Game Report for Spectre

Cooper Jackson

April 15th, 2013

Course: CMPC2GO4 – Graphics 1

Student ID Number: 100048124

Cooper Jackson

Game Report for Spectre

The game is relatively simple and can be accurately described as a platform shooter. Design and aesthetics were inspired by such games as contra and Tron. The goal of the game is to destroy all of the enemies without dying. I have designed the game to be difficult, and as such the player only has one life. If the player dies, the game is reset from the start of the level. The gameplay relies on shooting mechanics, and both the player and enemies have guns. Upon seeing a player, an enemy will stop and fire. The player has to successful y shoot each enemy without getting hit to win the game. The player can safely land on top of the enemies but cannot touch their sides. Game progress is measured by the progress through the sections of the level. In the main level, there are four different sections of enemies which must be cleared.

Class Structure

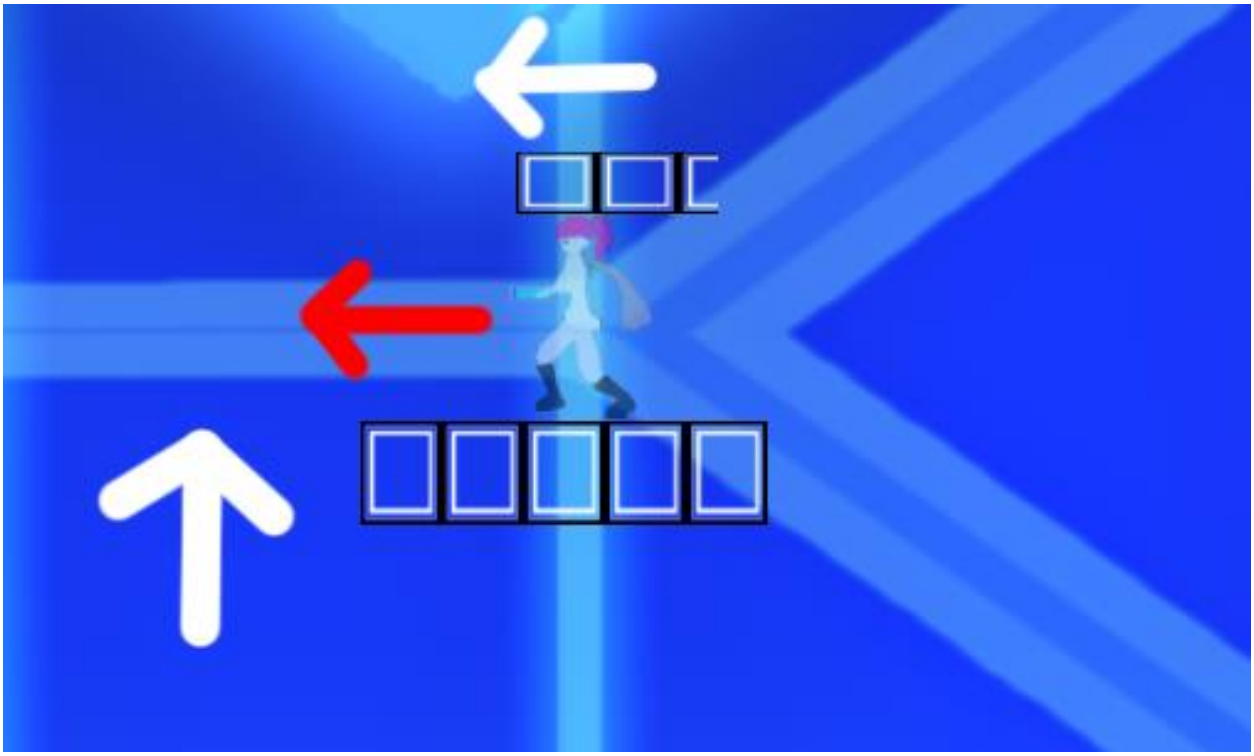
The game is laid out into two major classes, the player class and the main GUI class. Inside the player class, there are four de facto classes, Player, Block, AI, and Bullet. Originally these classes all were children of Player but due to my lack of C++ expertise I eventually compiled them into one class to avoid plaguing pointer errors. The main base work of my main class was taken from the third openGL example provided in the example code.

Assets

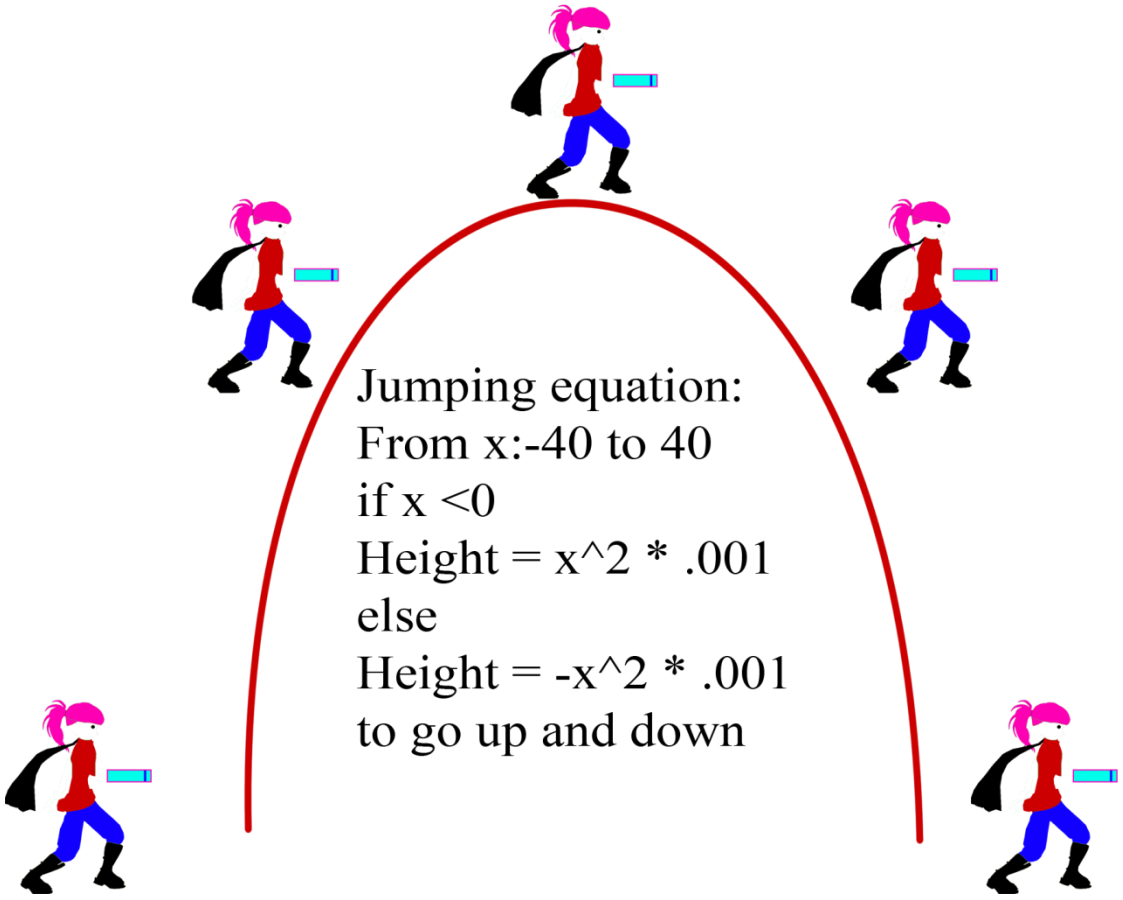
The majority of textures were hand drawn in Photoshop. That being said, part of the title screen and background were taken from a “free for personal” Photoshop website. The link is provided in sources. The background music in my game was provided by freesounds.com under a personal use license. To get the textures loaded into the game runtime, the PNG loader class that was provided on the portal website was used. The music is loaded in asynchronously and is looped.

Collision

The collision algorithm in my game is a simple AABB collision detection algorithm. To prevent clipping, especially with the player object, there are a couple of tweaks. If a moving platforms runs into a player object, the moving platforms stops and waits for the player to move on its own accord. If the platform runs into a wall or another moving object, the direction of travel will be reversed. The direction holding and switching works for both the x and y axis. When a player objects lands on a moving object, the player, barring any collision with other objects, assumes the velocity of the object the player object is on. This allows the player to be carried by moving objects. Since the player is being moved by an object in the y direction, there is a possibility of the player being squished into another object. If the player encounters two moving platforms as pictured below, the upper platform stops and the player if forced horizontally out of the collision. Bullets and enemies are subsets of moving platforms and they act very similar. They interact with each other and enemies can cause platforms to change direction or stop their movement and vice versa.



Jumping and falling



The jumping equation is different than my falling equation. Once a jump is triggered, the player object disregards the method that adds on a gravitation pull. After the arc is complete the player returns to the standard gravity rate. The standard rate is accomplished with a variable that increases each time the player does not encounter an object. The equation is quite simply: fall=fall+-.025. The starting value of fall depends on player actions. If the fall starts from rest, the fall variable is set to 1.5. If the player is coming out of a jump, the variable is set to 1.6. Changing the variable matches the speed out of the jump. The user can decide to cancel the jump by pressing the down arrow. This defaults to the fastest falling speed since it is mainly used for dodging bullets.

Transform screen x/y and letterboxing and 16:9

To keep the viewpoint and viewing area consistent for movement in the game and resizing of the window, I implemented a resizer. If the window is resized to a different resolution, the resizer finds the closest 16:9 resolution and the game window is letterboxed. The viewing area of the game never changes and the objects rendered scale properly with the window size. This effect is obtained by scaling the shapes via the aspect ratio on the horizontal axis.

The player is always in the center of the screen and the environment is translated around it, giving the perception of movement.

FPS openGL

The game depends on a steady frame rate of around 59-60 frames per second. To obtain this, vsync has to be on to deliver consistent performance and timing. About three quarters the way through coding I discovered that vsync was not forced default on some computers. Since this broke the game, I spent a significant amount of effort into trying to fix it. The best solution that I came across for forcing vsync is a method call named “wglSwapIntervalEXT.” However, as far as I can tell this does not work all the time. I tested my build on many computers with varying hardware and it appears to default with

vsync on on Nvidia and AMD cards. Intel integrated cards were a hit or miss, especially with the HD 3000/4000 series. The game will still work but vsync must be forced on via graphics drivers.

Sources:

http://www.brusheezy.com/psds/34536-neon-digital-circuit-technology-psd

spectre

Space

use arrow keys to move
press down to fall
press space to attack

Kill or be Killed