**FlappyMingo**

**Final Project Report**

**Kyle Williams and Thomas Franceschi**

**December 15, 2017**

1. **Project Description**

This project is an interpretation of the popular mobile game Flappy Bird that came out a few years ago. Using a three.js flamingo model, we developed a game similar to Flappy Bird in which the user has to hit the spacebar to make the flamingo model flap up in order to fly through a series of gaps in blocks. The blocks were also rendered using three.js. When the user runs into a block instead of flapping through the gap, the program throws an alert box, indicating a game over and displays the score of the user. The score is incremented each time the user successfully flaps the flamingo model through a gap in the generated blocks.

Disclosure: This project was built off of the webgl\_lights\_hemisphere.html from the three.js example code found athttps://threejs.org/examples/#webgl\_lights\_hemisphere.  In the following section, the added functions and changes to the example code are outlined.

1. **Main Function Implementation**

This project was implemented using JavaScript embedded in the HTML file. In that program, the blocks and the flamingo models are generated in the init() function. Block generation happens with the use of three.js box geometry object, THREE.BoxGeometry(), and the three.js mesh material, THREE.MeshStandardMaterial(), to create a custom box mesh with THREE.Mesh(). The flamingo model exists in a flamingo.js file under the /models/animated directory in the main three.js folder. This model is loaded with the THREE.JSONLoader(), and using the THREE.MeshPhongMaterial() object, the lighting is added to the flamingo model.

Following the init() function, the onKeyDown() function listens for the event that the user presses the ‘h’ key, ‘d’ key, or spacebar. The ‘h’ and ‘d’ keys toggle the hemisphere lighting and the directional lighting, and they were given in the three.js hemisphere model that we used. The case where the spacebar is pressed sets a jump variable to 1, indicating that a jump is true. In the jumpFunc(), if the jumping variable has been set to 1, the y position of the flamingo mesh is increased for several consecutive frames, simulating a flap upwards. After the jump has completed, the y position of the flamingo mesh is decremented until the bird returns to the base height, simulating falling.

In the onKeyDown() function, when the left or right arrow keys are pressed, the directional light is increased or decreased in the positive or negative in the x direction. When the up or down arrow keys are pressed, the y component of the directional light is increased or decreased in the positive or negative direction accordingly.

The pipes are translated from the far right side of the screen to the left side in the movePipes() function. In this function, each of the four box models’ x positions are decremented to give the appearance of the flamingo’s movement in the positive x direction. The y positions of the blocks are randomized using the current time to provide various heights for the user to try to flap through.

The hit detection for the flamingo into the blocks is calculated in the detectHit() function. This function checks to see if the flamingo’s x position falls within the blocks’ x positions and if flamingo’s y position falls within the blocks’ y positions, it returns 1 if a hit has been detected, otherwise it returns 0.

In the render() function, if the detectHit() function has returned 1, an alert is thrown, displaying the player’s score and indicating a game over. Otherwise, if the blocks’ x positions have passed through the origin, where the flamingo is located, and there has not been a hit, the score variable is incremented and displayed on the browser screen by creating a ‘div’ element and setting its innerHTML to the score.

The setScore() function increments the score variable if the user has not flapped into a pipe and then displays that score in the upper lefthand corner of the browser window during gameplay.

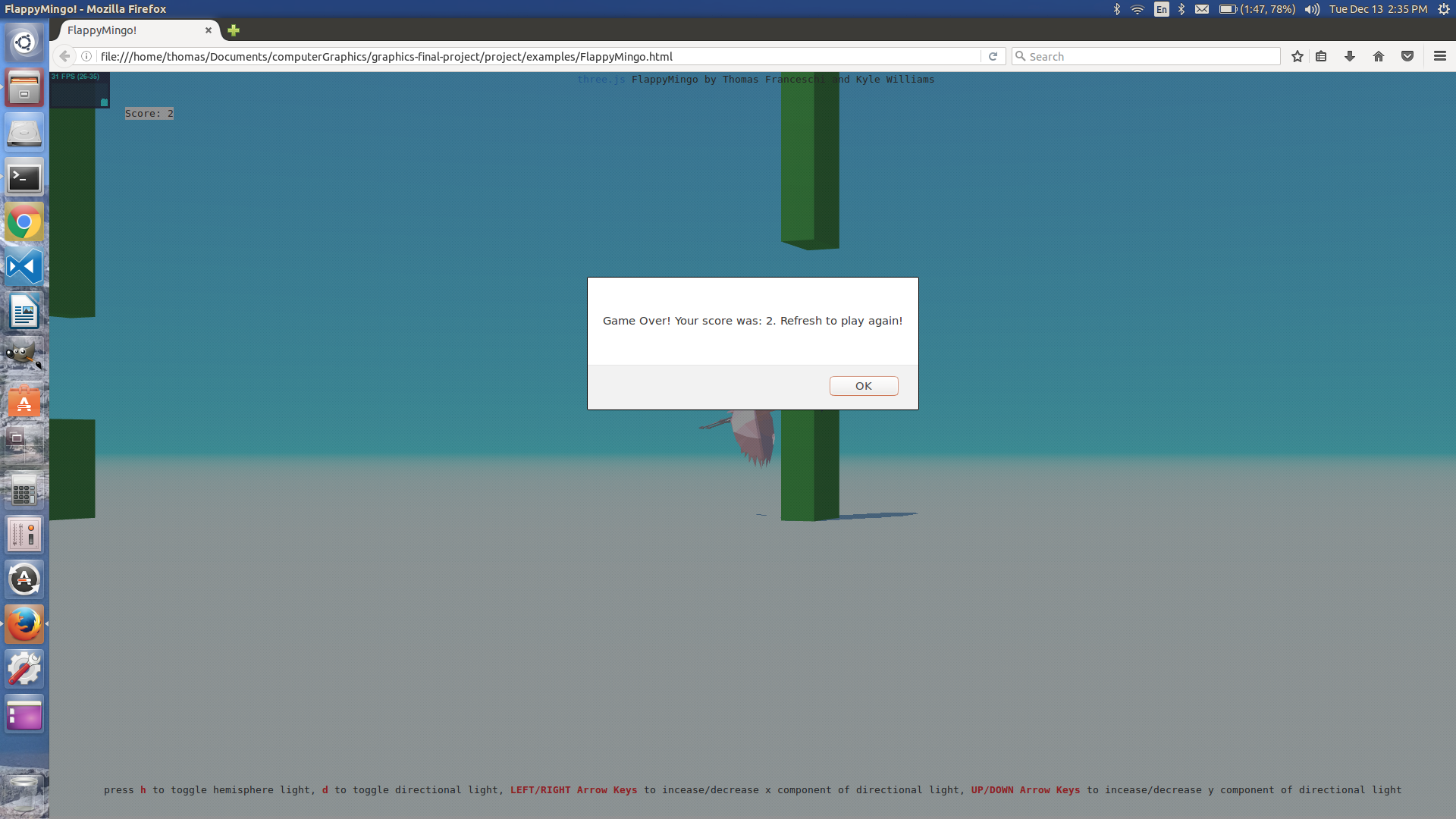
1. **Running the Program/Gameplay**

In order to run the game, the user simply needs to navigate to the project/final\_project/ folder and open the FlappyMingo.html file in a web browser. Although the game may work on Chrome, Safari, and Firefox, it seems like Firefox allows for the best performance.

**Figure 1:** Flamingo model flapping through opening in the blocks. The

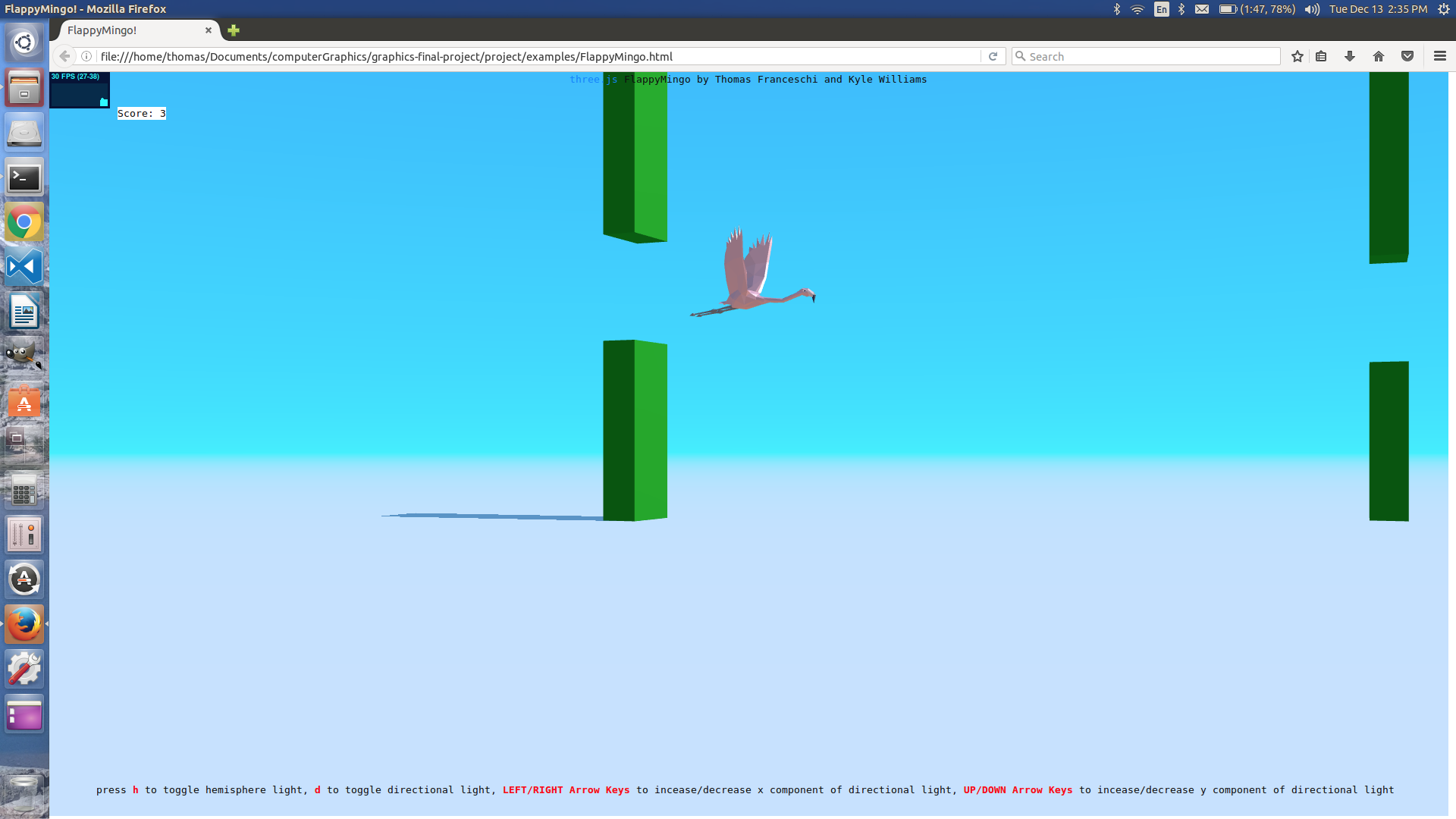
score is displayed in the little white box to the upper left of the browser window.

The game begins once the program is run, and the user must hit the spacebar to get the flamingo to flap through the spaces in the blocks. If the user runs into a block, the game is over, the score is displayed to the user in an alert box, and the page must be refreshed in order to restart the game.



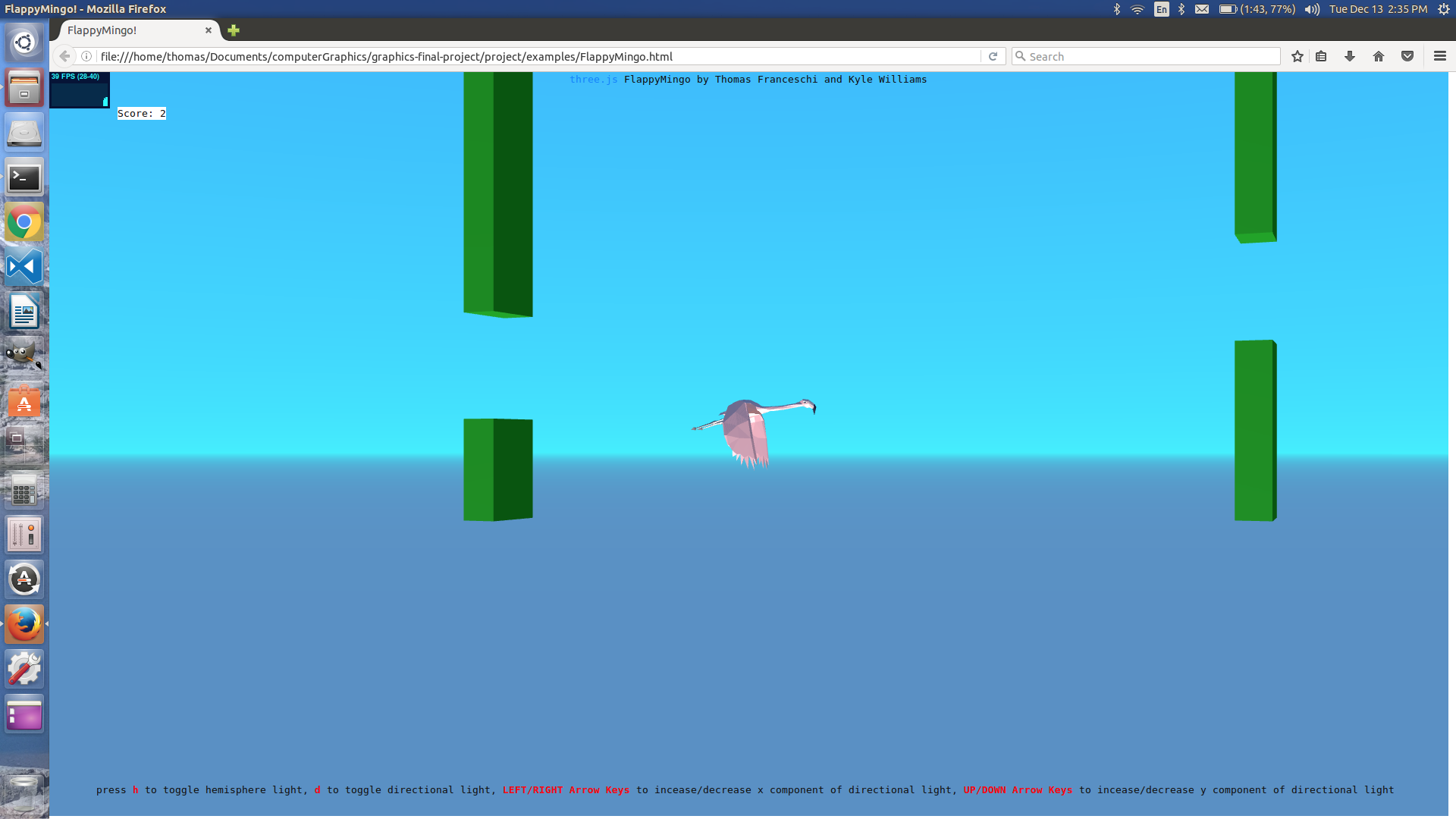
**Figure 2:** The flamingo has collided with a block, ending the game.

The alert box displays that the game is over, the ending score, and

that the user must refresh the page to play again.

**Figure 3:** The x component of the directional light changed in the x direction,

making longer shadows.



**Figure 4:** The y component of the directional light changed in the y direction,

making it look like the sun was going down.

1. **Technical Challenges**

Our biggest technical challenge was learning the three.js library. We were able to overcome this learning curve by reading documentation, looking over the video lecture and three.js lecture slides, and playing around with the example code in the three.js master folder. Otherwise, our desired implementation was relatively straightforward. In our proposal, we were a little concerned with the actual object modeling and game physics, however, three.js made modelling the flamingo and blocks very efficient, and our game physics is meant to be similar to the original Flappy Bird game. As a result, the game physics is not authentic in the natural sense.

1. **Teammate Effort Assessment**

I wrote the hit detection code, the jumping and falling code, as well as rendered the blocks and wrote the code to make them move.

Kyle added the functionality to change the directional light x and y coordinates, to change the look of the setting. He also added the scoring capability, and throwing an alert box upon hitting the blocks and getting a game over.

1. **Lessons Learned**

Throughout this project, I learned about rendering models using three.js, adding models and lighting to a scene, and setting up and rendering a scene. I also learned about some game play mechanics and how playability factors into building the parameters and rules of the game.