**Programmer’s Guide**

Project Name: Fraction Runner

Team Name: Team DBA

Team Members:

* Gregory Shelton gpsc7c@umsl.edu
* Maija Garson mmgzzn@umsl.edu
* Kayla Thurman kethurman@mail.umsl.edu
* Sedaf Shakeel ssmkh@missouri.edu
* James Platt jbpkcd@mail.umsl.edu

Revision History:

04/04/2023 – 1st Draft

4/11/2023 – Revision, publish to GitHub

4/12/2023 – Revision per team discussion

Section 1: What a Programmer should know about Fraction Runner

Programming the Fraction Runner game: The game is built using HTML, CSS, and JavaScript. The uses a simple game loop to update the game state and render the graphics.

To program the Fraction Runner game, you will need a basic understanding of HTML, CSS, and JavaScript. It is recommended to use an integrated development environment (IDE) to write and test your code.

Here are the basic steps to programming the Fraction Runner game:

1. Create the HTML structure for the game. This includes the game canvas and any necessary elements, such as the score display.

2. Style the game using CSS. This includes setting the dimensions of the game canvas, styling the characters, and adding any necessary background images or colors.

3. Use JavaScript to create the game loop. This loop should update the game state (the position of the characters and the obstacles) and redraw the game graphics.

4. Implement the game logic. This includes detecting collisions between the character and the obstacles, updating the score, and ending the game when the character collides with an obstacle.

5. Add keyboard controls for the character. This includes detecting when the space bar or down arrow key is pressed and updating the character’s position accordingly.

6. Taking a user input to create a repeating decimal. These numbers are displayed on the ground as the character runs.

Pro Tips: Here are some tips and tricks to keep in mind when programming the Fraction Runner game.

1. Use requestAnimationFrame() to create a smooth game loop that runs at a consistent frame rate. It is the best choice because it can optimize system and browser resources.

2. Implemet collision detection algorithms, such as AABB or SAT. They can detect collisions between the character and obstacles.

3. Use a sprite sheet / Free Texture Packer to animate the character and obstacles.

4. Be prepared to buy a hosting service from a provider like hostgator.

5. Create 2 Databases by the use of MySQL to store the high score, fractions and user information database between game sessions. We choose this method because

locally hosting the database off-server is not a valid option in this case, and the numbers and user data need to be stored on separate tables to

prevent column bloat. We are using mysqli within php to access the server.

[IMPORTANT]

Firing query "Database Setup Code" in dba/codebase/database using MySQL Workbench or similar is REQUIRED to set up a local database.

For purposes of our current endeavor, everything here should already be handled, but note usage in case of errors

mysql database has two important user types, "root" (administration, passcode is set as "VfX!565WW!t552") intended to be set with all permissions,

and "siteuser" (average access to database, passcode set as "edcvfr43edcvfr4") intended to be set with permissions to DELETE, INSERT. SELECT, and UPDATE records

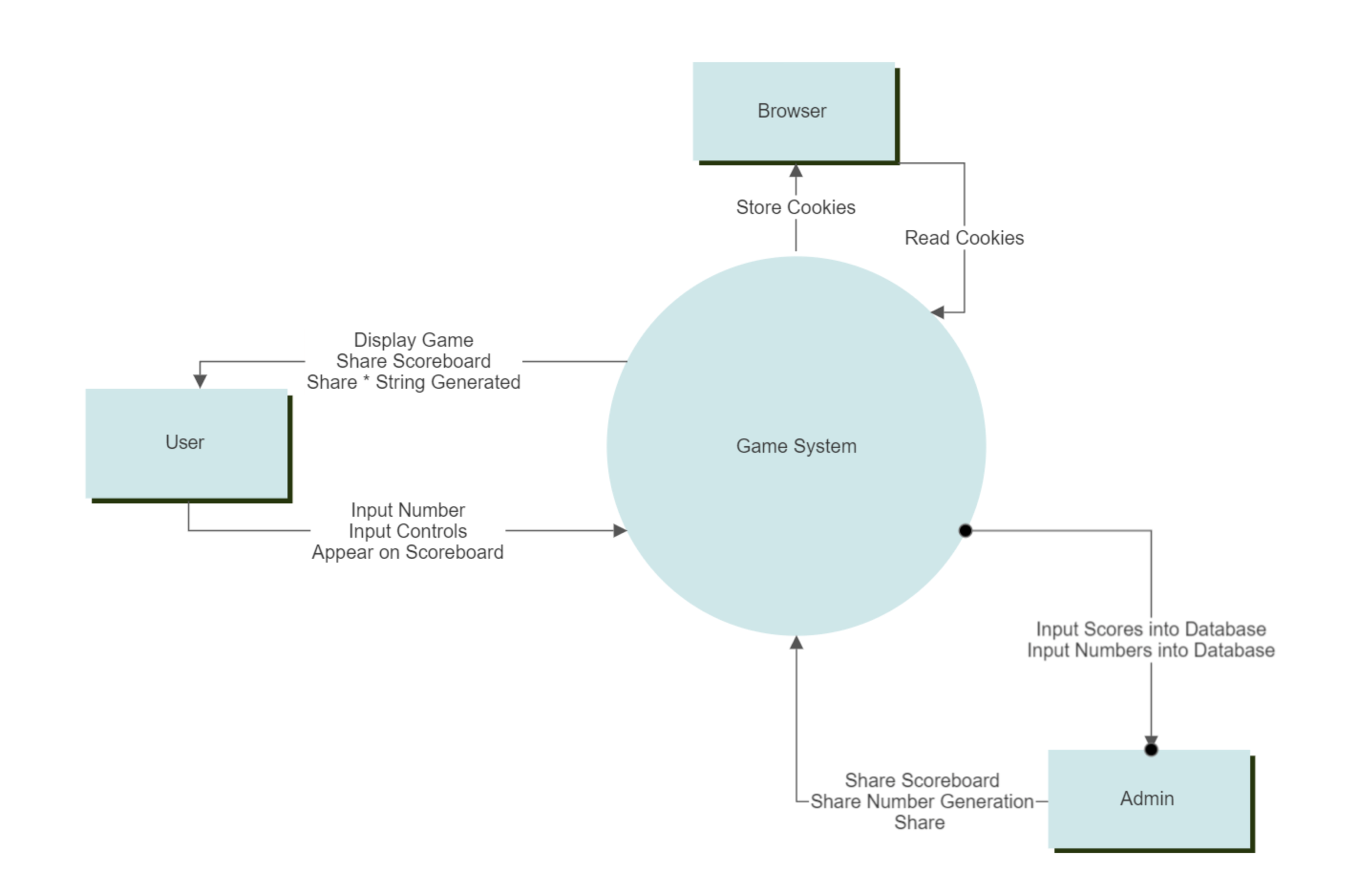
Finally, intended servername should be at "127.0.0.1", for testing, Machine Local Network IP (usually 198.68.0.\*) Or webserver IP ,subject to change based on

webserver settings

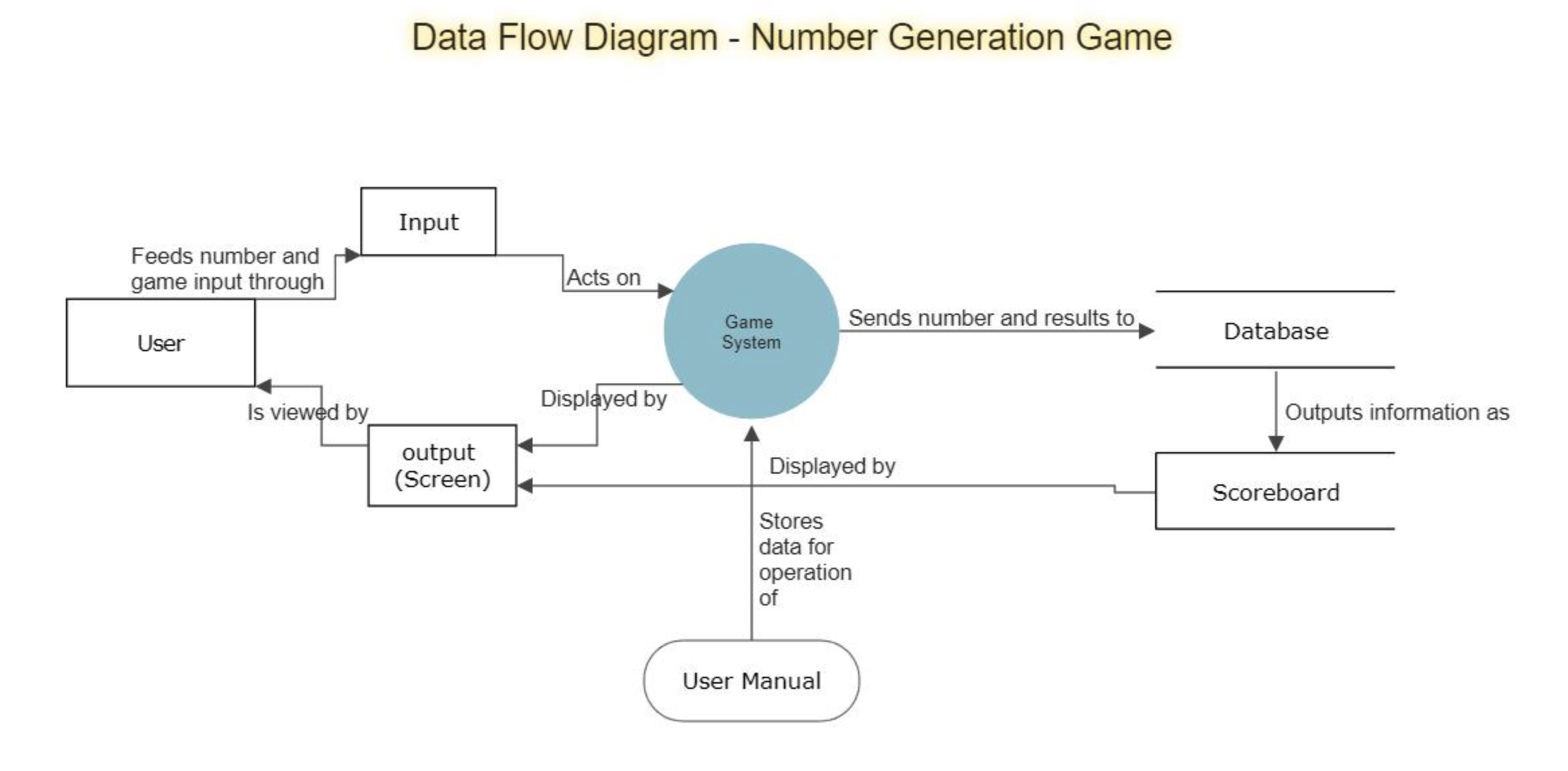
Conclusion: The Fraction Runner game is a simple yet addictive game that is also educational. By following the steps outlined in this guide, you can implement and troubleshoot/test the game using HTML, CSS, PHP, MySQL, and JavaScript.

Thank you for playing Fraction Runner!

Section 2: High Level Design



Section 3: More Detailed Designs



Section 4: Installation Instructions

Fraction Runner game runs in a web browser. The installation instructions are quite straight forward. You will simply open a modern web browser. Google Chrome is recommended.

Please navigate to our site: <https://fractionrunnerpro.com>

You will then be brought to our homepage where you will have several options including to start the game.

Appendix A: Implementation Code

**WEB PAGE**

HTML -  
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Document</title>

<link rel="stylesheet" href="style.css" />

</head>

<body>

<script src="scripts.js"></script>

<div class="game">

<div id="dino"></div>

<div id="cactus"></div>

</div>

</body>

</html>

CSSS

.game {

width: 600px;

height: 200px;

border: 1px solid black;

margin: auto;

}

#dino {

width: 50px;

height: 50px;

background-image: url(img/trex.png);

background-size: 50px 50px;

position: relative;

top: 150px;

}

.jump {

animation: jump 0.3s linear;

}

@keyframes jump {

0% {

top: 150px;

}

30% {

top: 130px;

}

50% {

top: 80px;

}

80% {

top: 130px;

}

100% {

top: 150px;

}

}

#cactus {

width: 20px;

height: 40px;

position: relative;

top: 110px;

left: 580px;

background-image: url("img/cactus.png");

background-size: 20px 40px;

animation: block 1s infinite linear;

}

@keyframes block {

0% {

left: 580 px;

}

100% {

left: -20px;

}

}

JavaScript -

Enemy creation:

class Enemy {

constructor(x, y, width, height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

this.isAlive = true;

}

draw() {

// Code to draw the enemy on the canvas

}

takeDamage() {

this.isAlive = false;

this.die();

}

die() {

// Code to remove the enemy from the canvas and update the game state

}

}

Character creation:

class Character {

constructor(x, y, width, height, maxHealth) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

this.isJumping = false;

this.jumpSpeed = 10; // The speed at which the character jumps

this.jumpHeight = 100; // The maximum height of the character's jump

this.jumpDuration = 20; // The number of frames the jump will take

this.jumpFrames = 0; // The number of frames the character has been jumping

this.jumpDirection = 1; // 1 means the character is going up, -1 means the character is going down

this.isDucking = false;

this.health = maxHealth;

}

jump() {

if (!this.isJumping && !this.isDucking && !this.isAttacking) {

this.isJumping = true;

this.jumpFrames = 0;

this.jumpDirection = 1;

}

}

update() {

if (this.isJumping) {

this.jumpFrames++;

// Calculate the character's vertical position based on the current jump frame

const yDelta = this.jumpSpeed \* this.jumpDirection;

const newY = this.y - yDelta;

// If the character has reached the maximum jump height, start descending

if (this.jumpFrames >= this.jumpDuration || newY <= this.jumpHeight) {

this.jumpDirection = -1;

}

// If the character has landed, reset the jump state

if (newY >= 200) {

this.isJumping = false;

this.jumpFrames = 0;

this.jumpDirection = 1;

}

// Update the character's position

this.y = newY;

}

}

duck() {

if (!this.isJumping && !this.isDucking && !this.isAttacking) {

this.isDucking = true;

this.height = this.height / 2; // reduce the character's height to make it look like it's ducking

}

}

standUp() {

this.isDucking = false;

this.height = this.height \* 2; // restore the character's original height

}

isTouching(obj) {

return (

this.x < obj.x + obj.width &&

this.x + this.width > obj.x &&

this.y < obj.y + obj.height &&

this.y + this.height > obj.y

);

}

attack(enemy) {

if (!this.isJumping && !this.isDucking && !this.isAttacking) {

this.isAttacking = true;

// attack code here, for example:

if (this.isTouching(enemy)) {

enemy.takeDamage();

}

}

}

takeDamage() {

if(this.isTouching(enemy))

this.health -= 10; // Character loses 10 health points when hit by an enemy

if (this.health <= 0) {

this.die();

}

}

die() {

// Code to handle the character's death

}

handleKeyDown(event) {

if (event.code === 'Space') {

this.jump();

} else if (event.code === 'ArrowDown') {

this.duck();

}

}

handleKeyUp(event) {

if (event.code === 'ArrowDown') {

this.standUp();

}

}

// other methods for drawing the character and handling collisions with other objects

}

const character = new Character(100, 200, 50, 100); // example width and height

document.addEventListener('keydown', (event) => {

character.handleKeyDown(event);

});

document.addEventListener('keyup', (event) => {

character.handleKeyUp(event);

});

PHP –

<!DOCTYPE html>

<?php

include './ScoreServerConnect.php';

?>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Document</title>

<link rel="stylesheet" href="style.css" />

</head>

<body>

<script src="scripts.js"></script>

<div class="game">

<div id="dino"></div>

<div id="cactus"></div>

</div>

<div>

<a href="../Homepage/GroupIntroPage.php"><button>Front Page</button></a>

</div>

<div>

<a href="../Scorepage/ScorePage.php"><button>Extended Scoreboard</button></a>

</div>

<?php /\*

#This is test code for later insertion of data

$username = 'test';

$password = 'password';

#this line creates the instruction to be sent

$sql = "INSERT INTO scoreboard\_dba.users VALUES (0,2,$username,$password,2,'001001001');";

#This line sends the instruction, success line can be changed, and sends the error otherwise

if ($dbconn->query($sql) === TRUE) {

echo "New user entry created successfully";

} else {

echo "Error: " . $sql . "<br>" . $dbconn->error;

}

#This id is a line that pulls information

$sql = "SELECT \* FROM scoreboard\_dba.users"

$dbconn->close();\*/

?>

<?php

#SET @r=0;

#UPDATE table SET Ranking= @r:= (@r+1) ORDER BY Score DESC;

?>

</body>

</html>

**DATABASE**

Database Setup:

#initial creation of database, drop is delete in this case, use states we're using it as the base database going forwards

DROP DATABASE IF EXISTS `scoreboard\_dba`;

CREATE DATABASE `scoreboard\_dba`;

USE `scoreboard\_dba`;

#character sets

SET NAMES utf8mb4 ;

SET character\_set\_client = utf8mb4 ;

#creation of an actual table within the database, users is the database name

CREATE TABLE `users` (

#Variable/column name/ids and rules

#NOT NULL

`user\_id` int NOT NULL AUTO\_INCREMENT,

`user\_name` varchar(50) NOT NULL,

`user\_score` bigint,

`password` varchar(50) NOT NULL,

`digits` varchar(9),

#`time\_set` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`user\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `fractions` (

#Variable/column name/ids and rules

#NOT NULL

`digits` varchar(9),

`fraction` decimal(10,9) CHECK(fraction>0) CHECK(fraction<1),

`divisor` int,

PRIMARY KEY (`digits`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

Database Trigger Code:

SELECT \* FROM scoreboard\_dba.users;

DELIMITER $$

CREATE TRIGGER trigger1

BEFORE INSERT

ON users

FOR EACH ROW

BEGIN

SELECT COUNT(\*) INTO @count FROM users;

IF @count >= 10000 THEN

DELETE FROM users

WHERE user\_rank = (SELECT min(user\_rank) FROM users);

END IF;

END

$$

DELIMITER ;

Appendix B: User Manual

**User Manual**

Welcome to Fraction Runner, an educational running game!

In this game, the object is to see how long you can stay alive.

Upon start, the player is asked to write a number.

The game puts player’s number in a fraction as the numerator over the same number of 9s in denominator.

(Example: 443 becomes 443/999)

The division produces a string of repeating decimals which will be displayed as the ground.

(From our earlier example: 443/999 becomes 0.443443443...)

The player runs on top of the numbers using Jump, Duck, or Attack to stay alive.

The player that stays alive the longest time is the winner.

When running, check out the repeating decimals.

On the Home Screen you will see three choices:

Fraction Runner – select when you are ready to begin the game

Introduction Page – click to learn more about the game and its developers

Top 100 Scoreboard – pick this to see who has the high score

Controls:

Each character can perform three different actions:

\*Jump – spacebar or click jump

\*Attack – enter or click attack

\*Duck – down or click duck

Gameplay:

At the start, the player is asked to select a character.

There is no skill difference between the characters.

When starting the game, the player must pick a number up to 999,999,999.

After entering the number, the running game begins.

The object of the game is to see how long the player can stay alive.

The timer starts at the beginning of the game.

Once the player has been hit by an object, the game is over.

The time is calculated and added to the Scoreboard.

Tips and Tricks:

\*There are three different obstacles. Each one can only be defeated by the correct action.

\*Use Jump when a hole appears

\*Use Duck when a bat is flying

\*Use Attack to break a wall

\*The runner will speed up as time continues. Stay alert!

We Thank you for playing Fraction Runner.

Appendix C: Test Plan

Our test plan involves testing for the following:

\*MySQL database

\*JavaScript

\*CSS

\*HTML

\* Fraction Runner is still in development. The Programmer’s Guide will be updated accordingly.