printing.
    g. Game saving and resuming.
    h. boss health and phase changing.
    i. Adding sound effects and background music.
    j. Player movement.
    k. Difficulty level.
    l. Game over Screen.
    m. Debugging.

2. **M Muztoba Hasan Sinha, Roll: AE - 15**
    a. Struct base.
    b. Initialization of SDL and assets.
    c. UI navigation with mouse & keyboard.
    d. Window Events
    e. UI scaling
    f. Mouse Mode.
    g. iFrames.
    h. Collision Detection.
    i. Player and boss movement.

j. Power-up variety.
k. Obstacle spawning and movement.
l. Enemy attack.
m. Debugging.

Platform

The project was developed in Ubuntu 20.04.

Libraries

The project was built using the c/c++ language and SDL2.0 library. Mainly SDL_Image, SDL_ttf, SDL_mixer library was used.

Tools

Tools such as VS Code, GNU Image Editor, Terminal were used to code, edit or run the project.

Limitations:

1. The game has only one character. We believe both co-op and vs multiplayer can be implemented with minimal to no modification.
2. Only one level was designed.
3. Attack variety is still on the lower side. Again this should be relatively easy to increase.
4. Only one type of obstacle was included.
5. Enemy attacks are still not sufficiently varied.
6. Only two bosses.
7. The text print features were written in such a way that it does not allow for the text in title screen and game over screen to remain centred when screen resolution is changed.
8. There is an exploit due to the design that a player can return to the main menu and hit continue to reinitialize the elements.

9. There is a bug where the game in fullscreen mode cannot be recorded by any recording software.
10. Since load ops are done at initialization it is relatively slow to start. A load screen wouldn't be very interactive and due to time constraints so we decided against it. Besides, the wait isn't too long.
11. The game is actually a closed loop. Ideally we wanted it to be a boss rush. Now while there are several bosses there is no end to the game.
12. Background jitters in some resolutions. Due to our not stitching the texture in image editors correctly
13. Walls might pop in front of the player when they 1st spawn. While this may look jarring this might happen once per game loop, and gameplay ramifications are negligible.
14. Scoring is time based. Ideally a robust system for scoring should be in place.
15. The documentation and report covers most of the systems on the surface. But does not explain every function individually in depth.

    Aside from the full screen bug, we think all other bugs are addressable very easily with some time. Aside from these we are confident that gameplay and UI elements work nicely.

Future Plan:

In the future, we want to address issues raised in the limitation section like turning the game as a level completion to progress type with distinct levels. Adding more player and enemy attacks to make the game more enjoyable. Keeping with the theme of accessibility we'd also like to add controller support soon.

Conclusion:

We faced various challenges during the time we worked on this project. Due to some device issues we couldn't simultaneously code until the last 2 weeks of the project. In the meantime we studied different elements of the workflow individually. But linking the systems together gave rise to several unforeseen bugs.

Most notably the animations kept breaking. And perhaps the biggest issue faced during the project was the scaling issues. We were adamant to seamlessly scale the UI on window deformations and changes. This gave rise to a slew of bugs which took a lot of time and effort to squash. Although one still stumps us. This is the screen recording bug.

Other than that we learned that structural programming actually helped us find faults faster and better. Going into it we didn't have any experience of working with such a huge amount of code. Structured approach and error logging helped us in this regard. Through this project, we learned a great deal about teamwork and game development. We found out that most of the problems that we face can be solved using the knowledge we already possess, we just have to be tactful about how we approach the problem. Also almost all of these problems are tiny basic mistakes, thus it proves the approach of teaching us the basics of c/c++ first by our instructors. As this is our first game, there has been a very steep learning curve. Thus, all the features that we thought of at first could not be implemented correctly. And due to a personal problem of a team member, the coding process wasn't very smooth. But we tried our best to present something creative and to be proud of.

Github Repo: https://github.com/sinhSlumbering/project_sdl_1211

Youtube Video Link: https://www.youtube.com/watch?v=31lILidVpOk

References:

1. https://lazyfoo.net/tutorials/SDL/index.php
2. https://www.libsdl.org/
3. https://www.willusher.io/pages/sdl2/
4. Almost all the pictures used is from the game “Cuhead” so if you like you can buy the game from here: https://store.playstation.com/en-us/product/UP8062-CUSA20499_00-CUPHEAD000000000