Final Project: "Mini Asteroids" Video Game

Gloria Yael Martínez Zuñiga(2024)

Daniel Alexandro Arenas Pérez(2024)

Universidad Pólitecnica de San Luis Potosí

T07D: Programming I

Instructor: Gerardo Franco Delgado

29/05/2024

Table of Contents:

MINI ASTEROIDS

Introduction

The video game is based on a popular vector-based arcade game released in 1979 by Atari. The objective of the game is to shoot and destroy asteroids while avoiding collision with the fragments that break off from them.

Objective

The objective of the project was to create a video game where we would implement the material covered in class, as well as use external resources to complement our code. This involved using classes, functions, lists, and other programming concepts.

Methodology

1. Pre-production

Idea and Concept

Brainstorming: Generate ideas and define the game concept.

2. Production

Programming: Develop the game code Internal Testing

Debugging: Identify and solve bugs and technical issues.

3. Pre-delivery

Adjustments and Improvements: Make adjustments based on the feedback received.

4. Final Delivery

Publishing on Platforms: Repository on GitHub and documentation of the code

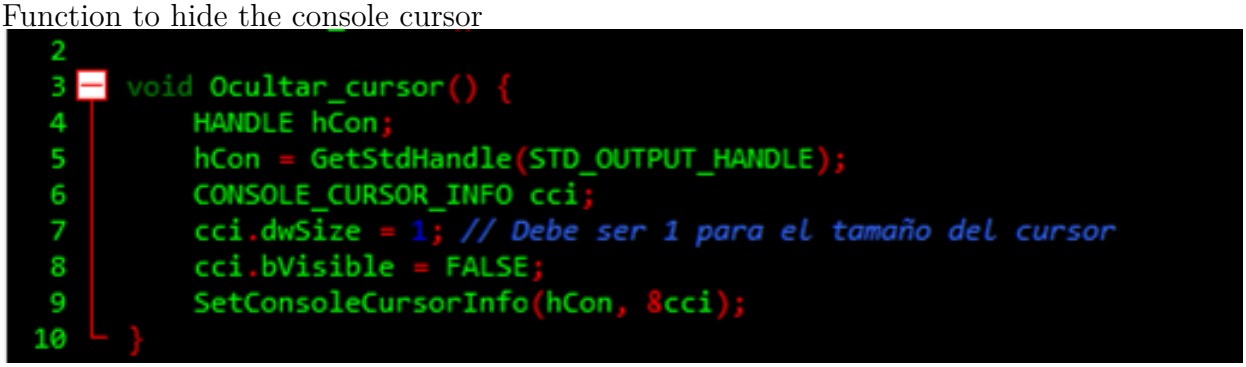
Code Explanation

The code presents a simple game in C++ that runs in the console. In this game, the player controls a ship that moves across the screen and shoots bullets to destroy asteroids while dodging collisions with them. Below is an explanation of each part of the code in a basic manner:

Function

Hide Cursor Function

Function to hide the console cursor



```
2
3   void Ocultar_cursor() {
4       HANDLE hCon;
5       hCon = GetStdHandle(STD_OUTPUT_HANDLE);
6       CONSOLE_CURSOR_INFO cci;
7       cci.dwSize = 1; // Debe ser 1 para el tamaño del cursor
8       cci.bVisible = FALSE;
9       SetConsoleCursorInfo(hCon, &cci);
10  }
```

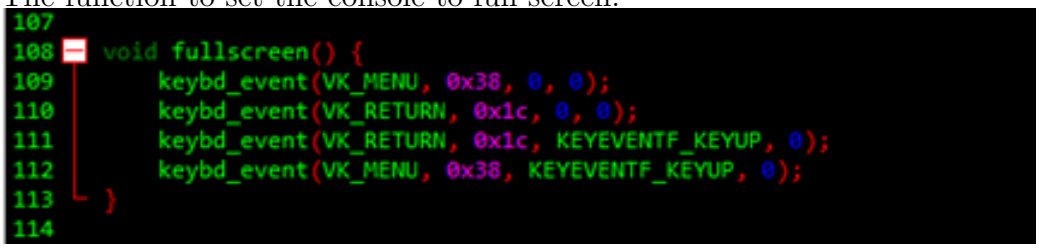
Function to paint boundaries

Function to draw the borders of the game area in the console using ASCII characters

```
11
12  void pintar_limites() {
13      for (int i = 2; i < 130; i++) {
14          gotoxy(i, 3); printf("%c", 205);
15          gotoxy(i, 35); printf("%c", 205);
16      }
17
18      for (int i = 4; i < 35; i++) {
19          gotoxy(2, i); printf("%c", 186);
20          gotoxy(130, i); printf("%c", 186);
21      }
22
23      gotoxy(2, 3); printf("%c", 201);
24      gotoxy(2, 35); printf("%c", 200);
25      gotoxy(130, 3); printf("%c", 187);
26      gotoxy(130, 35); printf("%c", 188);
27  }
28
```

Fullscreen Function

The function to set the console to full screen.



```
107
108  void fullscreen() {
109      keybd_event(VK_MENU, 0x38, 0, 0);
110      keybd_event(VK_RETURN, 0x1c, 0, 0);
111      keybd_event(VK_RETURN, 0x1c, KEYEVENTF_KEYUP, 0);
112      keybd_event(VK_MENU, 0x38, KEYEVENTF_KEYUP, 0);
113  }
114
```

mainJuego Function

The main function containing the game loop.

Classes

It is used to organize and structure the code more efficiently, allowing the reuse and abstraction of common data and behaviors. Classes are fundamental in object-oriented programming, where each object created from a class has its own characteristics and can perform specific actions defined in the class.

The Ship Class

Defines the player-controlled ship.

SHIP(int x, int y, int hearts, int lives): Constructor of the SHIP class that initializes the coordinates (x, y) of the ship, the number of hearts, and lives.

int X() return x; and int Y() return y; : Methods that return the coordinates (x, y) of the ship.

int LIVES() return lives; : Method that returns the remaining number of lives of the ship.

void HEARTS() hearts–; : Method that decrements the number of hearts of the ship by one.

void draw(): Method that draws the ship on the screen using ASCII characters.

void erase(): Method that erases the ship from the screen.

```
28
29  class NAVE {
30      int x, y;
31      int corazones;
32      int vidas;
33  public:
34      NAVE(int _x, int _y, int _corazones, int _vidas);
35      int X() { return x; }
36      int Y() { return y; }
37      int VIDAS() { return vidas; }
38      void COR() { corazones--; }
39      void pintar();
40      void borrar();
41      void mover();
42      void pintar_corazones();
43      void morir();
44  };
45
46  NAVE::NAVE(int _x, int _y, int _corazones, int _vidas)
47      : x(_x), y(_y), corazones(_corazones), vidas(_vidas) {}
48
49  void NAVE::pintar() {
50      gotoxy(x, y); printf(" %c%c%c", 201, 205, 187);
51      gotoxy(x, y + 1); printf("%c%c %c%c", 201, 188, 200, 187);
52      gotoxy(x, y + 2); printf("%c%c%c%c%c", 200, 205, 205, 205, 188);
53      gotoxy(x, y + 3); printf("     ");
54  }
55
56  void NAVE::borrar() {
57      gotoxy(x, y); printf("       ");
58      gotoxy(x, y + 1); printf("       ");
59      gotoxy(x, y + 2); printf("       ");
60  }
61
```

void move(): Method that allows the ship to move on the screen according to the keys pressed by the player.

void draw hearts(): Method that displays on the screen the number of lives and remaining hearts of the ship.

```
void NAVE::mover() {
    if (_kbhit()) {
        char tecla = _getch();
        borrar();
        if (tecla == 'a' && x > 3) x--;
        if (tecla == 'd' && x + 5 < 130) x++;
        if (tecla == 'w' && y > 4) y--;
        if (tecla == 's' && y + 3 < 35) y++;
        if (tecla == 'e') corazones--;
        pintar();
        pintar_corazones();
    }
}

void NAVE::pintar_corazones() {
    gotoxy(70, 2); printf("Vidas %d", vidas);
    gotoxy(90, 2); printf("HP");
    gotoxy(95, 2); printf("       ");

    for (int i = 0; i < corazones; i++) {
        gotoxy(95 + i, 2); printf("%c", 254);
    }
}
```

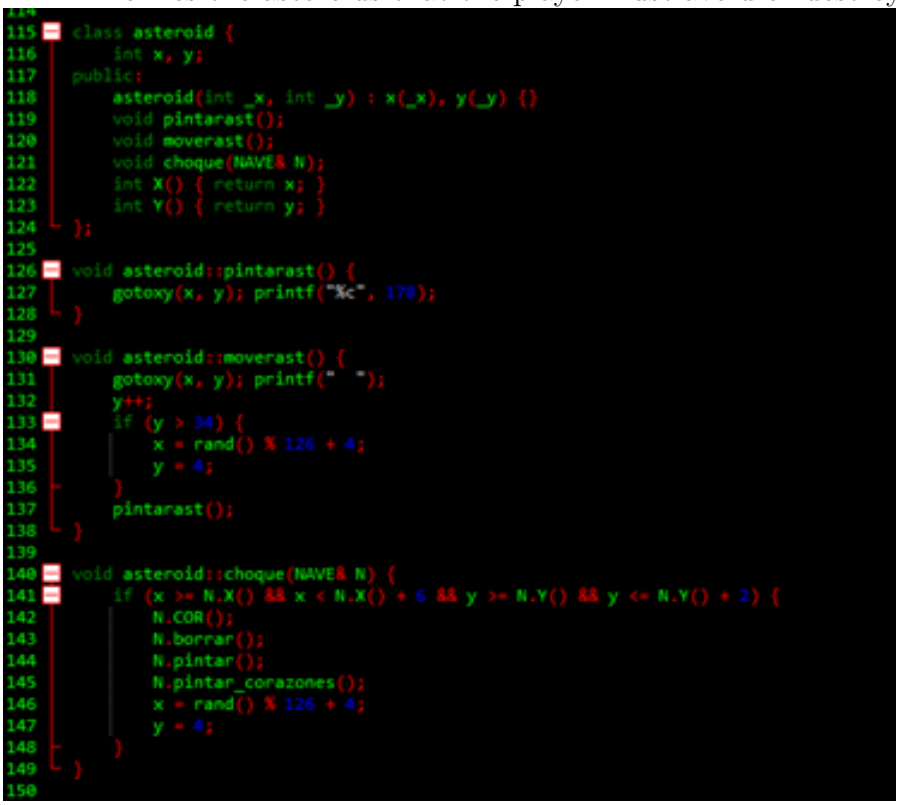
void die(): Method that handles the ship's death, decreasing lives, displaying an animation, and resetting the hearts.

```
void NAVE::morir() {
    if (corazones == 0) {
        borrar();
        gotoxy(x, y); printf("     ");
        gotoxy(x, y + 1); printf(" *** ");
        gotoxy(x, y + 2); printf(" *** ");
        Sleep(300);

        borrar();
        gotoxy(x, y); printf(" *** ");
        gotoxy(x, y + 1); printf("*   *");
        gotoxy(x, y + 2); printf("*   *");
        gotoxy(x, y + 3); printf(" *** ");
        Sleep(300);
        borrar();
        vidas--;
        corazones = 3;
        pintar_corazones();
        pintar();
    }
}
```

The Asteroid Class

Defines the asteroids that the player must avoid or destroy.



```
class asteroid {
    int x, y;
public:
    asteroid(int _x, int _y) : x(_x), y(_y) {}
    void pintarast();
    void moverast();
    void choque(NAVE& N);
    int X() { return x; }
    int Y() { return y; }
};

void asteroid::pintarast() {
    gotoxy(x, y); printf("%c", 178);
}

void asteroid::moverast() {
    gotoxy(x, y); printf(" ");
    y++;
    if (y > 34) {
        x = rand() % 126 + 4;
        y = 4;
    }
    pintarast();
}

void asteroid::choque(NAVE& N) {
    if (x >= N.X() && x < N.X() + 6 && y >= N.Y() && y <= N.Y() + 2) {
        N.COR();
        N.borrar();
        N.pintar();
        N.pintar_corazones();
        x = rand() % 126 + 4;
        y = 4;
    }
}
```

This class is called .Asteroid.and models an asteroid in a game. It has two instance variables for the x and y coordinates of the asteroid. The constructor initializes these coordinates.

It has three member functions:

"drawAsteroid()"which draws the asteroid on the screen.

"moveAsteroid()"which moves the asteroid downward on the screen and repositions it at the top when it reaches the bottom edge.

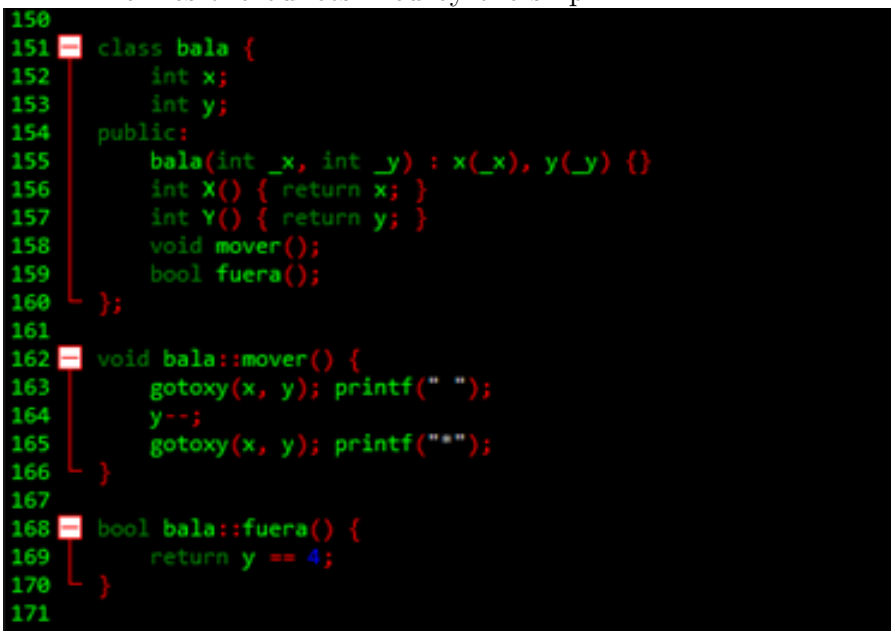
çollision(Ship ship)"which checks for collision between the asteroid and a ship. If there is a collision, the ship loses a life, gets erased from the screen, and repositions while the asteroid also repositions.

The "drawAsteroid().and "moveAsteroid()"functions use an external function "gotoxy()"to handle the position on the screen and "printf()"to print characters.

The çollision(Ship ship)"function uses methods of the "Shipçlass to handle collision and the ship's state.

The Bullet Class

Defines the bullets fired by the ship.



```
150
151 class bala {
152     int x;
153     int y;
154 public:
155     bala(int _x, int _y) : x(_x), y(_y) {}
156     int X() { return x; }
157     int Y() { return y; }
158     void mover();
159     bool fuera();
160 };
161
162 void bala::mover() {
163     gotoxy(x, y); printf(" ");
164     y--;
165     gotoxy(x, y); printf("*");
166 }
167
168 bool bala::fuera() {
169     return y == 4;
170 }
171
```

This class defines the bullets fired by the ship. It has two member functions:

"move()"which moves the bullet upwards on the screen and updates its position. This function uses an external function

"gotoxy()"to handle the position on the screen and "printf()"to print characters. It first erases the current position of the bullet on the screen and then moves it upwards, representing it with an asterisk "*".

.ºut()"which checks if the bullet has gone off the screen, returning true if the y-coordinate is equal to 4, indicating it has reached the top of the screen.

Headers

Headers, also known as header files, are files containing declarations of functions, classes, variables, and other elements used in a program. These files typically have the extension ".hïn C and C++, and they are used to provide information about the interfaces of modules or libraries used in a program. Headers allow for separating declarations from implementations, facilitating code organization and component reuse in different parts of the program.

Conclusion

In conclusion, the code provides the foundation for a simple game in C++ where the player controls a spaceship. The classes are organized such that the ship (SHIP) has functions for drawing, erasing, moving, managing its lives, and displaying its state on the screen. Additionally, there are classes for asteroids and bullets, which have functions for their movement and collision detection. Main functions like mainGame() control the flow of the game, managing user interaction, screen drawing, and game logic.

In addition, we implemented headers in our code as this helps to organize the code in a cleaner and more modular way. It separates the declaration from the implementations and avoids code repetition. Additionally, it facilitates code maintainability, since if there is a need to modify the signature of a function, it can be done solely in the header file, and it will automatically affect all files that include it.