CSE104-WEB PROGRAMMING

Final Project Description

May 9, 2022

Yubo Cai, Junyuan Wang

CONTENTS

You can find the link to the website of the Project

1 INTRODUCTION

In this project, we tried to create an integrated website by using HTML, CSS and JavaScript, with a recreation of the classic trivia game with those tools. This project is organized by

1. Home page and navigation bar
2. Flappy bird start page and the game
3. 2048
4. Gobang Game

Yubo is charge for the Flappy bird and Gobang game also the Home page, Junyuan was working on the 2048 game which have more complex algorithms and the deployment of website.

2 FLAPPY BIRD START PAGE AND GAME PAGE

First, we can enter the start page of the Flappy bird. In this page we have the animation of the birds flying and the window shift up and down in certain period which is realized by the CSS code:

```
@keyframes birdmove {
    0% {
        top: 100px;
    }
    50% {
        top: 120px;
    }
    100% {
        top: 100px;
    }
}
```

Also we use the similar code with the 2 bird picture which the first is flying up and second is flying downwards in order to achieve the vision of the bird is flying. Then if you click on the start button it will jump to the game page.

The game page is organized by the score window above and the game window in the center. The algorithms of score counter is counted by the pillars we created.

```
let index = 0
if (isGameOver === false) { // we want to not create new pillars if
   the game is over
    setTimeout(generate_pillars, 4000);
    index += 1;
    console.log(index);
    // we put the index value to the console
    score.innerHTML = index;
}
```

To achieve the effect of the column and the bird moving, We fix the position of the bird so that the pillar moves from left to right according to a fixed ***px***. Also, we add the function ***jump***() so that for every click and press on the button the bird would jump for 53 ***px***.

The most struggling part is how to determine if the bird has hit the pillars. However, limited by the mathematical model we use, we are setting the physical model of the bird being a rectangle to make the determination. Therefore, there are still errors in the process of collision determination. We have tried to achieve better results by adjusting the parameters several times, here is the specific code.

```
pillar_left > 120 && pillar_left < 190 && bird_left == 140 && (
   bird_bottom < pillar_height + 298 || bird_bottom > pillar_height +
    gap - 68) //
/* add the similiar restriction to the top pillar */
|| bird_bottom === 150
```

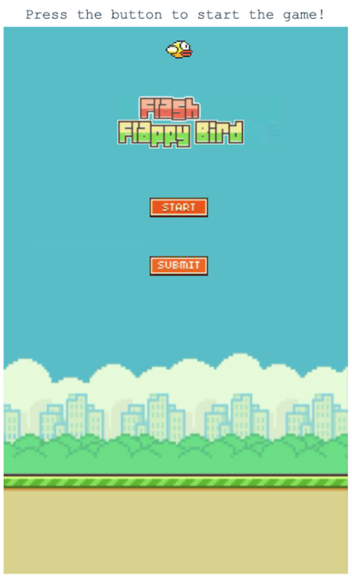


Figure 1: Start Page

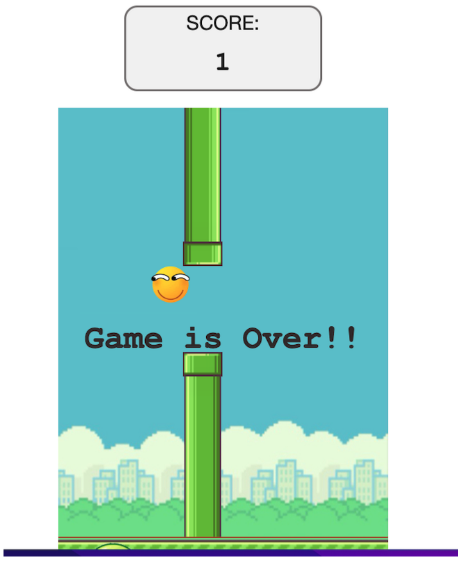


Figure 2: Game Page

If the bird hits the pillars means the game is over, Then text ***GameisOver*!** would jump out. We realized this first by setting a ***display* : *none*** on the CSS and when the game is over the

JavaScript function ***alert_function()*** would remove the class of the div in order to make the text visible and if you click the text it will refresh the website, or you can wait and do nothing the page would jump to the start page which is realized by set a time function.

```
function gameOver() {
    clearInterval(GametimeID);
    console.log("Game Over");
    isGameOver = true;
    // remove the eventlistener of click and keydown
    container.removeEventListener("click", jump);
    document.removeEventListener("keydown", jump);
    alert_function()
}

function alert_function() {
    // we add a div in the container with the class gameover
    const gameover = document.createElement("div");
    gameover.classList.add("gameover");
    gameover.innerHTML = "Game Over";
    container.appendChild(gameover);
    gameOver_display.classList.remove("display");
    setTimeout(() => {
        location.replace("start.html");
    }, 5000);
}
```

3 GOBANG

In this game we mainly have three parts, the drawing of the board, the placement of the chess pieces and the algorithms of how to distinguish the winner.

For drawing the board we simply use ***canvas*** in JavaScript with 2 loops to draw the lines of the borad. And also we can set the coordinate of the nodes in the board according to the lines. Then we use function ***oneStep()*** also the variable ***black*** to draw the chess pieces and distinguish whether is white or black. Then we create the function ***oneClick()*** to stimulate every step of GoBang and determind is white turn or black turn. Also, we try to use a 2-dimension number matrix to distinguish whether here is occuiped or not.

```
function oneclick(event) {
let x = event.offsetX; // now we set the location of the mouse
let y = event.offsetY;
let i = Math.floor((x) / 40);
let j = Math.floor((y) / 40);

if (chessBoard[i][j] == 0) {
    oneStep(i, j, black);
    turns();
```

```
            index += 1;
            console.log(index);
            if (black) {
                chessBoard[i][j] = 1;
            } else {
                chessBoard[i][j] = 2;
            }
            black = !black;
        } else {
            alert("I would recommend you to choose another place, since this
                place is already occupied !");
            return;
        }
```

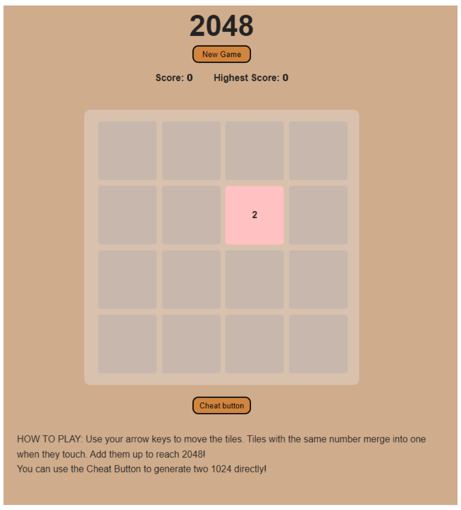
4
2048

In this 2048 game, we just use basic HTML, JavaScript, CSS to achieve the game. The design of the game are divided into 4 parts, the control of board and moving methods, the scoreboard, function buttons, and win or lose checking methods.

The 4×4 board are only made of HTML and CSS with absolute positions. It is more reliable for most devices to display correctly. We use a 4×4 2D array to store all the values inside the board. When an arrow key is clicked, the $\boldsymbol{moveLeft(), moveRight(), moveUp(),}$ and $\boldsymbol{moveDown()}$ methods will iterate by rows or columns, depending on the arrow key, and move each grid to correct positions. When the movement occurs, the there is a grid with the same value on another grid's moving direction, they will combined together. A new grid with the value 2 or 4 will be added to an empty place. After processing the movement, there is a $\boldsymbol{display()}$ method for showing the 2D array to the board to users.

The score function and function buttons are simple. We just simply update the score after each move and add an EventListener for the buttons. The lose checking method is relatively more difficult. One must check that every grid is empty and every grid has no grid with the same value close to it. After a lose situation detected, an explorer alert will be raised.

```
        function check_lose(){
            /* check if the game is lose */
            for(let i = 0; i < 4; i++){
                for(let j = 0; j < 4; j++){
                    if(chessboard[i][j] == 0){
                        return false;
                    }
                }
            }
            for(let i = 0; i < 4; i++){
                for(let j = 0; j < 3; j++){
                    if(chessboard[i][j] == chessboard[i][j+1]){
                        return false;
```



Score: 0 Highest Score: 0

HOW TO PLAY: Use your arrow keys to move the tiles. Tiles with the same number merge into one when they touch. Add them up to reach 2048!
You can use the Cheat Button to generate two 1024 directly!

Figure 3: 2048 Board

```
                }
            }
        }
        for(let i = 0; i < 3; i++){
            for(let j = 0; j < 4; j++){
                if(chessboard[i][j] == chessboard[i+1][j]){
                    return false;
                }
            }
        }
        return true;
    }
```

There are still lots of places for the game to be improved. The arrangement of the board could be changed for smaller screens or mobile devices. The movement of the grids could be accomplished by animation. Moreover, after learning back-end programming, a scoreboard for multiple users could be achieved.