Nawal Tariq Warisha Zia

OOP project

23f0776 23f0534

**Cheese Burger and Nyan Cat**

**Code:**

**CheeseBurger.cpp**

#include "CheeseBurger.h"

Cheeseburger::Cheeseburger(int startX, int startY)

: GameObject(startX, startY, 1, 1, "B"),

lives(3), score(0), speed(5), hasShield(false) {}

void Cheeseburger::move() {

if (\_kbhit()) {

char direction = \_getch();

switch (direction) {

case KEY\_LEFT:

x = clamp(x - 1, 0, SCREEN\_WIDTH - 1);

break;

case KEY\_RIGHT:

x = clamp(x + 1, 0, SCREEN\_WIDTH - 1);

break;

}

}

}

void Cheeseburger::draw() {

COORD pos = { static\_cast<SHORT>(x), static\_cast<SHORT>(y) };

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

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

SetConsoleTextAttribute(hConsole, 14); // Yellow for Cheeseburger

std::cout << symbol;

SetConsoleTextAttribute(hConsole, 7); // Reset color to normal

}

void Cheeseburger::increaseScore(int points) {

score += points;

}

void Cheeseburger::update() {

hasShield = false;

speed += 1;

score \*= 1;

}

void Cheeseburger::handleCollision(GameObject\* other) {

if (hasShield) {

std::cout << "Collision avoided with shield!" << std::endl;

}

else {

lives--;

std::cout << "Hit! Lives left: " << lives << std::endl;

if (lives <= 0) {

std::cout << "Game Over!" << std::endl;

}

}

}

void Cheeseburger::activatePowerUp(std::string type) {

if (type == "Shield") {

hasShield = true;

std::cout << "Power-Up Activated: Shield" << std::endl;

}

else if (type == "SpeedBoost") {

speed += 2;

std::cout << "Speed Boost Activated! Cheeseburger speed increased temporarily" << std::endl;

}

else if (type == "ScoreMultiplier") {

increaseScore(15);

std::cout << "Score Multiplier Activated! Points doubled temporarily" << std::endl;

    }

}

**CheeseBurger.h**

#ifndef CHEESEBURGER\_H

#define CHEESEBURGER\_H

#include <iostream>

#include <string>

#include <conio.h>

#include <windows.h>

#include "GameObject.h"

#include "Constants.h"

#define KEY\_LEFT 75

#define KEY\_RIGHT 77

class Cheeseburger : public GameObject {

private:

int lives;

int score;

int speed;

bool hasShield;

public:

Cheeseburger(int startX, int startY);

void move() override;

void draw() override;

void increaseScore(int points);

void update();

void handleCollision(GameObject\* other);

void activatePowerUp(std::string type);

int getLives() const;

int getScore() const;

friend void displayScore(Cheeseburger\* cheeseburger);

friend void resetGame(Cheeseburger\* cheeseburger);

friend class Friend;

friend class Game;

};

#endif // CHEESEBURGER\_H

**Constants.h**

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#define SCREEN\_WIDTH 50

#define SCREEN\_HEIGHT 25

template <typename T>

T clamp(T value, T minVal, T maxVal) {

if (value < minVal) return minVal;

if (value > maxVal) return maxVal;

return value;

}

#endif // CONSTANTS\_H

#pragma once

**Friend.cpp**

#include "Friend.h"

Friend::Friend(int x, int y, std::string name, int supportlevel)

: GameObject(x, y, 2, 2, "F"), name(name), supportLevel(supportlevel), isActive(true), target\_x(x) {}

void Friend::draw() {

if (!isActive) return;

COORD pos = { static\_cast<SHORT>(x), static\_cast<SHORT>(y) };

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

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

SetConsoleTextAttribute(hConsole, 3); // Cyan for Friend

std::cout << symbol;

if (supportLevel == 1) {

std::cout << "F";

}

else if (supportLevel == 2) {

std::cout << "FF";

}

else if (supportLevel >= 3) {

std::cout << "FFF";

}

SetConsoleTextAttribute(hConsole, 7); // Reset color to normal

}

void Friend::move() {

y++;

if (y > SCREEN\_HEIGHT) {

resetPosition();

}

}

void Friend::resetPosition() {

y = 0;

x = rand() % SCREEN\_WIDTH;

}

void Friend::setTargetX(int newTarget) { target\_x = clamp(newTarget, 0, SCREEN\_WIDTH - 3); }

bool Friend::collide(GameObject\* other) {

return isActive && (abs(getX() - other->getX()) <= 1 && getY() == other->getY());

}

void Friend::offerHelp(Cheeseburger& burger) const {

if (!isActive) return;

std::cout << "\t\t\t\t\t\n" << name << " offers level " << supportLevel << " support!\n";

if (supportLevel == 1) {

burger.lives += 1; // Increment lives by 1 if supportLevel is 1

std::cout << "\t\t\t\t\tYou gained 1 life!";

}

else if (supportLevel == 2) {

burger.lives += 2; // Increment lives by 2 if supportLevel is 2

std::cout << "You gained 2 lives!";

}

else if (supportLevel >= 3) {

burger.lives += 3; // Increment lives by 3 for higher support levels

std::cout << "\t\t\t\t\tYou gained 3 lives!";

}

}

void Friend::increasesupportlevel(int increment) {

supportLevel += increment;

}

void Friend::setActive(bool active) { isActive = active; }

**Friend.h**

#ifndef FRIEND\_H

#define FRIEND\_H

#include <iostream>

#include <string>

#include <windows.h>

#include "GameObject.h"

#include "Constants.h"

class Cheeseburger; // Forward declaration

class Friend : public GameObject {

private:

std::string name;

int supportLevel;

bool isActive;

int target\_x;

public:

Friend(int x, int y, std::string name, int supportlevel);

void draw() override;

void move() override;

void resetPosition();

void setTargetX(int newTarget);

bool collide(GameObject\* other);

void offerHelp(Cheeseburger& burger) const;

void increasesupportlevel(int increment = 1);

void setActive(bool active);

};

#endif // FRIEND\_H

**Game.cpp**

#include "Game.h"

Game::Game() : isGameOver(false), gameLevel(1), isPaused(false) {

player = new Cheeseburger(SCREEN\_WIDTH / 2, SCREEN\_HEIGHT - 1);

// Create Nyan Cats

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) {

nyanCats[i] = new NyanCat(rand() % SCREEN\_WIDTH, rand() % 10, rand() % 5);

}

// Create Powerups

for (int i = 0; i < MAX\_POWER\_UPS; ++i) {

powerUps[i] = new Powerup(rand() % SCREEN\_WIDTH, rand() % SCREEN\_HEIGHT, rand() % 3 == 0 ? "Shield" : (rand() % 3 == 1 ? "Speed" : "Score"));

}

// Initialize Friends

for (int i = 0; i < Max\_friend; ++i) {

friendObj[i] = new Friend(rand() % SCREEN\_WIDTH, rand() % SCREEN\_HEIGHT, "wishi", rand() % 3 + 1);

}

}

void Game::draw() {

system("cls");

// Draw game objects

player->draw();

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) {

nyanCats[i]->draw();

}

for (int i = 0; i < MAX\_POWER\_UPS; ++i) {

powerUps[i]->draw();

}

for (int i = 0; i < Max\_friend; ++i) {

friendObj[i]->draw();

}

// Display Score

displayScore(player);

}

void Game::update() {

if (isGameOver) return;

player->update();

player->move();

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) {

nyanCats[i]->move();

}

for (int i = 0; i < MAX\_POWER\_UPS; ++i) {

powerUps[i]->move();

}

for (int i = 0; i < Max\_friend; ++i) {

friendObj[i]->move();

}

checkCollisions();

}

void Game::checkCollisions() {

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) {

if (player->checkCollision(\*nyanCats[i])) {

if (player->getScore() > 0) {

player->handleCollision(nyanCats[i]);

nyanCats[i]->resetPosition();

}

}

}

for (int i = 0; i < MAX\_POWER\_UPS; ++i) {

if (powerUps[i]->collide(player)) {

player->increaseScore(5);

powerUps[i]->activate(\*player);

powerUps[i]->deactivate();

}

}

for (int i = 0; i < Max\_friend; ++i) {

if (friendObj[i]->collide(player)) {

player->increaseScore(10);

friendObj[i]->setActive(true);

friendObj[i]->offerHelp(\*player);

}

}

}

void Game::levelup() {

if (player->getScore() > 100 && player->getScore() < 200) {

gameLevel++;

}

else if (player->getScore() >= 200 && player->getScore() < 300) {

gameLevel++;

}

}

void Game::handleUserInput() {

if (\_kbhit()) {

char ch = \_getch();

if (ch == 27) { // ESC key to pause the game

pauseGame();

}

if (\_getch()) { // any key to resume

resumeGame();

}

}

}

void Game::pauseGame() {

isPaused = true;

}

void Game::resumeGame() {

isPaused = false;

}

bool Game::getPaused() const {

return isPaused;

}

bool Game::getGameOver() const {

return isGameOver;

}

void Game::displayMenu() {

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 10);

std::cout << "\n\n\n\n\n";

std::cout << "\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\t\*\*\*\* Cheeseburger vs. Nyan Cat \*\*\*\*\n";

std::cout << "\t\t\t\t\t\*\n";

// Set text color to yellow for the menu options

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

std::cout << "\t\t\t\t\t1. Start Game\n";

std::cout << "\t\t\t\t\t2. Display Score\n";

std::cout << "\t\t\t\t\t3. Replay game\n";

std::cout << "\t\t\t\t\t4. Game Instructions and rules\n";

std::cout << "\t\t\t\t\t5. Team members\n";

std::cout << "\t\t\t\t\t6. Exit\n";

// Set text color to cyan for the footer and user input prompt

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11);

std::cout << "\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\tEnter your choice: ";

// Reset the text color to default (white)

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

}

void Game::displayScore(Cheeseburger\* cheeseburger) {

COORD pos = { 0, SCREEN\_HEIGHT };

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

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

SetConsoleTextAttribute(hConsole, 11); // Light Blue for Score

std::cout << "Level: " << gameLevel << " Lives: " << cheeseburger->getLives() << " Score: " << cheeseburger->getScore() << std::endl;

SetConsoleTextAttribute(hConsole, 7); // Reset to normal

}

void Game::resetGame(Cheeseburger\* cheeseburger) {

cheeseburger->lives = 3; // Reset lives

cheeseburger->score = 0; // Reset score

cheeseburger->speed = 5; // Reset speed

cheeseburger->hasShield = false;

// Reset NyanCats and Powerups

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) {

nyanCats[i]->resetPosition();

}

for (int i = 0; i < MAX\_POWER\_UPS; ++i) {

powerUps[i]->resetPosition();

powerUps[i]->deactivate();

}

for (int i = 0; i < Max\_friend; ++i) {

friendObj[i]->setActive(false);

}

gameLevel = 1; // Reset game level

isGameOver = false;

std::cout << "Game has been reset. Good luck!" << std::endl;

}

void Game::startGame() {

isGameOver = false;

std::fstream file;

int highestScore = 0;

// Open the file for reading and writing

file.open("highscore.txt", std::ios::in | std::ios::out);

int choice;

do {

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

// Set the background color to red and text color to blue

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 101);

displayMenu();

std::cin >> choice;

switch (choice) {

case 1:

while (!isGameOver) {

if (!isPaused) {

levelup();

if (player->getLives() <= 0) {

isGameOver = false;

}

else {

update();

draw();

checkCollisions();

Sleep(100);

}

// Open the file for reading and writing

file.open("highscore.txt", std::ios::in | std::ios::out);

if (!file) {

// If the file doesn't exist, create it and set highestScore to 0

file.clear(); // Clear any errors

file.open("highscore.txt", std::ios::out); // Create a new file

if (!file) {

std::cout << "Error while creating file!" << std::endl;

return;

}

file.close();

highestScore = 0; // Set initial highest score to 0

}

else {

// If the file exists, read the highest score

file >> highestScore;

file.close();

}

// Compare with the player's score

if (player->getScore() > highestScore) {

highestScore = player->getScore();

// Now save the new highest score to the file

file.open("highscore.txt", std::ios::out | std::ios::trunc); // Open for writing and truncate the file

if (!file) {

std::cout << "Error opening file for writing!" << std::endl;

return;

}

file << highestScore; // Write the new highest score

file.close();

}

}

}

system("cls");

break;

case 2:

if (file) {

file >> highestScore;

std::cout << "Highest Score: " << highestScore << std::endl;

file.close();

system("cls");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 12);

std::cout << "\n\n\n";

std::cout << "\t\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\t\t\* \*\n";

std::cout << "\t\t\t\t\t\t\* HIGHEST SCORE \*\n";

std::cout << "\t\t\t\t\t\t\* \*\n";

std::cout << "\t\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\t\t\* Highest Score: " << highestScore << " \*" << std::endl;

std::cout << "\t\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\t\t Press any key to exit! " << std::endl;

if (\_getch()) {

// Reset the text color to normal (white)

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

system("cls");

break;

}

}

else {

std::cout << "No highscore data found!" << std::endl;

}

break;

case 3:

std::cout << "Replaying game...\n";

resetGame(player);

break;

case 4:

system("cls");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 6);

std::cout << "\n\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\t\t\t\t\t\* GAME INSTRUCTIONS \*\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t|1. Welcome to the Ultimate Game Challenge! |\n";

std::cout << "\t\t\t\t\t| Get ready to test your skills and have fun. |\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\t\t\t\t\t\* GAME RULES \*\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t|1. Each level will increase in difficulty. |\n";

std::cout << "\t\t\t\t\t|2. You earn points by completing challenges. |\n";

std::cout << "\t\t\t\t\t|3. Make sure to beat the timer to advance. |\n";

std::cout << "\t\t\t\t\t|4. If you lose all your lives, the game ends. |\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\t\t\t\t\t\* HOW TO PLAY \*\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t|1. Use arrow keys to navigate. |\n";

std::cout << "\t\t\t\t\t|2. Press 'Enter' to select an option. |\n";

std::cout << "\t\t\t\t\t|3. Collect powerups/friends and avoid nyancats. |\n";

std::cout << "\t\t\t\t\t|4. Press 'ESC' to pause the game anytime. |\n";

std::cout << "\t\t\t\t\t|5. Press 'R' to resume the game. |\n";

std::cout << "\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\t\t\t\t\t\* ENJOY THE GAME! \*\n";

std::cout << "\t\t\t\t\t\*\*\n";

std::cout << "\t\t\t\t\t Press any key to exit! " << std::endl;

if (\_getch()) {

// Reset the text color to normal (white)

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

system("cls");

break;

}

case 5:

system("cls");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11);

std::cout << "\t\t\t\t\t\*\n";

std::cout << "\t\t\t\t\t\* GAME TEAM \*\n";

std::cout << "\t\t\t\t\t\* Pixel Pioneers \*\n";

std::cout << "\t\t\t\t\t\*\n";

// Set text color to yellow for the team member list header

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

std::cout << "\t\t\t\t\t\* Member Name | Roll Number \*\n";

std::cout << "\t\t\t\t\t\*\n";

// Set text color to green for team member details

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

std::cout << "\t\t\t\t\t\* Warisha Zia | 23F-0534 \*\n";

std::cout << "\t\t\t\t\t\* Nawal Tariq | 23F-0776 \*\n";

std::cout << "\t\t\t\t\t\*\n";

// Set text color to magenta for the closing line

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 13);

std::cout << "\t\t\t\t\t Let the Games Begin! \n";

std::cout << "\t\t\t\t\t Press any key to exit!" << std::endl;

if (\_getch()) {

// Reset the text color to normal (white)

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

system("cls");

break;

}

case 6:

std::cout << "Exiting Game\n";

break;

default:

std::cout << "Invalid choice. Please try again.\n";

}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

} while (choice != 6);

}

Game::~Game() {

delete player;

for (int i = 0; i < MAX\_NYAN\_CATS; ++i) delete nyanCats[i];

for (int i = 0; i < MAX\_POWER\_UPS; ++i) delete powerUps[i];

for (int i = 0; i < Max\_friend; ++i) delete friendObj[i];

}

**Game.h**

#ifndef GAME\_H

#define GAME\_H

#include <iostream>

#include <string>

#include <windows.h>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <conio.h>

#include <algorithm>

#include <fstream>

#include "CheeseBurger.h"

#include "NyanCat.h"

#include "Powerup.h"

#include "Friend.h"

#define SCREEN\_WIDTH 50

#define SCREEN\_HEIGHT 25

#define MAX\_NYAN\_CATS 5

#define MAX\_POWER\_UPS 3

#define Max\_friend 2

#define KEY\_LEFT 75

#define KEY\_RIGHT 77

class Game {

private:

Cheeseburger\* player;

NyanCat\* nyanCats[MAX\_NYAN\_CATS];

Powerup\* powerUps[MAX\_POWER\_UPS];

Friend\* friendObj[Max\_friend];

int gameLevel;

bool isGameOver;

bool isPaused;

public:

Game();

void draw();

void update();

void checkCollisions();

void levelup();

void handleUserInput();

void pauseGame();

void resumeGame();

bool getPaused() const;

bool getGameOver() const;

void displayMenu();

void displayScore(Cheeseburger\* cheeseburger);

void resetGame(Cheeseburger\* cheeseburger);

void startGame();

~Game();

};

#endif // GAME\_H

**GameObject.cpp**

#include "GameObject.h"

GameObject::GameObject(int x, int y, int width, int height, std::string Symbol)

: x(x), y(y), width(width), height(height), symbol(Symbol) {}

int GameObject::getX() const { return x; }

int GameObject::getY() const { return y; }

int GameObject::getWidth() const { return width; }

int GameObject::getHeight() const { return height; }

bool GameObject::checkCollision(GameObject& other) const {

return (x < other.x + other.width &&

x + width > other.x &&

y < other.y + other.height &&

y + height > other.y);

}

**GameObject.h**

#ifndef GAMEOBJECT\_H

#define GAMEOBJECT\_H

#include <iostream>

#include <string>

class GameObject {

protected:

int x, y;

int width, height;

std::string symbol;

public:

GameObject(int x, int y, int width, int height, std::string Symbol);

virtual void move() = 0;

virtual void draw() = 0;

int getX() const;

int getY() const;

int getWidth() const;

int getHeight() const;

bool checkCollision(GameObject& other) const;

};

#endif // GAMEOBJECT\_H

**NyanCat.cpp**

#include "NyanCat.h"

NyanCat::NyanCat(int x, int y, int level)

: GameObject(x, y, 1, 1, "N"), level(level), isVisible(true) {

fallSpeed = 1.0f + level \* 0.5f;

}

void NyanCat::move() {

y += static\_cast<int>(fallSpeed);

if (y > SCREEN\_HEIGHT) {

resetPosition();

}

}

void NyanCat::draw() {

if (isVisible) {

COORD pos = { static\_cast<SHORT>(x), static\_cast<SHORT>(y) };

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

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

SetConsoleTextAttribute(hConsole, 5); // Magenta for NyanCat

std::cout << symbol;

SetConsoleTextAttribute(hConsole, 7); // Reset color to normal

}

}

void NyanCat::resetPosition() {

y = 0;

x = rand() % SCREEN\_WIDTH;

isVisible = (rand() % 100) > (level \* 10);

}

**NyanCat.h**

#ifndef NYANCAT\_H

#define NYANCAT\_H

#include <iostream>

#include <string>

#include <windows.h>

#include "GameObject.h"

#include "Constants.h"

class NyanCat : public GameObject {

private:

int level;

float fallSpeed;

bool isVisible;

public:

NyanCat(int x, int y, int level);

void move() override;

void draw() override;

void resetPosition();

int getLevel() const;

friend class Powerup;

};

#endif // NYANCAT\_H

**Powerup.cpp**

#include "Powerup.h"

Powerup::Powerup(int X, int Y, std::string T)

: GameObject(X, Y, 1, 1, "G"), type(T), isActive(true) {}

void Powerup::draw() {

if (!isActive) return;

COORD pos = { static\_cast<SHORT>(x), static\_cast<SHORT>(y) };

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

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

SetConsoleTextAttribute(hConsole, 10);

std::cout << (type == "Shield" ? "s" : (type == "Speed" ? "S" : "/"));

SetConsoleTextAttribute(hConsole, 7);

}

void Powerup::move() {

y++;

if (y > SCREEN\_HEIGHT) {

resetPosition();

}

}

bool Powerup::collide(GameObject\* object) {

return isActive && (getX() == object->getX() && getY() == object->getY());

}

void Powerup::activate(Cheeseburger& cheeseburger) const {

cheeseburger.activatePowerUp(type);

}

void Powerup::resetPosition() {

y = 0;

x = rand() % SCREEN\_WIDTH;

}

void Powerup::deactivate() { isActive = false; }

**Powerup.h**

#ifndef POWERUP\_H

#define POWERUP\_H

#include <iostream>

#include <string>

#include <windows.h>

#include "GameObject.h"

#include "Constants.h"

class Cheeseburger; // Forward declaration

class Powerup : public GameObject {

private:

std::string type;

bool isActive;

public:

Powerup(int X, int Y, std::string T);

void draw() override;

void move() override;

bool collide(GameObject\* object);

void activate(Cheeseburger& cheeseburger) const;

void resetPosition();

void deactivate();

bool isActivated() const;

std::string getType() const;

};

#endif // POWERUP\_H

**main.cpp**

#include "Game.h"

int main() {

srand(time(NULL)); // Seed for random number generation

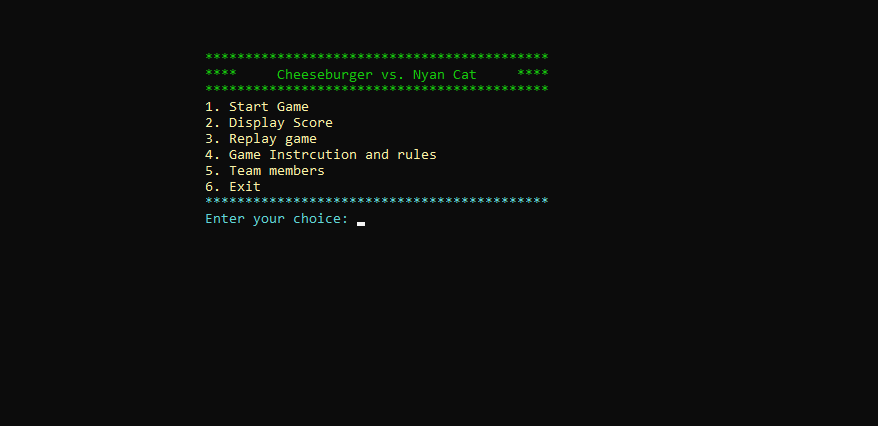
Game game;

game.startGame();

system("pause");

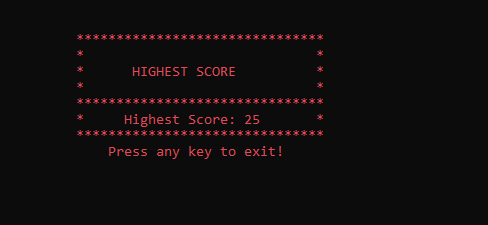
    return 0;

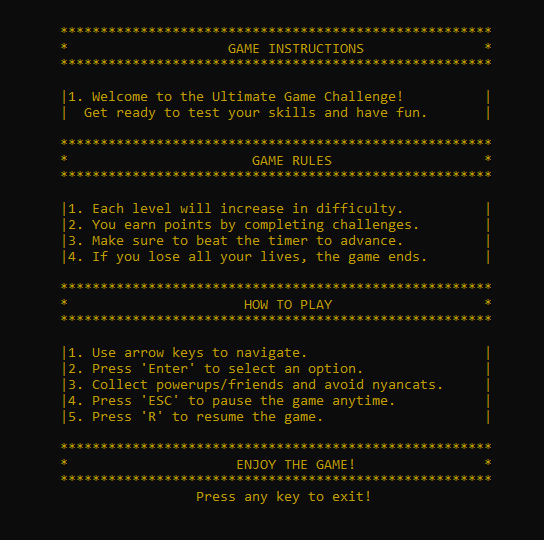
}

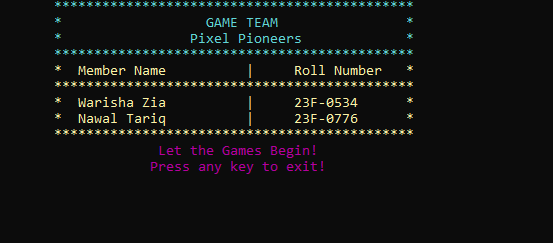












**Summary:**

**Cheeseburger vs. Nyan Cat Game**

This is a C++ console-based game where the player, represented by a *Cheeseburger*, battles falling *Nyan Cats* while collecting *Power-ups* and receiving help from *Friends*. The game emphasizes object-oriented programming principles like inheritance, polymorphism, and encapsulation.

**Core Features:**

1. **Player (Cheeseburger)**:
   * Moves left and right to avoid collisions with Nyan Cats.
   * Gains points by collecting power-ups and interacting with friends.
   * Has abilities like activating shields, boosting speed, and doubling score through power-ups.
   * Loses lives on collisions with Nyan Cats unless shielded.
2. **Nyan Cats**:
   * Fall from random positions at varying speeds based on difficulty.
   * Reset to the top of the screen upon collision or leaving the screen.
   * Create a dynamic challenge for the player as levels progress.
3. **Power-ups**:
   * Spawn randomly and provide special boosts such as:
     + **Shield**: Prevents damage from collisions temporarily.
     + **Speed Boost**: Increases player movement speed.
     + **Score Multiplier**: Increases points earned.
4. **Friends**:
   * Friendly game objects that provide help like increasing player lives when interacted with.
   * Have varying levels of support, affecting the amount of help offered.
5. **Levels and Difficulty**:
   * Game difficulty increases as the player earns more points, introducing faster Nyan Cats and challenging dynamics.
   * Levels increment based on score milestones (e.g., 100 points).
6. **Game Mechanics**:
   * The game employs **collision detection** to manage interactions between game objects.
   * A **pause/resume** functionality is available using specific keys.
   * A simple **menu system** provides options like starting the game, displaying scores, instructions, and team credits.
7. **Graphics**:
   * Uses ASCII symbols to represent game elements (e.g., "B" for Cheeseburger, "N" for Nyan Cats, "F" for Friends).
   * Utilizes **color-coded text** to differentiate objects and enhance visual clarity.
8. **Game State Management**:
   * Keeps track of player lives, score, and current level.
   * Resets game state for replay after game over.

**Key Classes:**

1. **GameObject** (Base Class): Abstract class for all game entities, implementing basic attributes (position, size) and collision detection.
2. **Cheeseburger** (Player): Controls player actions and manages score, lives, and power-ups.
3. **NyanCat**: Enemy objects that fall from the top of the screen with varying speed.
4. **Powerup**: Bonus objects providing temporary boosts when collected.
5. **Friend**: Supportive objects that increase player lives based on their support level.
6. **Game**: Manages overall gameplay, rendering, user input, and game logic.

**Game Flow:**

* The player moves the Cheeseburger left or right to avoid falling Nyan Cats and collect power-ups.
* Lives are lost on collision with Nyan Cats, but power-ups and help from Friends can mitigate damage or restore lives.
* The game ends when the player runs out of lives, but it can be replayed with a reset state.