**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

04/11/2023 – Revision, publish to GitHub

04/12/2023 – Revision per team discussion

04/14/2023 – Revision per team discussion

05/02/2023 – Screenshots and polishing

5/09/2023 – Added screenshots

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. Pull files from GitHub.

2. Get a domain name and a web host.

3. Recreate the file structure inside code bases / webpages (directory / folders)

4. After testing on local machine, export database using MySQL server 8.0.

5. Change password, hostname and database name listed in the scoreDatabaseFunctions.php file under function()makeConnection.

6. If webhost errors for requirements, all file locations referenced in code must be changed to fix this.

7. Create 2 Databases on web server or local MySQL by use of the included by theMySQL database set up code file in codebase / database. This will create the table that will 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 MySQL within php to access the server.

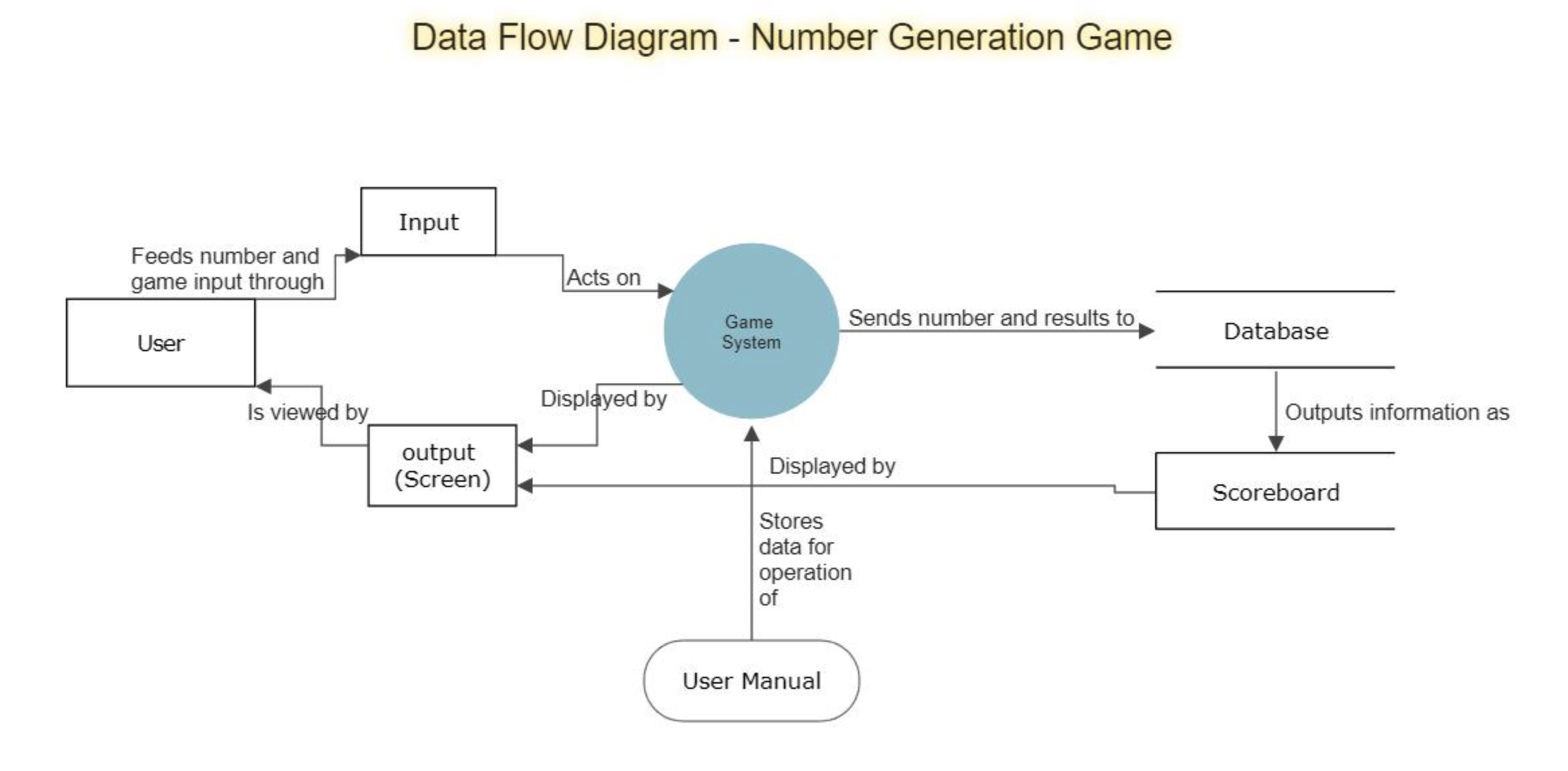
[IMPORTANT]

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

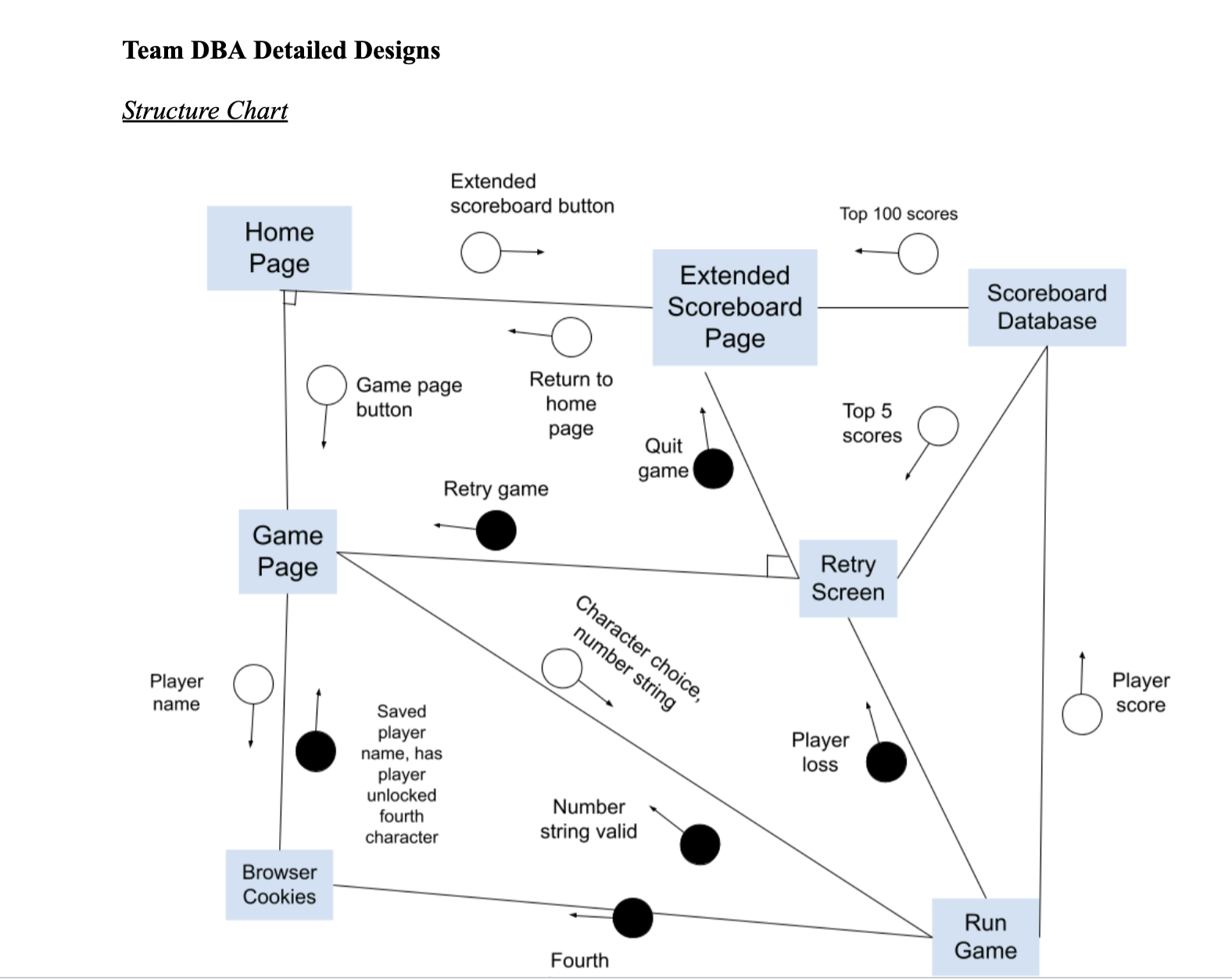
For purposes of our current endeavor, everything here should already be handled, but note usage in case of errors mysql database has two important user types, "root" (administration, passcode is set as "VfX!565WW!t552") intended to be set with all permissions, and "siteuser" (average access to database, 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.

Section 2: High Level Design



Section 3: More Detailed Designs



Pseudocode Detailed Design for Fraction Runner  
Front page:  
CSS: all centered  
<header (Welcoming)>  
<close header>  
<introduction to concept and game>  
<link to video>  
<close link>  
<close intro>  
<images>  
<static images>  
<close static images>  
<gif of game>  
<close gif of game>  
<close images>  
<container>  
<button>  
<link to game>  
<close link>  
<close button>  
<button>  
<link to extended scoreboard>  
<close link>  
<close button>  
<close container>  
<footer>  
<close footer>  
Game page:  
CSS: all centered  
<header (Welcoming)>  
<close header>  
<game border>  
<game window>  
<close window>  
<close border>  
<container>  
<button>  
<link to intro>  
<close link>  
<close button>  
<button>  
<link to extended scoreboard>  
<close link>  
<close button>  
<close container>  
<footer>

<close footer>  
Extended Scoreboard Page:  
CSS: all centered  
<header (Welcoming)>  
<close header>  
<score border>  
<scoreboard table (100 rows linked to database)>  
<close table>  
<close border>  
<container>  
<button>  
<link to intro>  
<close link>  
<close button>  
<button>  
<link to game>  
<close link>  
<close button>  
<close container>  
<footer>  
<close footer>  
JS:  
Javascript psuedocode:  
Function to layout data(information: table name, rank, name, points)  
Variables:  
Table name  
Table row  
Table Divider (Rank)  
Table Divider (Name)  
Table Divider (points)  
Table Divider (Repeating String Generated)  
Set dividers = children of row  
Set row = child of table  
Function to repeat above function 100 times onload (tablename, name, points)  
if(i = 0, i< 100, i++){  
Function to layout data(tablename, rank(i), name at rank I, points at name at rank i  
}  
Game:  
Game start  
Print brief explanation of math that was referenced on the intro page

Event listener for pressing of mute button  
IF audio is NOT muted  
mutes audio  
ELSE  
unmutes audio  
Event listener for pressing of quit button  
Pauses gameplay  
Asks confirmation that player wants to return to home page  
IF player clicks YES to return to home page  
returns user to intro page  
IF player clicks NO to continue game  
resume gameplay  
Cookie check  
IF no cookie  
input window for name to put on scoreboard  
IF cookie present  
checks for unlocked fourth character  
Number input window, onhover:  
add color around box  
onclick: change color around box  
Event listener for number input window  
IF number input is out of range (<1 or >9)  
display some appropriate error messaging  
allow user to redo input  
IF number is valid  
Generate repeating decimal  
Count number of significant digits in number  
Divide number by equal number of 9s (ex. 221332/999999)  
Show brief fake load screen showing the number get generated in a flashy way.  
proceed with related game logic  
Number input window, onhover:  
add color around box  
onclick: change color around box  
Event listener for character select to choose avatar  
Onclick: sets variable that determines character  
Sets strings so that character sprites display correct character  
Event listener for GAME START button  
Onclick:  
start game  
begin generating ground made of numbers as well as obstacles

Event listener for jump:  
IF character is not jumping:  
trigger jump  
set character state to jumping  
when jump action is complete  
reset character state  
ELSE  
do nothing  
Event listener for duck:  
IF character is not ducking:  
trigger duck  
set character state to ducking  
when player releases duck button  
reset character state  
Event listener for attack:  
IF character is not dodging  
trigger attack  
set character state to attacking  
when attack action is complete  
reset character state  
ELSE  
do nothing  
event listener for minute passed:  
Check transition number  
If transition number > 5  
transition back to first background  
set transition number back to 1  
Trigger transition linked to number  
reset minute time for next transition  
event listener for obstacle collision with character:  
Trigger game over  
stop movement on page on collision  
set player state to loss  
Record score  
Checks cookies for high score on browser  
display top 5 scoreboard with retry/quit buttons  
Event listener for retry button, onclick:  
Reset score to zero  
Reset other relevant variables (character state, transition number,  
timer/elapsed time, repeating decimal, etc) back to initial values  
Reopens input window for repeating decimal and character select  
Event listener for quit button, onclick:  
Take user to extended scoreboard page  
Event listener for home button on extended scoreboard page  
Return user to homepage

scoreboard databases  
XML file including rules of database  
CREATE TABLE users (  
#Variable/column name/ids and rules  
#NOT NULL  
'user\_id' int(10) UNSIGNED NOT NULL AUTO\_INCREMENT,  
'user\_rank' int(10) UNSIGNED NOT NULL,  
'user\_name' varchar(50) NOT NULL,  
'user\_score' bigint(10) UNSIGNED,  
'fraction' decimal(13,12) UNSIGNED CHECK(fraction>0) CHECK(fraction<1),  
'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;  
Stored in above cookie  
Accessed by:  
Scoreboard page  
Game Scoreboard  
MYSQLAdmin  
Trigger or loop to remove lowest score and lower all scores below an added score

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://www.fractionrunner.com>

Installation for a fresh install: Please see Section 1 which discusses installation of the game.

Permissions for a fresh install: Will need to create a user that has Admin privileges on all capabilities. A user profile that is based off the user listed in the scoreDatabaseFunctions.php file under function()makeConnection. With abilities to insert, delete, select, and update.

User can then be brought to our homepage where there are 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.

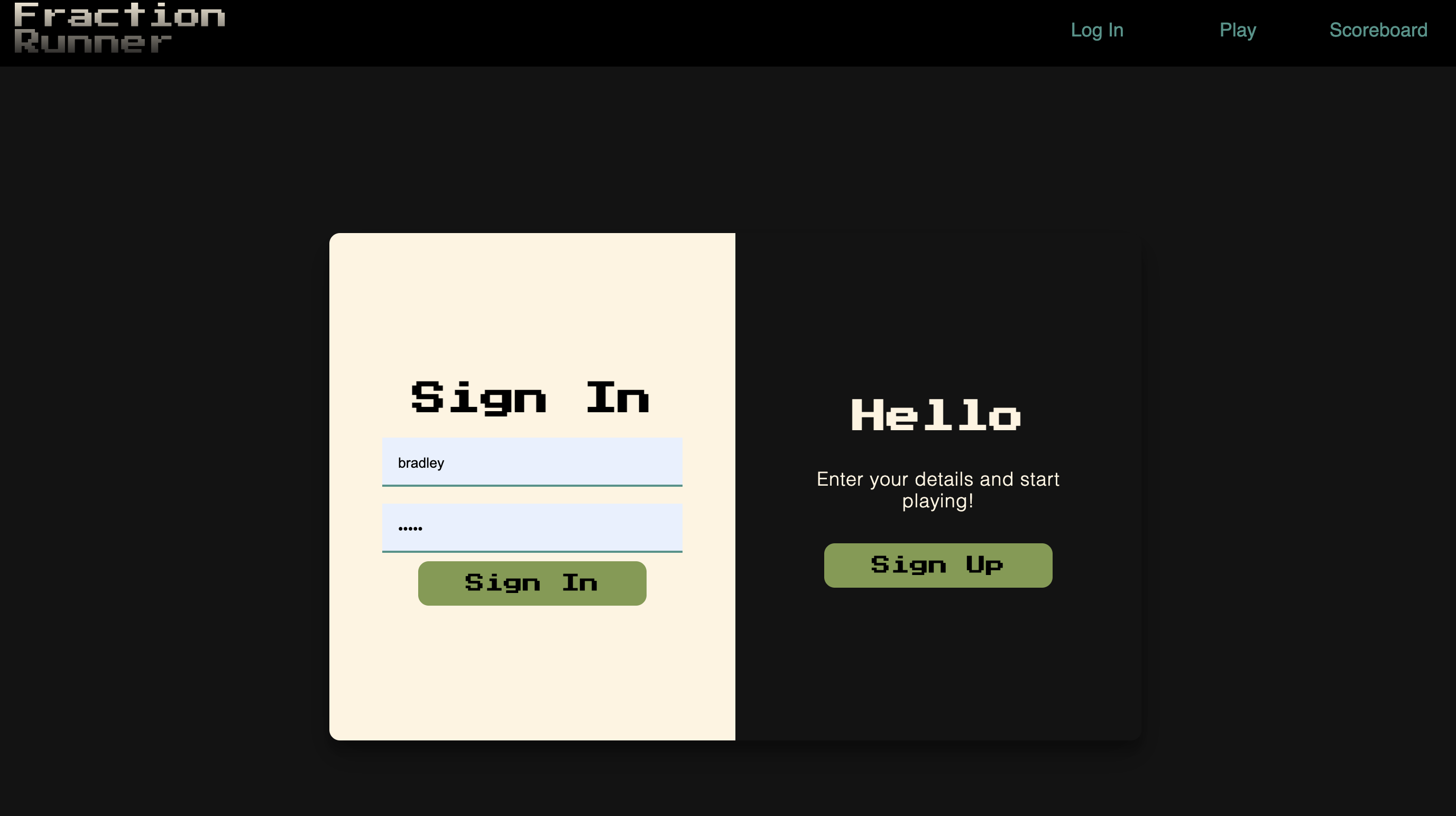
On the Home Screen you will see three choices:

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

Player will log in on the Log in Page:



Controls:

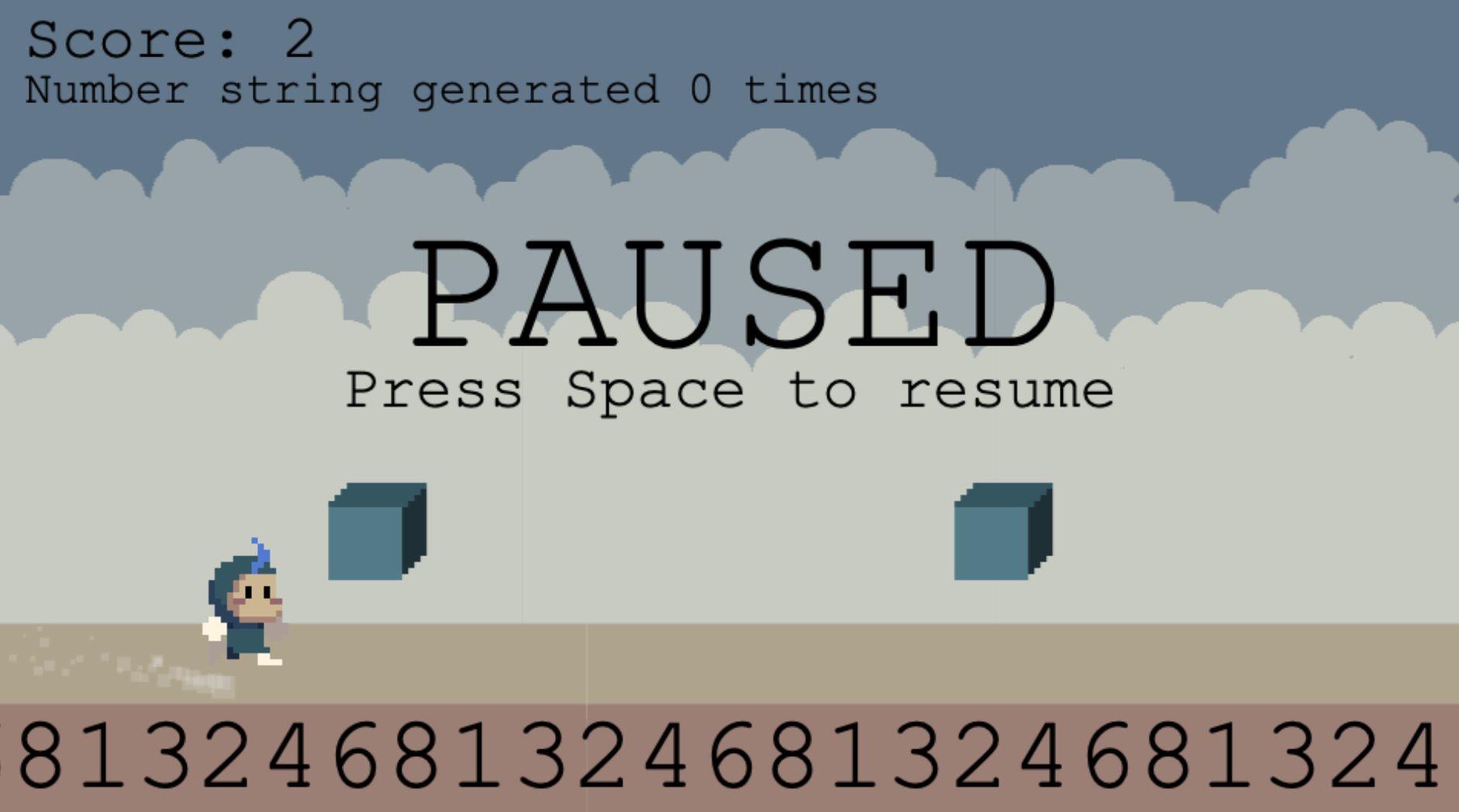
Each character can perform three different actions:

\*Jump – press the up arrow

\*Attack – hit the attack button

\*Duck – press the down arrow

(Screenshot of character needing to duck)



Gameplay:

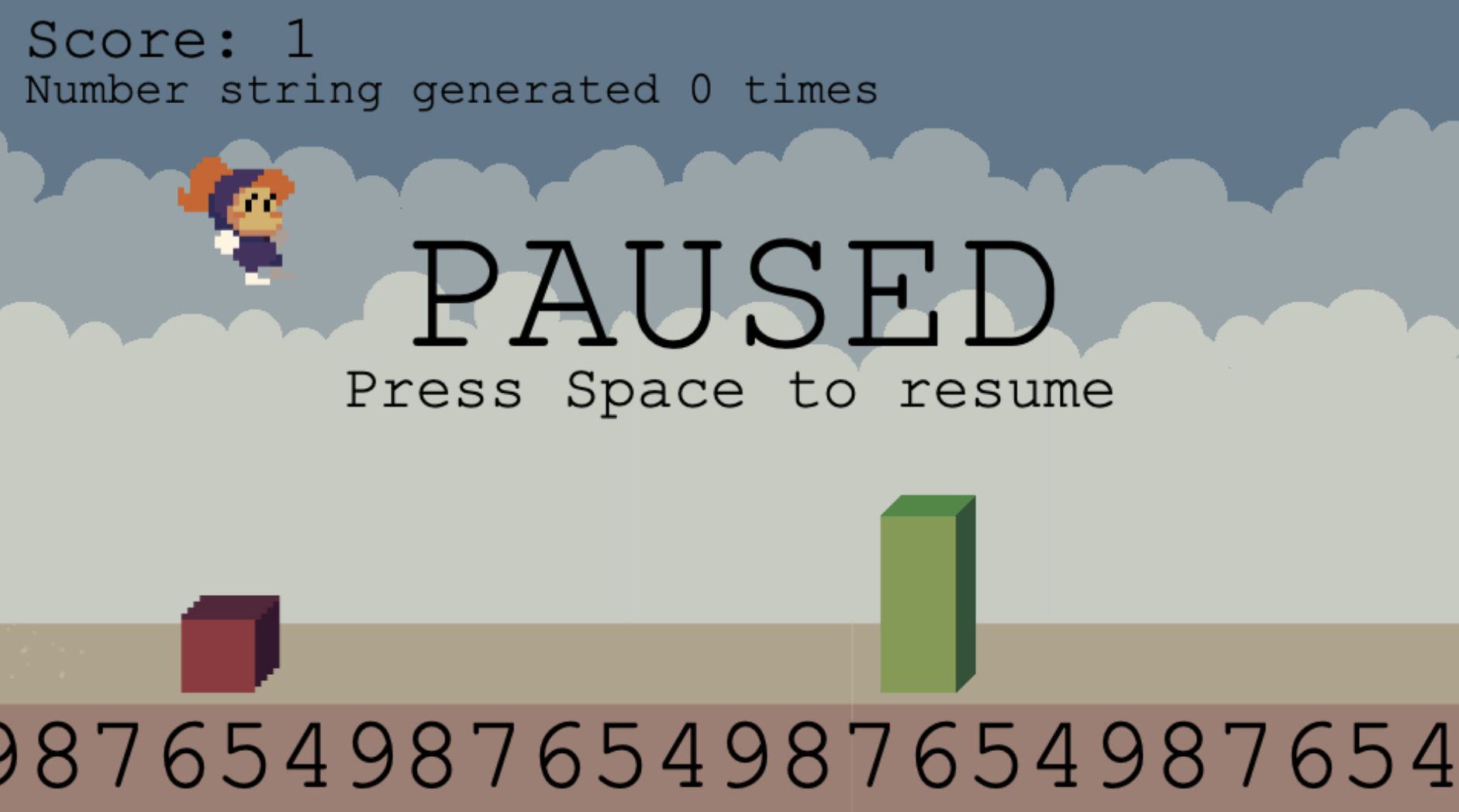
At the start, the player is asked to select a character.

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.

(Screenshot of character jumping over an obstacle)



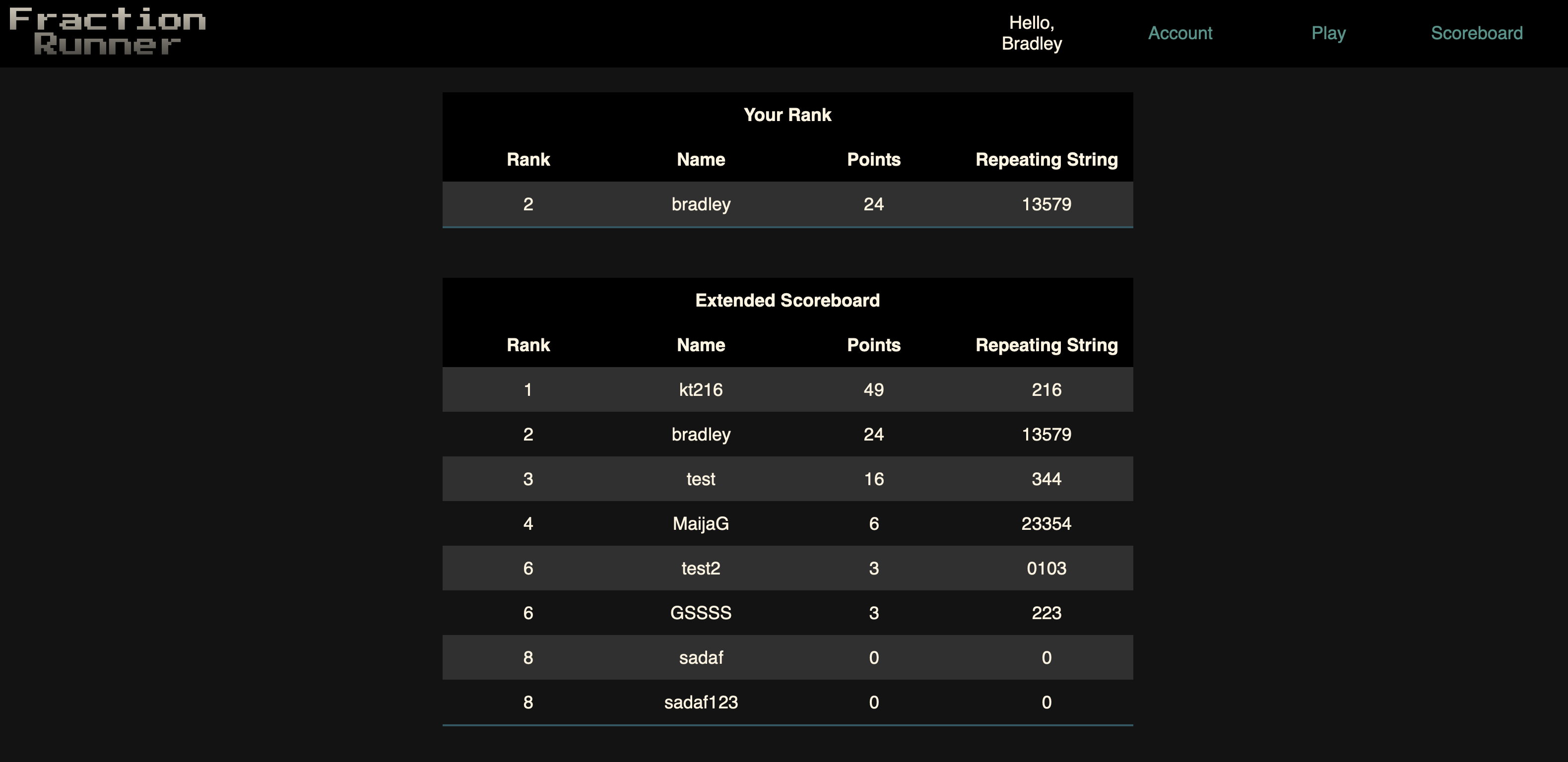
The object of the game is to see how long the player can stay alive.

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.

Fraction Runner Scoreboard:



Tips and Tricks:

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

\*Use Jump when a red obstacle appears

\*Use Duck to go under the blue block

\*Use Attack to break a green 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 – make sure insert, delete, update actions work from login and gamepage. Make sure Select actions work from score page and the gamepage.

\*JavaScript – make sure character actions match the buttons.

\*CSS – make sure the site loads correctly from the intro page. Check the loading, colors, size, etc, of the game during each step of input. Check scoreboard loads properly.

\*HTML – make sure site loads and structure is intact.