Hunt, Hill, Langham, Lonsway
CSCE 315-503

Space Escape Textual Overview

Instructions:

All of the necessary code will be in the zip folder Prototype. This needs to be unzipped. Now open a putty session with SSH port forwarding. When putty is started go to **Connection** on the side bar then go to **SSH** then to **Tunnels.** In the **Source port** enter “10268” and in the **Destination** box type “localhost:10268”. Then click the add button. Once this has been done, log on to the compute.cs.tamu.edu server and navigate to the folder with the unzipped files. Next, run the server by typing “node server.js” in putty. The server is now running on port 10268. To launch a client, type “compute.cs.tamu.edu:10268” in a browser (preferably Google Chrome).

The first thing the client does is require a log-in. For now, this is just simply requires your name. Then, you are taken to the main menu where you can choose a game mode or look at the instructions and high scores (high scores are displayed by clicking the appropriate game mode in the bottom left). The multiplayer modes require two players. Once two clients have selected the same multiplayer mode, then that game begins. Lastly, the game is played by using the up arrow key to thrust and the left and right arrow keys to turn the ship. Time trial and race are finished by reaching the finish line; whereas challenge mode ends when a player dies.

Introduction:

This design describes an original game which attempts to provide all of the aspects of meaningful play. It is designed as an online, multiplayer racing game with a unique control system application. The style targets a retro-aesthetic appeal, which is complimented by a simplistic control scheme. The game will support two modes: race and challenge. In race mode, the screen will scroll based on the movement of the player towards the edge of the screen. However, in challenge mode, the screen automatically scrolls at a predetermined speed regardless of the position of the player on the track.

Inventory of Source Code:

Our source code will be contained into one folder where the server.js, index.html, and the other JavaScript files that make up our game.

Animator.js

This file contains the code for animating a player’s death. This includes the breaking up of the ship upon impact with an obstacle and respawning the ship at the correct checkpoint on the map.

Block.js

The block.js file is included in Level.js and in CollisionDetector.js to represent the game layout and for game to detect whether the player has hit an obstacle. Block.js uses the implicit function and linear function to create the lines of a block and to determine intersection of the player and these lines. The block object has an array of four points and an array of four lines connecting the four points using the linearFunction(p1, p2) function.

Bullet.js

Bullet.js contains the function for creating and drawing bullet objects on the course. These are circular objects which appear on the right side of the screen and shoot towards the left side of the screen. If a ship collides with one of these bullets, then it is destroyed and must respawn. Currently, bullets are disabled due to the problems we are facing with the game being too hard. Right now, it is hard enough without bullets flying in your face every level.

CanvasText.js

The CanvasText.js is used by the MenuManager to be able to have clickable text that results in an action for the game to perform. The CanvasText.js file holds one function that takes as arguments: a string to be the clickable text, x position, y position, a clickable Boolean variable, a callback function, and which game mode the text is for.

CollisionDetector.js

The collisionDetector.js is a file that contains the hasCollidedWithShip function. This function takes a ship, a block, and whether the game is in multiplayer mode. This function is used in the update function of GameManager.js. In this update function, the hasCollidedWithShip function is put in a for loop testing the ship against all of the blocks in the level. If the ship collides with any of the block, the ship respawns in a new starting location.

Game.js

Game.js is where the Game object is defined. A game object has a graphics member which is passed from the index.html. The game object also will create a GameManager and a MenuManager. The start function begins the game at the start menu. The goToGame function is a function that starts a new game. Game.js has functions such as run, which draws and updates what is returned from the gameManager's

draw and update functions. The game object also provides functions to end the game and to return to the main menu.

This Game.js file will also communicate to the server.js on where the clients are in the game. Based on this information and if there is another client wanting to race, the server.js will send this information through the socket.io as a JSON object. This is done so the two clients will know where each other are in the game and be able to begin the race at the same time.

GameManager.js

In GameManager.js is where the gameManager object is defined. A game object is passed to the gameManager which in turn creates a ship for the both players, a level layout for each player, a buffer for the levels to be ready to be displayed to the players, the current levels of the players, and their state in the game. (i.e., thrust, turning, dead, pause) The gameManager will then place the player objects on the track and generate the level layout and keep the player's position. The update function in gameManger.js will determine if the ships have collided with an obstacle; it will also set where the ship respawns and have the data for redisplaying the ship's movements. The draw function will call all the other draw functions necessary for the game play environment including: drawShip, drawBackground, drawBlocks, draw opponent's Ship, and opponent's blocks.

GameManager.js also handles broadcasting updates to the other client of a multiplayer game via the server. This is achieved by sending update objects through the websocket to the server, which then emits that object to the other client. The data encoded in these update objects consists of: ship x-position, ship y-position, ship rotation, screen offset and opponent's current level. The encoding is done by using the basic JSON features of JavaScript and socket.io.

ImplicitFunction.js

This code currently is extra and not being used. It is unlikely that this will ever be used, but it is still there just in case.

Index.html

Index.html is what the client sees. It is written in JavaScript, CSS, and HTML. It uses built-in graphics for the game. We initially were going to use Easeljs, but it ended up being easier just to not use that. The HTML code simply sets up the basic pre-game stuff, and then launches the game. More specifically, it creates Event Listeners to tell the Game object if the user is pressing any relevant keys or clicks the mouse, takes the user through a log-in screen, and then launches the game.

Level.js

Level.js is where the levels are defined. A single game is created by making the starting line, the finish line, and choosing from the array levels to display to the player between the starting line and finish line. The levels are defined before-hand by an array of blocks. An array of levels are designed which is then displayed in random order to the clients. In multiplayer mode, both clients will see the same series of levels. Currently the game only has five levels. These five levels are linked together to create one racing track for the players to use.

LinearFunction.js

LinearFunction.js defines a LinearFunction class and a function to compute the intersection of two linear functions. All that a LinearFunction object consists of is the definition of a line between two points. So, LinearFunction defines a line, and the intersect function computes the intersection of two lines defined by LinearFunctions.

MenuManager.js

The menuManager object is what controls which menu to display and how to transition into the difference menus and game modes. The menuManager keeps track of the current screen which starts at the MAIN_MENU. The menuManager holds an array of CanvasText objects to display at the very beginning of the game. The menuManager will use these Canvas objects on the game object that is passed to it. The menuManager object will also draw on the graphics object that is passed to it. The menuManager will draw the CanvasText that correspond to the current game mode. The menuManager also has functions to go from one game mode to the other based on the player's clicks. It also handles the drawing of the instructions and high score screens, along with requesting and receiving the actual high scores from the server.

Point.js

This is simply a file with only one function that is used to represent points in 2D. This file is used in every JavaScript file we created.

Server.js

Server.js is where the database of high scores and the records and passwords of all the people who have played the game. The server implementation will be achieved using JavaScript along with node.js and

socket.io. It will be hosted on the Computer Science Department's compute server. It will send JSON objects through the web sockets to the clients. This is how the clients are able to see the progress of their opponent as they play the game. Another one of the tasks that the server performs is keeping track of who is online, where each player is in the game, and who is playing who. This aspect gives the ability to do multiplayer gaming. The last main task that the server performs is relaying updates from one client to the other, so the client can play each other in real-time. The server will send the JavaScript object "updateObject" to the clients that are playing each other, and the server will also receive this object from the clients. The server.js contains a request handler that will process the responses from the index.html. The server.js has the login functionality to decide if this is a returning player or a new player. From here the server will process the actions of the client, whether the client choose to play multiplayer or single player, time trial or challenge mode, or if the client wants to view the high scores or instructions.

Also, the server keeps track of the top 10 times and challenge distances for each player, along with their multiplayer rating. The overall top 10 of all three of these things are also stored separately. All of this data is stored to files so that the server can be shut down without losing any data.

Ship.js

In ship.js is where the player's piece is defined. The ship object is a movable triangle; it has a rotation speed, x position, y position, a height, and the two variables (vx, vy) that are used to for the ship's velocity and direction. In ship.js, there are four functions:

Thrust: Calculates the next x and y coordinates based on the current speed.

Update: Simply adds the values returned by the thrust function to the ship's x and y coordinates.

Left and right turn: Changes the rotation of the ship based on the ship's rotation speed which is a set value.

Timer.js

This is the code for creating and updating the timer that is found at the top of the screen during a game. It maintains minutes, seconds, and tenths of a second.