# **PROGRAMMING FUNDAMENTALS**

# FINAL PROECT

# GROUP MEMBERS:

# MEMBER ONE:

MUHAMMAD KASHAN ZUHAIB (24f-0681)

# MEMBER TWO:

GHULAMMOHIYUDIN (24F-0834)

## SOURCE CODE:

#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <conio.h>

#include <iomanip>

#include <windows.h>

#include <fstream>

#include <sstream>

using namespace std;

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

// Game state variables

int lives = 3;

// Game board dimensions

const int BOARD\_WIDTH = 30;

const int BOARD\_HEIGHT = 20;

// Player position variables

int playerX = BOARD\_WIDTH / 2;

int playerY = BOARD\_HEIGHT - 1;

bool isJumping = false;

bool isSliding = false;

int jumpHeight = 0;

const int MAX\_JUMP\_HEIGHT = 3;

int slideCounter = 0;

const int SLIDE\_DURATION = 5;

string playerName;

// Computer player variables

int enemy = 0;

int compY = 0;

bool compActive = false;

// Obstacle positions

int obstacleX[3] = { 0 };

int obstacleY[3] = { 0 };

// Coin positions

int coinX[3] = { 0 };

int coinY[3] = { 0 };

// Game statistics variables

int coins = 0;

int score = 0;

int highestDistance = 0;

int distanceCovered = 0;

// Forward declarations

void fileHandling(string playerName, int distance);

void drawBoard();

void moveObstacles();

void moveCoins();

void movePlayer(char input);

void moveComputerPlayer();

bool checkCollision();

bool checkComputerCollision();

int collectCoins();

void initializeGame();

void game();

void showCredits();

void showScore();

void showMainMenu();

void showLogo();

void showInstructions();

int getKey();

void fileHandling(string playerName, int distance)

{

if (distance > highestDistance)

{

highestDistance = distance;

ofstream outFile("high\_scores.txt", ios::app);

if (outFile.is\_open())

{

outFile << playerName << "\t\t" << distance << "\t\t" << score << endl;

outFile.close();

outFile << "\t";

}

else {

cout << "Unable to write to file." << endl;

}

}

}

void drawBoard()

{

ostringstream buffer;

system("cls");

// Draw top border

for (int i = 0; i < BOARD\_WIDTH + 2; i++)

buffer << "-";

buffer << endl;

// Draw game board with elements

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

buffer << "|";

for (int j = 0; j < BOARD\_WIDTH; j++)

{

// Draw player

if (i == playerY && j == playerX)

{

if (isSliding)

buffer << "\_";

else

buffer << "0";

}

// Draw computer player if active and distance > 2000

else if (compActive && i == compY && j == enemy)

buffer << "U";

// Draw obstacles

else if ((i == obstacleY[0] && j == obstacleX[0]) ||

(i == obstacleY[1] && j == obstacleX[1]) ||

(i == obstacleY[2] && j == obstacleX[2]))

buffer << "X";

// Draw coins

else if ((i == coinY[0] && j == coinX[0]) ||

(i == coinY[1] && j == coinX[1]) ||

(i == coinY[2] && j == coinX[2]))

buffer << "\*";

else

buffer << " ";

}

buffer << "|" << endl;

}

// Draw bottom border

for (int i = 0; i < BOARD\_WIDTH + 2; i++)

buffer << "-";

buffer << endl;

cout << buffer.str();

buffer.str("");

buffer.clear();

// Game Stats

SetConsoleTextAttribute(h, 11);

cout << "Distance Covered: [ ";

SetConsoleTextAttribute(h, 15);

cout << distanceCovered << " ]" << endl;

SetConsoleTextAttribute(h, 11);

cout << "Score: [ ";

SetConsoleTextAttribute(h, 15);

cout << score << " ]" << endl;

SetConsoleTextAttribute(h, 11);

cout << "Coins: [ ";

SetConsoleTextAttribute(h, 15);

cout << coins << " ]" << endl;

SetConsoleTextAttribute(h, 11);

cout << "Lives Remaining: [ ";

SetConsoleTextAttribute(h, 15);

cout << lives << " ]" << endl;

}

void moveObstacles()

{

// Move all obstacles down the screen

for (int i = 0; i < 3; i++)

{

obstacleY[i]++;

// If obstacle goes off-screen, respawn it at top

if (obstacleY[i] >= BOARD\_HEIGHT)

{

obstacleY[i] = -i \* 3 - 1; // Stagger obstacle spawns

obstacleX[i] = rand() % BOARD\_WIDTH;

}

}

}

void moveCoins()

{

// Move all coins down the screen

for (int i = 0; i < 3; i++)

{

coinY[i]++;

// If coin goes off-screen, respawn it at top

if (coinY[i] >= BOARD\_HEIGHT)

{

coinY[i] = -i \* 5 - 3; // Stagger coin spawns

coinX[i] = rand() % BOARD\_WIDTH;

}

}

}

void movePlayer(char input)

{

switch (input)

{

case 'A':

case 'a':

playerX--;

break;

case 'D':

case 'd':

playerX++;

break;

case 'S':

case 's':

if (!isJumping && !isSliding) {

isSliding = true;

slideCounter = SLIDE\_DURATION;

}

break;

case 'W':

case 'w':

if (!isJumping && !isSliding) {

isJumping = true;

jumpHeight = 0;

}

break;

}

// Handle jumping mechanics

if (isJumping)

{

if (jumpHeight < MAX\_JUMP\_HEIGHT)

{

playerY--;

jumpHeight++;

}

else if (playerY < BOARD\_HEIGHT - 1)

{

playerY++;

if (playerY >= BOARD\_HEIGHT - 1)

{

isJumping = false;

playerY = BOARD\_HEIGHT - 1;

}

}

}

// Handle sliding mechanics

if (isSliding)

{

slideCounter--;

if (slideCounter <= 0)

{

isSliding = false;

}

}

// Boundary checks

if (playerX < 0)

playerX = 0;

if (playerX >= BOARD\_WIDTH)

playerX = BOARD\_WIDTH - 1;

}

void moveComputerPlayer()

{

// Computer follows player with a delay

if (compActive)

{

// Move horizontally toward player

if (enemy < playerX)

enemy++;

else if (enemy > playerX)

enemy--;

// Move vertically toward player

if (compY < playerY)

compY++;

else if (compY > playerY)

compY--;

}

else if (distanceCovered >= 2000)

{

// Activate computer player when distance exceeds 2000

compActive = true;

enemy = playerX;

compY = playerY - 3;

}

}

bool checkCollision()

{

// Check for collision with obstacles

for (int i = 0; i < 3; i++)

{

if (playerX == obstacleX[i] && playerY == obstacleY[i])

{

// If sliding, avoid collision with obstacle

if (isSliding)

return false;

else

return true;

}

}

return false;

}

bool checkComputerCollision()

{

// Check collision with computer player when active

if (compActive && playerX == enemy && playerY == compY)

{

return true;

}

return false;

}

int collectCoins()

{

int collectedCoins = 0;

// Check collision with each coin

for (int i = 0; i < 3; i++)

{

if (playerX == coinX[i] && playerY == coinY[i])

{

collectedCoins++;

coins++;

// Increase score for each coin collected

score += 10;

// Respawn coin at top of screen

coinY[i] = -i \* 2 - 1;

coinX[i] = rand() % BOARD\_WIDTH;

}

}

return collectedCoins;

}

void initializeGame()

{

// Reset game variables

lives = 3;

playerX = BOARD\_WIDTH / 2;

playerY = BOARD\_HEIGHT - 1;

isJumping = false;

isSliding = false;

compActive = false;

distanceCovered = 0;

coins = 0;

score = 0;

// Initialize obstacles and coins with random positions

for (int i = 0; i < 3; i++)

{

obstacleX[i] = rand() % BOARD\_WIDTH;

obstacleY[i] = -i \* 7; // Staggered to avoid immediate collisions

coinX[i] = rand() % BOARD\_WIDTH;

coinY[i] = -i \* 5 - 10;

}

}

void game()

{

srand(static\_cast<unsigned int>(time(0)));

initializeGame();

char input = 0;

while (lives > 0)

{

drawBoard();

moveObstacles();

moveCoins();

moveComputerPlayer();

// Check for object collisions

if (checkCollision())

{

lives--;

// Flash screen for collision feedback

system("color 4F");

Sleep(100);

system("color 0F");

if (lives <= 0)

break;

}

// Check for computer player collision

if (checkComputerCollision())

{

lives--;

// Flash screen for collision feedback

system("color 5F");

Sleep(100);

system("color 0F");

if (lives <= 0)

break;

// Reset computer position after collision

compY = playerY - 5;

}

// Handle coin collection

int coinsCollected = collectCoins();

if (coinsCollected > 0)

{

// Visual feedback for coin collection

SetConsoleTextAttribute(h, 14);

cout << "\nCoin collected! +" << (coinsCollected \* 10) << " points!" << endl;

SetConsoleTextAttribute(h, 15);

Sleep(50);

}

// Handle player input

if (\_kbhit())

{

input = \_getch();

movePlayer(input);

}

// Handle jumping mechanics

if (isJumping)

{

if (jumpHeight < MAX\_JUMP\_HEIGHT)

{

playerY--;

jumpHeight++;

}

else if (playerY < BOARD\_HEIGHT - 1)

{

playerY++;

if (playerY >= BOARD\_HEIGHT - 1)

{

isJumping = false;

playerY = BOARD\_HEIGHT - 1;

}

}

}

// Adjust game speed based on distance

int gameSpeed;

if (distanceCovered <= 1000)

gameSpeed = 120;

else if (distanceCovered <= 2000)

gameSpeed = 90;

else if (distanceCovered <= 3000)

gameSpeed = 60;

else if (distanceCovered <= 4000)

gameSpeed = 40;

else

gameSpeed = 30;

// Increase distance counter

distanceCovered += 10;

// Add points for distance covered

if (distanceCovered % 100 == 0)

score++;

Sleep(gameSpeed);

}

// Game over screen

system("cls");

SetConsoleTextAttribute(h, 12);

cout << "\n\n\n\t\t GAME OVER!" << endl;

SetConsoleTextAttribute(h, 14);

cout << "\n\t\t Distance covered: " << distanceCovered << endl;

cout << "\t\t Coins collected: " << coins << endl;

cout << "\t\t Final score: " << score << endl;

SetConsoleTextAttribute(h, 15);

// Update high score

fileHandling(playerName, distanceCovered);

cout << "\n\t\t Press any key to return to main menu...";

\_getch();

}

void showCredits()

{

system("cls");

SetConsoleTextAttribute(h, 11);

cout << setw(70) << "--CREDITS-- " << endl;

SetConsoleTextAttribute(h, 15);

cout << "\n Based on a project by:" << endl;

cout << " - KASHAN ZUHAIB-(24f-0681) " << endl;

cout << " - GHULAM MOHYIUDIN-(24f-0834) " << endl;

cout << "\n Features Implemented:" << endl;

cout << " 1. Player Movement with WASD Controls" << endl;

cout << " 2. Jumping and Sliding Mechanics" << endl;

cout << " 3. Coin Collection System" << endl;

cout << " 4. Obstacle Avoidance" << endl;

cout << " 5. Computer Player AI" << endl;

cout << " 6. Progressive Difficulty" << endl;

cout << " 7. High Score System" << endl;

cout << "\n Press any key to return to main menu...";

\_getch();

}

void showScore()

{

system("cls");

SetConsoleTextAttribute(h, 14);

cout << "\n\n\t\t ===== HIGH SCORES ===== \n\n";

SetConsoleTextAttribute(h, 15);

cout << "\tPlayer Name\tDistance\tScore\n";

cout << "\t---------------------------------\n";

ifstream infile("high\_scores.txt");

if (infile.is\_open())

{

string line;

while (getline(infile, line))

{

cout << "\t" << line << endl;

}

infile.close();

}

else

{

cout << "\tNo high scores recorded yet.\n";

}

cout << "\n\tPress any key to return to main menu...";

\_getch();

}

void showMainMenu()

{

system("cls");

system("Color 09");

cout << endl << endl;

cout << setw(105) << " <============> ENDLESS RUNNER! <============>" << endl;

cout << endl;

cout << setw(98) << "=======> MAIN MENU <=======" << endl << endl;

cout << setw(90) << "1 -- PLAY GAME" << endl;

cout << setw(90) << "2 -- INSTRUCTIONS" << endl;

cout << setw(90) << "3 -- HIGH SCORES" << endl;

cout << setw(90) << "4 -- CREDITS" << endl;

cout << setw(90) << "5 -- EXIT" << endl << endl;

cout << setw(90) << "Select Your Choice: ";

}

void showLogo()

{

system("cls");

cout << "\n\n";

system("Color 09");

SetConsoleTextAttribute(h, 11);

cout << "\t \_\_\_\_\_\_ \_ \_ \_\_\_\_\_ " << endl;

cout << "\t | \_\_\_\_| | | | | \_\_ \\ " << endl;

cout << "\t | |\_\_ \_ \_\_ \_\_| | | \_\_\_ \_\_\_ \_\_\_ | |\_\_) |\_ \_ \_ \_\_ \_ \_\_ \_\_\_ \_ \_\_ " << endl;

cout << "\t | \_\_| | '\_ \\ / \_` | |/ \_ / \_\_/ \_\_| | \_ /| | | | '\_ \\| '\_ \\ / \_ | '\_\_|" << endl;

cout << "\t | |\_\_\_\_| | | | (\_| | | \_\_\\\_\_ \\\_\_ \\ | | \\ \\| |\_| | | | | | | | \_\_| | " << endl;

cout << "\t |\_\_\_\_\_\_|\_| |\_|\\\_\_,\_|\_|\\\_\_\_|\_\_\_|\_\_\_/ |\_| \\\_\\\\\_\_,\_|\_| |\_|\_| |\_|\\\_\_\_|\_| " << endl;

SetConsoleTextAttribute(h, 15);

cout << "\n\n\t\t Press any key to continue...";

\_getch();

}

void showInstructions()

{

system("cls");

SetConsoleTextAttribute(h, 14);

cout << "\n\n\t\t ===== GAME RULES ===== \n\n";

SetConsoleTextAttribute(h, 15);

cout << " 1. Control your character using the following keys:\n";

cout << " - W: Jump over obstacles\n";

cout << " - A: Move left\n";

cout << " - S: Slide under obstacles\n";

cout << " - D: Move right\n\n";

cout << " 2. You start with three lives. You lose a life when you hit an obstacle or the computer player.\n\n";

cout << " 3. The game ends when you run out of lives.\n\n";

cout << " 4. Collect coins to increase your score. Each coin is worth 10 points.\n\n";

cout << " 5. The game speed increases as you cover more distance.\n\n";

cout << " 6. After 2000 meters, a computer player will appear and try to catch you.\n\n";

cout << " 7. You cannot pass through the walls on the sides of the screen.\n\n";

cout << " 8. Your high score will be saved with your name.\n\n";

cout << " Press any key to return to the main menu...";

\_getch();

}

int getKey()

{

return \_getch();

}

int main()

{

srand(static\_cast<unsigned int>(time(0)));

ShowWindow(GetConsoleWindow(), SW\_MAXIMIZE);

showLogo();

cout << "\n\n Please enter your name: ";

getline(cin, playerName);

while (true)

{

showMainMenu();

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

game();

break;

case 2:

showInstructions();

break;

case 3:

showScore();

break;

case 4:

showCredits();

break;

case 5:

system("cls");

cout << "\n\n\tThanks for playing Endless Runner!" << endl;

return 0;

default:

system("cls");

cout << "\n\tINVALID CHOICE! Please try again." << endl;

Sleep(1000);

break;

}

}

return 0;

}

# **OUTPUTS:**

# **LOGO:**

A screen shot of a computer

AI-generated content may be incorrect.

# **NAME INPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

# **MENU:**

A black screen with blue text

AI-generated content may be incorrect.

# **GAME:**

A black screen with a black border

AI-generated content may be incorrect.

# **SCORE:**

A screen shot of a computer

AI-generated content may be incorrect.

# **GAME RULES:**

A screenshot of a computer

AI-generated content may be incorrect.

# **Leaderboard:**

A black screen with a black border

AI-generated content may be incorrect.

# **CREDITS:**

A black screen with a black background

AI-generated content may be incorrect.

# **EXIT:**

A black screen with blue text

AI-generated content may be incorrect.

# **OVERALL VIDEO:**

# 