**PROJECT REPORT**

*Data Structure and Algorithms Lab (CSL-221)*



**FLAPPY BIRD GAME**

**BS(CS) – 3B**

*Group Members*

|  |  |
| --- | --- |
| **Name** | **Enrollment** |
| 1. M.Tariq Fahim Qureshi | 02-134231-078 |
| 1. Hamza Adnan | 02-134231-031 |
| 1. Mohammed-Ur-Rehman | 02-134231-035 |

**Submitted to:**

**Ma’am Rabia Amjad**

**BAHRIA UNIVERSITY KARACHI CAMPUS**

*Department of Computer Science*

# **ABSTRACT**

This project presents the development of a simple 2D game implemented in C++, resembling the classic "Flappy Bird" game. The game involves controlling a bird character to navigate through obstacles on the screen, with the objective of achieving the highest score possible. The project explores basic concepts of game development, including graphics rendering, user input handling, collision detection, and score tracking.

# **INTRODUCTION**

2D games have been an integral part of computer programming education, offering a practical platform for learning various programming concepts while providing an engaging and interactive experience. In this project, we aim to develop a simplified version of the popular "Flappy Bird" game using C++ and standard libraries. The game will involve guiding a bird character through a series of obstacles while scoring points based on successful navigation. Through this project, we seek to enhance our understanding of C++ programming, data structures, and basic game development principles.

**PROBLEM STATEMENT:**

1. Rendering Graphics:

* How can we render graphics on the console screen to create a visually engaging game environment?

1. User Input Handling:

* How can we implement responsive controls for user interaction, allowing the player to control the bird's movement?

1. Collision Detection:

* How can we accurately detect collisions between the bird character and obstacles to determine game outcomes?

1. Score Tracking:

* How can we track the player's score based on successful navigation through obstacles?

1. Game State Management:

* How can we manage the overall game state, including starting, playing, and ending the game?

**METHODOLOGY:**

1. Rendering Graphics:

* By utilizing functions provided by the Windows API for console graphics rendering and managing the display of game elements such as the bird character and obstacles.

1. User Input Handling:

* By utilizing functions for keyboard input handling provided by the Windows API to detect and respond to user input for controlling the bird's vertical movement.

1. Collision Detection:

* By implementing collision detection algorithms that compare the positions of the bird and obstacles on the screen and trigger appropriate game actions based on collision outcomes.

1. Score Tracking:

* By implementing a scoring system that increments the score each time the bird successfully passes through an obstacle, and displaying the current score on the screen during gameplay.

1. Game State Management:

* By designing a game loop structure that manages the flow of gameplay, including starting the game, updating game elements such as obstacles and the player's score, and ending the game when specific conditions, such as collisions, are met.

**PROJECT SCOPE:**

The project scope encompasses the development of a Flappy Bird game using C++ and  
integrating principles of Data Structures and Algorithms (DSA). The game will include basic  
functionalities such as bird movement, pipe generation, collision detection, scoring, and game  
over conditions. The scope also involves the application of DSA concepts to optimize game  
performance and maintainability.

# **CODE:**

#include <iostream>

#include <windows.h>

#include <math.h>

#include <time.h>

#include <conio.h>

#include <queue>

#include <vector>

using namespace std;

void goToXY(int x, int y) {

COORD coord = { (SHORT)x, (SHORT)y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

const int hurdleCount = 4;

struct Node {

int x;

int y;

Node\* next;

Node(int x, int y) : x(x), y(y), next(nullptr) {}

};

Node\* head = nullptr;

int screenWidth = 80;

int screenHeight = 25;

int hurdleGap = 10;

int betweenHurdleGap;

int birdX = 17;

int birdY = 15;

int jump = 4;

int score = 0;

priority\_queue<int> scores;

void insertNode(int x, int y) {

Node\* newNode = new Node(x, y);

if (!head) {

head = newNode;

}

else {

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

}

void deleteFirstNode() {

if (!head) return;

Node\* temp = head;

head = head->next;

delete temp;

}

void printHurdle() {

Node\* temp = head;

Node\* prev = nullptr;

while (temp) {

int count = 0;

for (int j = 0; j < screenHeight; j++) {

if (temp->y == j) count = hurdleGap;

if (count == 0) {

if (temp->x < screenWidth) {

goToXY(temp->x + 1, j);

cout << " ";

goToXY(temp->x, j);

cout << "\*";

}

}

else {

if ((count == 1 || count == hurdleGap) && temp->x < screenWidth) {

if (temp->x + 1 > 0) {

goToXY(temp->x + 1, j);

cout << " ";

}

if (temp->x - 1 > 0) {

goToXY(temp->x - 1, j);

cout << "===";

}

}

count--;

}

}

temp->x--;

if (temp->x == -1) {

int new\_x = screenWidth + betweenHurdleGap;

int new\_y = rand() % (screenHeight / 3) + hurdleGap;

if (prev) {

prev->next = temp->next;

}

else {

head = temp->next;

}

Node\* toDelete = temp;

temp = temp->next;

delete toDelete;

insertNode(new\_x, new\_y);

}

else {

prev = temp;

temp = temp->next;

}

}

}

bool collisionCheck() {

if (birdY == 0 || birdY + 3 == screenHeight) return true;

Node\* temp = head;

while (temp) {

if (temp->x == birdX &&

(birdY >= temp->y || birdY + 3 <= (temp->y + hurdleGap))) {

score++;

}

if (temp->x >= birdX - 5 &&

temp->x <= birdX &&

(birdY <= temp->y || birdY + 3 >= (temp->y + hurdleGap))) {

return true;

}

temp = temp->next;

}

return false;

}

void clearBird() {

goToXY(birdX - 5, birdY);

cout << " ";

goToXY(birdX - 5, birdY + 1);

cout << " ";

goToXY(birdX - 5, birdY + 2);

cout << " ";

}

void printScore() {

goToXY(0, screenHeight + 4);

cout << "Score: " << score;

}

void printBird() {

goToXY(birdX - 5, birdY); cout << "(\*>";

goToXY(birdX - 5, birdY + 1); cout << "/))";

goToXY(birdX - 5, birdY + 2); cout << "``";

}

void printRoad() {

for (int i = 0; i <= screenWidth; i++) {

goToXY(i, screenHeight); cout << "\_";

goToXY(i, screenHeight + 1); cout << "/";

goToXY(i, screenHeight + 2); cout << "=";

}

}

void play() {

score = 0;

birdX = 17;

birdY = 15;

head = nullptr;

betweenHurdleGap = (screenWidth / hurdleCount) + 10;

for (int i = 0; i < hurdleCount; i++) {

int x = betweenHurdleGap \* (i + 1);

int breakPos = rand() % (screenHeight / 3) + hurdleGap;

insertNode(x, breakPos);

}

printRoad();

while (true) {

if (GetAsyncKeyState(VK\_SPACE)) {

birdY -= jump;

}

printHurdle();

printBird();

printScore();

if (collisionCheck()) break;

Sleep(100);

clearBird();

birdY += 1;

}

}

void displayScores(priority\_queue<int> scores) {

system("cls"); // Clear screen

goToXY(30, 5);

cout << "Scores:" << endl;

int rank = 1;

while (!scores.empty()) {

goToXY(30, 5 + rank);

cout << rank << ". " << scores.top() << endl;

scores.pop();

rank++;

}

}

void showCredits() {

system("cls");

goToXY(30, 10);

cout << "Game developed by:" << endl;

goToXY(30, 12);

cout << "Tariq Fahim, 078" << endl;

goToXY(30, 14);

cout << "Hamza Adnan, 031" << endl;

goToXY(30, 16);

cout << "Mohammad ur Rehman, 035" << endl;

goToXY(30, 20);

cout << "Press any key to go back to menu...";

\_getch(); // Wait for user input

}

void showMenu() {

while (true) {

system("cls");

goToXY(30, 10);

cout << "1. Start Game" << endl;

goToXY(30, 12);

cout << "2. Show Credits" << endl;

goToXY(30, 14);

cout << "3. Exit" << endl;

goToXY(30, 16);

cout << "Enter your choice: ";

char choice = \_getch();

if (choice == '1') {

play();

scores.push(score);

displayScores(scores);

goToXY(30, 30);

cout << "Do you want to play again? (Y/N)";

char ch;

cin >> ch;

if (ch == 'N' || ch == 'n') {

break;

}

}

else if (choice == '2') {

showCredits();

}

else if (choice == '3') {

break;

}

}

}

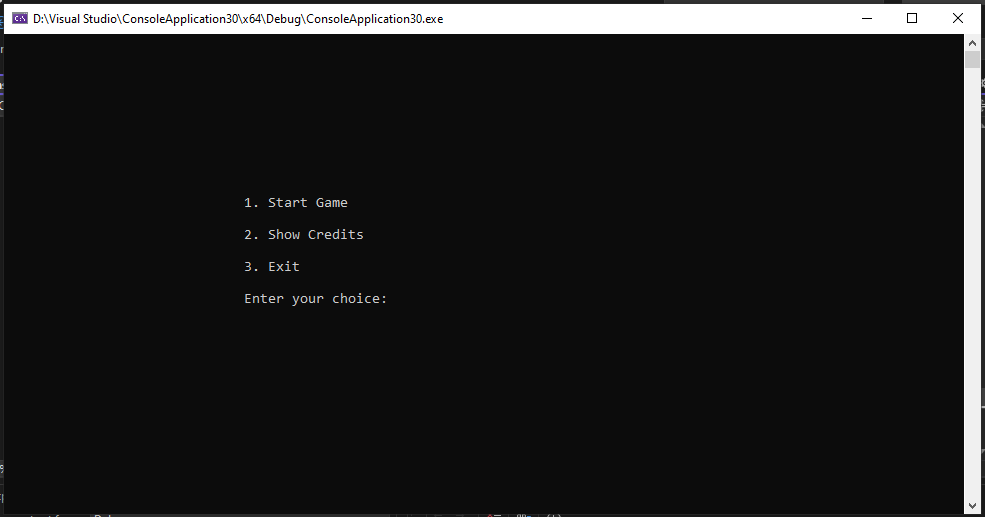
int main() {

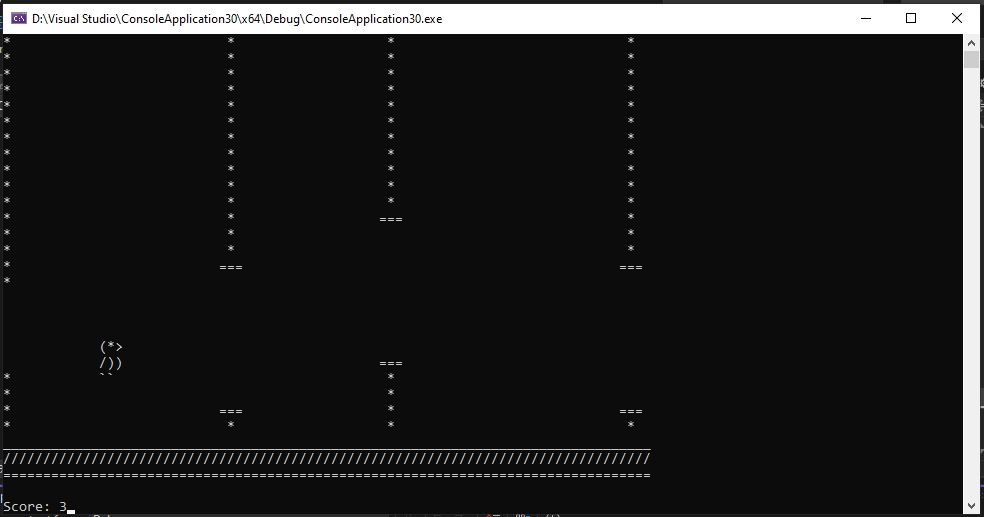
showMenu();

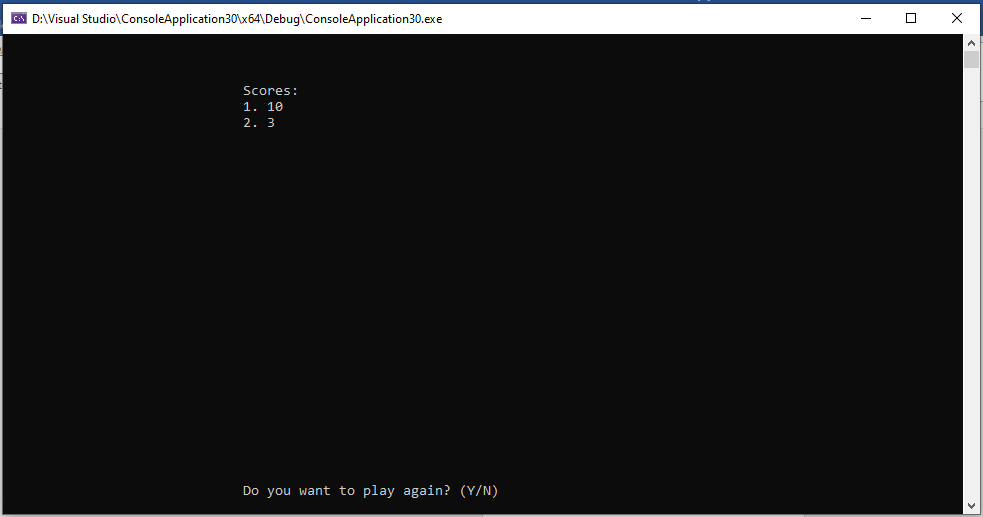
return 0;

}

# **OUTPUT:**









**Future Improvements:**

1. **Modularization: Break the code into smaller, more manageable functions to improve readability and maintainability. Separate functionalities such as obstacle generation, collision detection, and rendering into individual functions for better organization.**
2. **Enhanced Graphics: Explore libraries or frameworks that offer more advanced graphics capabilities, allowing for better visual representation of the game environment. Consider transitioning to graphical interfaces rather than console-based rendering for richer visuals.**
3. **Optimization: Optimize the code for better performance, especially in areas such as obstacle rendering and collision detection. Consider data structures and algorithms that can reduce computational overhead and improve responsiveness.**
4. **Input Handling: Implement more robust input handling mechanisms to support a wider range of input devices and improve user experience. Additionally, provide options for customizable controls to accommodate player preferences.**
5. **Scalability: Design the code with scalability in mind to accommodate future features and expansions. Consider implementing features such as additional obstacles, power-ups, or multiplayer support while maintaining code flexibility.**
6. **Error Handling: Implement error handling mechanisms to gracefully handle unexpected situations and provide informative error messages to users. This will improve the overall robustness and user-friendliness of the game.**

**Conclusion:**

In conclusion, the development of this simple 2D game in C++ has provided valuable insights into game development principles and programming concepts. While the current implementation serves as a functional prototype, there are numerous opportunities for improvement and expansion in future iterations.

By addressing the suggested improvements, such as modularization, enhanced graphics, optimization, and scalability, the game can be refined to offer a more polished and engaging experience for players. Additionally, implementing robust input handling, error handling, and thorough documentation will contribute to the overall quality and professionalism of the game.

Overall, this project demonstrates the potential for learning and creativity in game development, and serves as a foundation for further exploration and experimentation in the field. With continued iteration and refinement, the game can evolve into a compelling and enjoyable gaming experience for players of all ages.