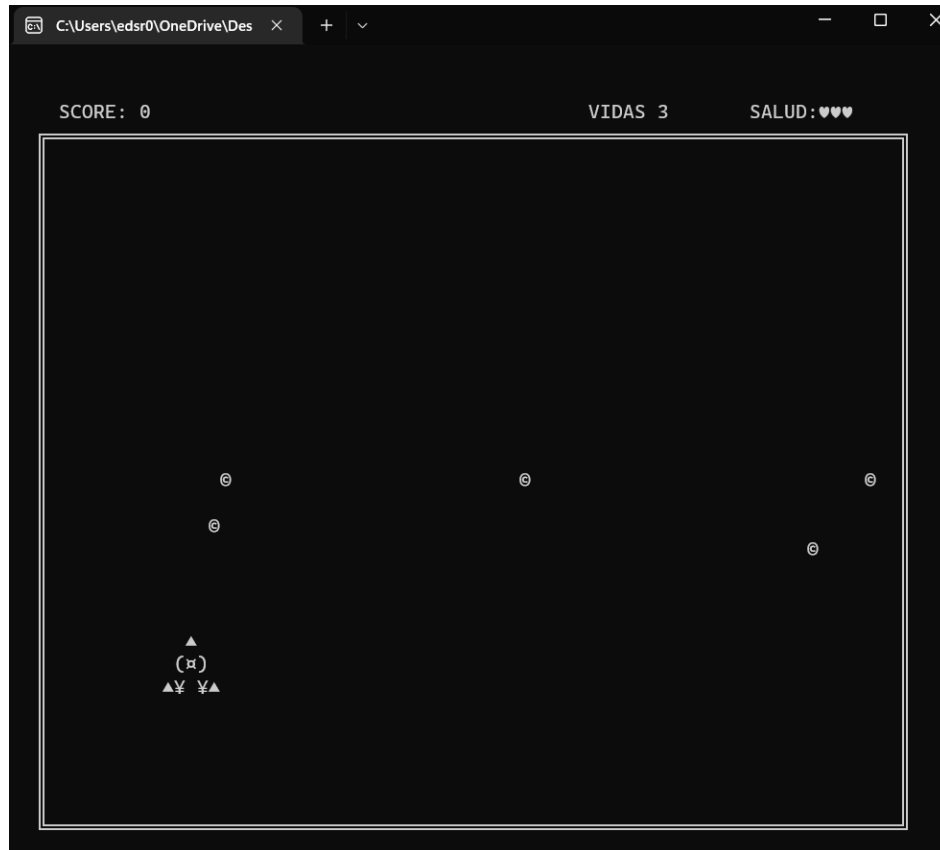


Proyecto Juego de Naves en C++.

En este proyecto creamos con las herramientas de la programación un juego inspirado en el clásico "galaga", que con el código ASCII en algunas partes del programa las cuales son las teclas los definimos con un define con el mismo código ASCII del movimiento de la nave la cual en el programa lo hicimos con la programación orientada a objetos la cual nos sirvió para que que le diéramos un movimiento conforme lo hacíamos para que tuviera mas cosas como las vidas y el puntaje, también la función gotoxy nos sirvió mucho o casi era lo fundamental del programa que hacia que se ubicara la nave.

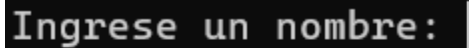


Para poder acomodar símbolos o letras en la posición que nosotros queremos, utilizamos la función gotoxy de Windows, la cual con dos valores enteros (X y Y) te posiciona el símbolo en la coordenada deseada.

```
//Obtiene el identificador de la ventana y las coordenadas
void gotoxy(int x, int y) {
    HANDLE hCon;
    hCon = GetStdHandle(STD_OUTPUT_HANDLE);

    COORD dwPos;
    dwPos.X = x;
    dwPos.Y = y;
    SetConsoleCursorPosition(hCon, dwPos);
}
```

Lo siguiente fue utilizar una función para ocultar el cursor y no este molestando en la pantalla.



Ingrese un nombre: |

```
//Oculta el cursor en la pantalla para que se vea más bonito
void OcultarCursor() {

    HANDLE hCon;
    hCon = GetStdHandle(STD_OUTPUT_HANDLE);

    CONSOLE_CURSOR_INFO cci;
    cci.dwSize = 2;
    cci.bVisible = FALSE;

    SetConsoleCursorInfo(hCon, &cci);

}
```

La siguiente función que hicimos fue para pintar los límites del juego en el tamaño de la ventana, se utilizó la función gotoxy y dos bucles for para pintar las 4 líneas de los límites.

```
//Pinta los limites del juego
void pintar_limites() {

    for (int i = 2; i < 78; i++) {
        gotoxy(i, 3); printf("%c", 205);
        gotoxy(i, 33); printf("%c", 205);
    }
    for (int i = 4; i < 33; i++) {
        gotoxy(2, i); printf("%c", 186);
        gotoxy(77, i); printf("%c", 186);
    }
    gotoxy(2, 3); printf("%c", 201);
    gotoxy(2,33); printf("%c", 200);
    gotoxy(77,3); printf("%c", 187);
    gotoxy(77,33); printf("%c", 188);

}
```



Objeto nave.

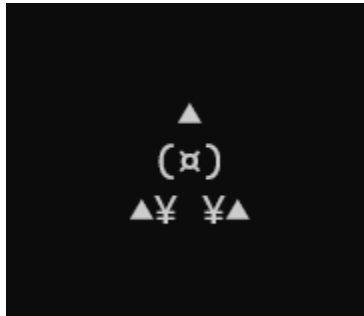
Para crear la nave se hizo un objeto que tienen el nombre de “NAVE” en el cual se utilizaron los atributos x,y para la función gotoxy, la variable vida y corazones.

```
//Clase NAVE con atributos y métodos
class NAVE{
    int x, y;
    int corazones;
    int vidas;
public:
    NAVE(int _x, int _y, int _corazones, int _vidas) :x(_x), y(_y), corazones(_corazones), vidas(_vidas){}
    int X(){ return x; }
    int Y() { return y; }
    int vida(){return vidas;}
    void cor() { corazones--; };
    void pintar();
    void borrar();
    void mover();
    void dibujar_corazones();
    void morir();
};
```

El método int x devuelve los valores de x de la nave al igual que el método. El método vida devuelve actual del jugador al igual que el método corazones.

El método pintar tiene como función pintar la nave en una posición determinada, para eso utilizamos la función gotoxy y utilizando símbolos del Código ASCII.

```
//Definiciones de los métodos
//Dibuja la nave en la consola
void NAVE::pintar() {
    gotoxy(x, y);    printf("  %c", 30);
    gotoxy(x, y + 1); printf(" %c%c%c", 40, 207, 41);
    gotoxy(x, y + 2); printf("%c%c %c%c", 30, 190, 190, 30);
}
```



El método borrar de la nave tiene como función borrar el rastro de la nave con espacios de texto vacíos.

```
//Borra el rastro de la nave
void NAVE::borrar() {
    gotoxy(x, y);    printf("    ");
    gotoxy(x, y+1);  printf("    ");
    gotoxy(x, y+2);  printf("    ");
}
```

El método mover tiene la función de detectar si se presiono una tecla y comparar cuál tecla se presionó para sumar-restar a X o Y, y hacer que la nave se pinte en la nueva posición y se borre en la posición pasada.

```
//Identifica la tecla presionada para mover la nave
void NAVE::mover() {
    if (_kbhit()) {
        char tecla = _getch();
        borrar();
        if (tecla == IZQUIERDA && x - 1 > 3) {
            x--;
        }
        if (tecla == DERECHA && x + 6 < 77) {
            x++;
        }
        if (tecla == ARRIBA && y > 4) {
            y--;
        }
        if (tecla == ABAJO && y + 3 < 33) {
            y++;
        }

        pintar();
        dibujar_corazones();
    }
}
```

El método dibujar corazones tiene como funcionar detectar los corazones de la nave con un bucle for e imprimirlos en pantalla, también se imprime la vida en el mismo método.

```
//Dibuja los corazones en pantalla
void NAVE::dibujar_corazones() {

    gotoxy(50, 2); printf("VIDAS %d", vidas);
    gotoxy(64, 2); printf("SALUD:");
    gotoxy(70, 2); printf("    ");
    for (int i = 0; i < corazones; i++) {
        gotoxy(70 + i, 2); printf("%c", 3);
    }
}
```

VIDAS 3

SALUD:♥♥♥

El método morir de la nave tiene la función de aplicar una animación en las coordenadas de la nave para hacer un efecto de explosión, aparte reinicia los corazones y se le disminuye uno a la vida.

```
//Se le baja la vida a la nave y aplica la animación de morir
void NAVE::morir() {

    if (corazones == 0) {
        borrar();
        gotoxy(x, y); printf(" ** ");
        gotoxy(x, y+1); printf(" **** ");
        gotoxy(x, y+2); printf(" ** ");
        Sleep(400);

        borrar();
        gotoxy(x, y); printf("** ** ");
        gotoxy(x, y + 1); printf(" **** ");
        gotoxy(x, y + 2); printf("** ** ");
        Sleep(800);
        borrar();
        vidas--;
        corazones = 3;
        dibujar_corazones();
        pintar();
    }
}
```

Objeto asteroide.

Para el objeto asteroide se utilizaron como atributos X y Y para la función gotoxy.

```
//Objeto asteroide
class AST {
    int x, y;
public:
    AST(int _x, int _y) :x(_x), y(_y) {}
    int X(){return x;}
    int Y(){return y;}
    void pintarast();
    void moverast();
    void colision(class NAVE &N);
};
```

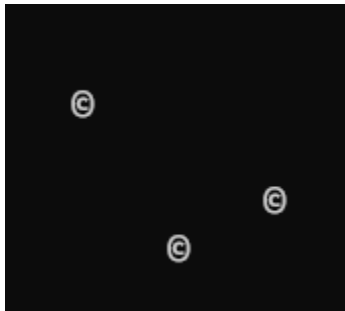
Se utilizaron dos métodos que devuelven los valores de X y Y que devuelven sus coordenadas X y Y respectivamente.

El método "pintarast" imprime en pantalla el símbolo 184 del código ASCII con ayuda de la función gotoxy.

```
//Pinta el asteroide
void AST::pintarast() {

    gotoxy(x, y); printf("%c", 184);

}
```



El método "moverast" imprime un espacio vacío en la posición actual para después sumar 1 a Y, después se usa un condicional para saber si el asteroide se encuentra dentro de los límites, de ser así se crea una posición aleatoria en X y se iguala Y a 4 (que es la altura a la que comienzan a caer los asteroides) y por último se pinta el asteroide con las nuevas coordenadas.

```
//Mueve el asteroide en línea recta hacia abajo
void AST::moverast() {

    gotoxy(x, y); printf(" ");
    y++;
    if (y > 32) {
        x = rand() % 71 + 4;
        y = 4;
    }
    pintarast();

}
```

El método colisión detecta si el asteroide está en las mismas coordenadas X y Y de la nave, y de ser así le quita un corazón, borra la nave, se vuelve a pintar la nave y los corazones, se genera otra coordenada X aleatoria y Y se iguala a 4 para volver a posicionar al asteroide.

```
//Detecta si hay colisiones con la nave
void AST::colision(class NAVE &N) {
    if (x >= N.X() && x < N.X()+6 && y >= N.Y() && y <= N.Y()+2) {
        N.cor();
        N.borrar();
        N.pintar();
        N.dibujar_corazones();
        x = rand() % 71 + 4;
        y = 4;
    }
}
```

Objeto bala.

Nuevamente este objeto utiliza los atributos X y Y para la función gotoxy. También se utilizan los métodos X y Y para devolver los valores de sus coordenadas.

```
//Objeto bala
class bala {
    int x, y;
public:
    bala(int _x, int _y) : x(_x), y(_y){}
    int X() { return x; }
    int Y() { return y; }
    void mover();
    bool fuera();
};
```

El método mover imprime borra la bala y la imprime de nuevo con un bucle que disminuye Y en 1 para hacer el efecto de que la bala sube en línea recta.

```
//Mueve la bala en línea recta hacia arriba
void bala::mover() {
    gotoxy(x, y); printf(" ");
    if (y > 4) {
        y--;
    }
    gotoxy(x, y); printf("*");
}
```



El método fuera es una función booleana que devuelve true si la bala llegó al límite o false en el caso contrario.

```
//Detecta si la bala está afuera
bool bala::fuera() {
    if (y == 4) {
        return true;
    }
    else {
        return false;
    }
}
```

Función main.

Para empezar primera se utilizan las funciones ocultar_cursor y pintar límites para que la ventana se vea limpia. Después se crea un objeto Nave que se llama N y se inicializa en las coordenadas (7,25), se le ponen 3 vidas y 3 corazones, después se llama el método pintar de la nave y el método dibujar_corazones, además se inicializa el puntaje en 0. Se crea una lista para los asteroides junto con su iterador, luego con un bucle for se crean 5 asteroides. También se crea una lista para las balas junto con su iterador y la variable booleana gameover se inicializa como false.

```
//Función main
int main()
{

    OcultarCursor();

    pintar_limites();

    //Se crea el objeto nave
    NAVE N(7, 25, 3, 3);
    //Se llama el objeto N con su método pintar
    N.pintar();
    //Se llama el objeto N con su método dibujar_corazones
    N.dibujar_corazones();
    //Se inicializa el puntaje en 0
    int score = 0;

    //Se crea una lista de asteroides
    list<AST*> A;

    //Se crea una iterador para la lista de asteroides
    list<AST*>::iterator itA;
    //Se rellena la lista de asteroides en posiciones aleatorias en la pantalla
    for(int i = 0; i<5; i++){
        A.push_back(new AST(rand()%75+3, rand()%4+4));
    }
    //Se crea una lista de balas
    list<bala*> B;
    //Se crea un iterador para la lista de las balas
    list<bala*>::iterator it;

    //Se inicializa el gameover en falso
    bool gameover = false;
```

Con un bucle while que se cumple siempre y cuando gameover sea falso se corre el juego. Primero se imprime el score, después con un condicional se verifica si las vidas han llegado a 0, de ser el caso gameover se vuelve verdadero, se sale del bucle y se acaba el juego.

También se utiliza otro condicional para detectar si se presionó la tecla “z” (que es la tecla con la que se disparan las balas), de ser cierto se crea una bala en la lista con posiciones X y Y para que aparezca delante de la nave.

Con un bucle for se le aplica a cada bala su método mover y con un condicional se detecta si la nave esta afuera con el método que lleva el mismo nombre, si es cierto se elimina la nave.


```

//Bucle while que se cumple mientras todavía no sea game over
while (gameover != true) {

    //Se imprime el puntaje
    gotoxy(4,2); printf("SCORE: %d", score);

    //Detecta si las vidas llegan a 0 y si es el caso gameover
    if(N.vida() == 0){
        gameover = true;
    }

    //Detecta si se pulsa la tecla z
    if (_kbhit()) {
        char tecla = _getch();
        //Se crean balas en la lista
        if (tecla == 'z') {
            B.push_back(new bala(N.X() + 2, N.Y() - 1));
        }
    }

    //Se les aplica el método mover a las balas y se eliminan si llegan al límite
    for (it = B.begin(); it != B.end(); it++) {
        (*it)->mover();
        if ((*it)->fuera()) {
            gotoxy((*it)->X(), (*it)->Y()); printf(" ");
            delete(*it);
            it = B.erase(it);
        }
    }
}

```

Con un bucle for se le aplican los métodos mover y colisión a los asteroides de la lista. Después con otro bucle for que recorre los elementos de la lista de los asteroides y con otro bucle for que recorre los elementos de la lista de las balas se detecta si hay colisión entre el asteroide y la bala, de ser cierto se elimina el asteroide, la bala y se suman 5 puntos.

Para finalizar se aplican el método morir y mover a la nave, además de un Sleep de 40 milisegundos para que se reduzca la velocidad del juego y sea más fácil jugarlo.

```

//Se les aplica el método mover a las balas y se eliminan si llegan al límite
for (it = B.begin(); it != B.end(); it++) {
    (*it)->mover();
    if ((*it)->fuera()) {
        gotoxy((*it)->X(), (*it)->Y()); printf(" ");
        delete(*it);
        it = B.erase(it);
    }
}

//Se les aplica los métodos mover y colisión a los asteroides de la lista
for(itA = A.begin(); itA != A.end(); itA++){
    (*itA)->moverast();
    (*itA)->colision(N);
}

//Se detectan las colisiones, se eliminan, se posicionan en otra posición, lo mismo con las balas
for(itA = A.begin(); itA != A.end(); itA++){
    for(it = B.begin(); it != B.end(); it++){
        if((*itA)->X() == (*it)->X() && ((*itA)->Y()+1 == (*it)->Y() || (*itA)->Y() == (*it)->Y())){
            gotoxy((*it)->X(),(*it)->Y()); printf(" ");
            delete(*it);
            it = B.erase(it);

            A.push_back(new AST(rand()%74 + 3, 4));
            gotoxy((*itA)->X(),(*itA)->Y()); printf(" ");
            delete(*itA);
            itA = A.erase(itA);
            score += 5;
        }
    }
}

//Se aplica
N.morir();
N.mover();
Sleep(40);
}

```

Este es el resultado final:

