UNIVERSITY OF MISSOURI
CMP_SC 4610 COMPUTER GRAPHICS I

FINAL PROJECT:

Endless runner game using mobile device's gyroscope

GROUP 14
May 8th, 2017

TEAM MEMBERS & ROLE ASSIGNMENTS

Jordan Jones - Enemy Spawn, Character positioning, Game Physics, Collision Detection, Documentation

Natalie Lung - Asset Loading, Art style, Scoring System, Sprite Animation, Collision Detection, Documentation, Presentation Video

Chase Hamilton - Character Positioning, Game Physics, World Bounds, Gyroscope Tracking, Documentation

YOUTUBE LINK
https://www.youtube.com/watch?v=gEiLgBrf2n8

PROJECT DESCRIPTION: CODE WALK-THROUGH

Our project is a Javascript and HTML5 game built using the Phaser framework. It works on iOS and Android devices (with device orientation locked), as well as on the Edge browser on Microsoft Surface devices. Everything is hosted and can be played here:

https://cg-mobilegame.github.io

This project can be walked through in its entirety through game.js. This Javascript file holds the entire functionality. The code starts by creating a new Phaser game. Then the variables for the game are created. The names are descriptive of what they do.

There are four main methods in this game: Preload, Create, Update, Render.

First, the goFullScreen() helper function is defined. It enables the device to go full screen and is done by scaling the page

Loading screen before the game

The function onPreload() starts the Phaser Physics engine. Then the function goFullScreen() is called. Then it displays the "Loading…" text on the screen while it loads certain assets into different keys to be used by the main functions.

Function onCreate() is where most of the game actually takes place. It starts with the logic for the constant scrolling background. We made our background image a repeating texture by making it a tileSprite in Phaser. The next two lines are for a background music track. This track doesn't loop infinitely the way we want, however it is one of the ways we could improve the app.

Office.png background

Player spritesheet with six frames. One of them is blank and unused.

The next four lines of code add the player to the stage and add his animation. The animation is done by splitting the loaded png file equally into 2 32x64px frames and looping through them. We then alter the physics by changing gravity and anchor the player to the bottom of the stage. We make sure collision can be detected and make sure there is no bounce for the player implemented.

Then we set the gyroscope frequency and the gyroscope tracking function. We confine the player to the world bounds that we have set on the x axis. This enables movement along the bottom by tilting your mobile device. The last couple of lines of this function are the implementation of random enemy spawning using random math function and two create helper functions.

The helper functions just add the sprites to the game on a random x coordinate and a set y coordinate of 0. Then the helper function sets physics and disables world collision and in the case of the boss, scales the sprite by 200%. Now we have the screen being populated with a single enemy but we want to call this function on repeat. Since this is supposed to be an endless runner we set the time to ten minutes of gameplay. We thought ten minutes of gameplay would be sufficient to keep the audience happy and to satisfy certain conditions we had preset.

The last three functions are update, collision Handling, and the render function.

The collision Handling was a basic implementation to handle the end of the game. Whenever the player hits an obstacle, a sound effect 'hit' will be played. The player's animation will then switch to its last three frames, causing it to 'die'.

Game over title

The game over title is also displayed in the center of screen. We couldn't find a way to stop the internal game clock without states at this stage. The gravity is unset to show the end of a game as a partial solution. This will be covered a bit later in the potential solutions part.

Update() is the function where collisions are handled and the offset position of the tiled background is set so that it scrolls vertically.

And the final function is render() where the score is shown as time run times 1000 on a debug text value. Currently, 1 second = 1000 points.

REMAINING ISSUES

We were unable to meet all of the things we wanted to do with our HTML game. The main functionality of controlling a character with a mobile device and toning the sensitivity to a reasonable value was implemented first. We didn't complete all game states which would have been a nice addition to tie the whole project together. Game states in this sense means a start screen. Another goal would have been to properly implement sound effects so that it plays on iOS and Android devices. Since the game is meant for mobile devices, tying in sound effects and game states would have completed our project by showing the user a score that can be a good indicator of their performance.