Developer Guide

Adam Mills

March 25th 2019

Introduction

Lone survivor is an HTML game written primarily in Javascript and is intended to be run within the browser and hosted on a server and connecting to a MySQL database. For installation of the various software packages in regards to setting up the development environment, please read the administrators Manual.

The code provided operates, for the most part, on the phaser 2.6.2 library included from a CDN within the head tag of the index.html file. Within the head tag, there is also the Bootstrap CDN, Jquery CDN and Ajax CDN, all which provide the functionality for the highscore table to the right of the main canvas of the game.

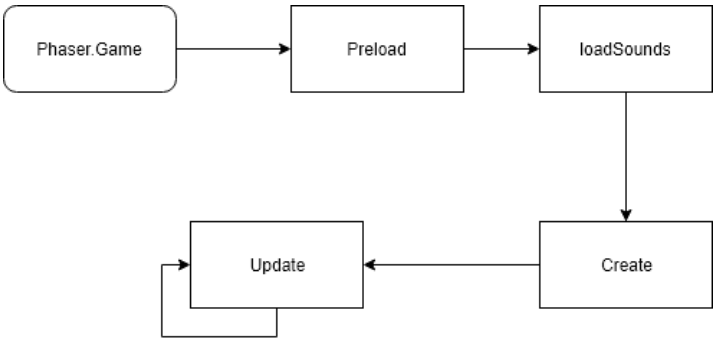
Code Description

The code provided in the source follows a monolithic structure, consisting of the main game creation and initialization within the index.html file. Additionally management of game elements is delegated to the manager files, where Manager classes and the game objects they operate on are defined.

Index.html

This file details the majority of the operations which compose the game at a high level, acting as a go between for the managers, especially in regards to hit detection and also details all of the game initialization.

Game initialization follows the route of the phaser game being initialized onto the game var which calls the preload function to import and rasterize the spites, then to the create operation which begins with the loadSounds function that encode the music and sound for the game which must happen first before the create method is called, because this is when the theme must begin playing, which initializes game object. The create function adds all text elements and managers to the game.



The update method exists as the core gameplay loop, it has 4 different running states; it performs no operations if the game is paused, it only moves the player object and background when in menu, it has standard checks for colisions and runs game object updates when the game is running, and finally performs collision detection for the boss only when a boss is spawned. It also will update the game timer and combo display in the standard gameplay loop.

Additionally, this file initiates progressing to the next level with the levelWin, levelProgress and gameWin functions, and toggles the game state between the four states of menu and playing with startGame and restart, ended and paused with togglePause, handles the operations between the various game objects being hit with all the collision functions and finally the getting and adding to the highscore board.

LevelManager.Js

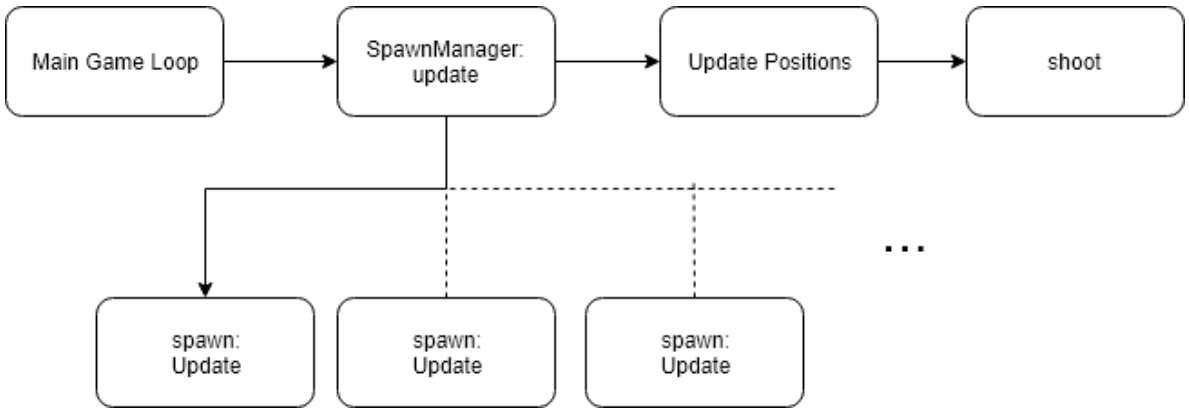
The LevelManager class within the LevelManager.js file. It is a simple manager class for spawning pickups and defining the level definitions that the game uses to generate the various elements like level length, number of spawns, types of enemies to spawn, spawn frequency and if a level has a boss or not.

This class is used by the index.html file to access the level definitions as well as the management of any powerups which have spawned as well as the addition of powerups to the game.

SpawnManager.Js

This file contains the two classes for managing the enemies in the game. The SpawnManager deals with creating spawns based on some random parameters determining enemy count, enemy type and spawn location. Additionally, it disposes of spawns when all enemies are killed in the spawn. The second class is the Spawn class which contains a set of enemies, a sprite key to load them with and a pattern to make them follow. This class manages the enemy movement and the time between spawning an enemy from its location.

The Spawn manager is prompted by the main game loop to add a spawn to the game, which has its parameters defined by the LevelManager level definition. The spawn manager checks the spawns in its set for disposal then prompts the set of spawns to add an enemy if it is time then updates each enemy position then finally has a check to determine if an enemy should shoot towards the player.



Player.js

The Player.js file is what initializes the player when the game loads, manages the player resources, and takes the players input and operates on it. This is done across three separate classes but this file only contains the Player class which contains the Weapon which then contains the Gun class which are specified in the Weapon.Js file. The player Class Specifically initializes the player game object along with the bombs and shield that the player uses and takes the keyboard input to move the player, shoot and drop bombs. These resources are managed within the class and can be operated on by the main game loop.

When the Player object is initialized, the Weapon class is added as a property, the sprites are created and the resources such as life and bombs are displayed on screen. On every update the bomb's explosion is updated if it was used, the player's velocity is reset based on input, as well as the shield, and bullets are spawned using the weapon class went the fire input is pressed.

Weapon.js

The Weapon.js file contains the classes for the Weapon and Gun objects which the Player class uses to manage their ammo and the type of firing pattern they use. Each cycle of the game loop that the player is shooting, the Weapon class checks if the player can fire a bullet and if so, the Weapon class uses the gun class to spawn a bullet or bullets into the pattern which is specified by the powerup that the player has aquired.

Boss.js

This file contains the two object Classes for creating the two bosses for the game. Both classes contain the definition for the respective boss sprite and boss components as well as their bullets. Both bosses are controlled by the main game loop simply by calling the update function within the boss's class.

The first boss is a helicopter. It has 3 states of either, moving or stopped and can be shooting when in either state. The helicopter class defines the boss sprite, the set of bullets that the boss uses, as well as the various parameters defining it's movement. The boss cycles between moving to a destination and waiting at the destination for one second while shooting directly at the player. The update function checks every loop the parameters to determine if it is time to move or time to shoot. The shoot function spawns a bullet, when it is time, directly at the player's current location in bursts of ten bullets every 4 seconds.

The second boss is a large gunship with 6 independently moving guns each with a set of bullets. This enemy is motion less but has a large heath pool and fires many bullets in a variety of patterns. Every few seconds this boss chooses a pattern to use when firing. The update function works though for each gun, spawn a bullet according to the pattern which was defined. After firing 50 bullets between all the guns they stop shooting for four seconds and selects a new pattern for the next burst.

Connection.php, HighscoreQuery.php, InsertHighScore.php

These three simple files each perform an operation on the database. Connection creates a connection to the database and is use in both highscoreQuery.php and insertHighScore.php. The highscoreQuery.php file is used to get the full table of highscores to be displayed in the html. It sends a query to the database to retireve the data and build the html table by echoing the results into table cells. The insertHighScore.php file requires two parameters for score and level to insert into the database and responds back to the caller with a response code.