Final Project Report

Jamie Maher, Joey Curci, James Farrington, Chris Rho

User Manual

This is the user manual for the Bui Buster's Edition of Mario Time Trials.

Libraries Needed

To utilize this program you will need the following libraries on your machine: `SDL2`, `SDL2_image`, `SDL2_ttf`, `SDL2_mixer`, and `C++11`.

Build Instructions

To compile the program, simply use the makefile we provided. The makefile will do all the compiling you will need by running the command line statement `g++ main.cpp -o main -lSDL2 -lSDL2_image -lSDL2_ttf -lSDL2_mixer -std=c++11` to compile the `.cpp` file into an executable.

Controls and Interaction

Once you run the program, the main menu will appear. Click on the start button to begin the game. The left/right arrow keys move you left and right, respectively. The up arrow key will make Mario jump. To kill the goombas and koopas you must land on the top of their heads. You cannot kill the thwomp, but if you land on top of it you can ride it. Killing the goomba will give you 150 points and killing a koopa will give you 200 points. In addition, collecting a coin will give you 100 points. If you suddenly discover a mystery box, hitting it from the bottom will make a green mushroom appear. The green mushroom gives you an extra life. You can lose a life by falling off the level or hitting an enemy anywhere but from the top. Your time, score, and lives counter are all displayed at the top of the screen. The goal of the game is to get to the end of the level as quick as possible with as many points and lives as you can. We developed a formula for calculating your total score which takes your final score and adds points for your lives and subtracts points depending on how long you took. Once you finish the level, the program will display your final score as well as a leaderboard for the high-scores

Known Bugs

While we were able to fix all major bugs, there were still a few minor issues that persisted. One minor bug is that sometimes the Mario character will get temporarily stuck on the side of a wall block when jumping. This happens very rarely and we found it not having a significant effect on the game. Another bug occurred when trying to run the program on a personal Mac computer versus on the student machines in Cushing. The thwomp enemy, when the game is run on a personal Mac, causes the user to oscillate when atop the thwomp. This does not occur on the student machines as we think the Mac is simply faster at rendering the SDL components, which causes the player image to load in the incorrect place. The last bug concerns the timer. In our game, the timer starts as soon as the SDL window opens (not when the player hits start). Some would say it is better to start the timer after the user clicks, but this way it test the users agile reflexes; thus bringing a physical edge to our computer game.

Other Information

Other than having all the SDL libraries properly installed, and the image and sound files in the proper directories, there are no other libraries or equipment needed. The program can easily be compiled on the student machines.

User Perspective

The users perspective for this program is to use keyboard controls to move the Mario character and kill enemies, collect coins, and reach the end as fast as possible, while trying to maximize their score. They will also have their score, if it was high enough, it is saved to a file and displayed among the high scores at the end screen. If the user fails to reach the final flag pole and has no lives left, the Game Over screen will appear with the appropriate sound file.

Internal Code

Internally, the program works by making use of classes, objects, inheritance, and the SDL libraries. To generate the level map, we used tiling. It parsed through a sprite sheet to load the tiles. Some tiles have certain properties, like a brick cannot be jumped through, and a question mark brick will make a mushroom appear. To generate the enemies and the player, it uses classes to instantiate these objects. As gameplay progresses, the objects are moved according the various collision functions that determine where the object should go next. Each collision function was slightly different, as they were needed for different purposes. Essentially, a collision function worked by taking two inputs of two hitboxes. If the boxes overlapped in the desired way, then the collision function returned true. Then all the objects are rendered and the program continues in this loop. To render all the various sprites and objects, the `LTexture` class is used. This class makes use of the `loadMedia` function, which loads all the images and sound files. The `LTexture` class assists with rendering and moving the images and objects.

Code Verification

We verified that the program was working correctly by playing the game many, many, and many times, to the point where some of our wrists are throbbing in pain. Each time we played we would observe how everything was interacting with each other and ensured that it was all correct. This was very important when handling enemies because we had to test to make sure that the collision detection was working. This required much play testing as we attempted to hit enemies or coins from every angle to ensure everything was working as intended. A few methods to accelerate our checking was changing Marios initial position so he loaded somewhere close to the object that we were checking with. Several issues were found by continued playthroughs of the game, including an issue with the jumping mechanic. By testing the game so much, we were able to find all the bugs in our code and fix them.