INFORME TÉCNICO DEL PROYECTO

NOMBRE DEL PROYECTO/JUEGO: F . A . R

Integrantes: Manfredi Angelo, Urieta Franco, Moran Rodrigo

Ingeniería Informática

materia: Informática II

Introducción **2**

Situación **2**

Idea **2**

Desarrollo de la solución **2**

descripción del juego 3
descripción del jugador 3
descripción del personaje y escenario del juego 3
descripción de las mecánicas del juego 3
clase correspondiente al jugador 3
clase correspondiente al obstáculo/asteroide 4
Implementación de lista 5
implementación de cola 6
implementación de pila 7

Resultados **9**

Menú principal 9
Menú de puntuaciones 10
menú de pausa 11
Pantalla de muerte 12
pantalla del juego 13
Archivo generado 14

Texturas, archivos y fuentes utilizados **14**

Textura utilizadas 14
Nave principal(Jugador)Obstáculo principal (Asteroide) 14
Fondo del juego 14
Archivo de cabecera de terceros utilizados 15
Colisión.h y colisión.cpp 15
Fuentes utilizadas 15
8-bit Arcade In 15

Introducción

Este es un informe técnico acerca del proyecto que fue encomendado a realizar para la materia de "Informática II" de Ingeniería Informática.

A continuación se contara la situación que fue presentada, las ideas que tuvimos para resolver "el problema", como fue el planteamiento y desarrollo de la solución

Situación

Los profesores de la materia dijeron que se tenía que realizar un proyecto el cual corresponde al segundo parcial de la materia. Este proyecto consistía en la realización de un juego utilizando la biblioteca gráfica "SFML", a su vez, el juego debía tener completando los temas que vimos durante el semestre pero principalmente debía tener una lista enlazada, una cola y una pila.

Idea

Para el juego se nos ocurrió el desarrollar un videojuego del género "Bullet Hell" pero más simplificado. El juego consistiría en una nave espacial la cual esquivará una serie de obstáculos y, mientras más tiempo lleve la nave sin chocar contra algún obstáculo, mayor será la puntuación del jugador.

Para poder implementar listas, colas y pilas se nos había ocurrido el realizar:

1. una lista de enemigos
2. una pila para cargar las puntuaciones de la partida
3. una cola para tomar las puntuaciones y luego mostrarlas

Para los demás temas supusimos que los íbamos a terminar utilizando mediante se iba desarrollando el juego lo que si, teníamos en claro, que todas las puntuaciones iban a encontrarse en un archivo el cual deberíamos de abrir para luego leer.

Desarrollo de la solución

Para poder empezar a desarrollar el proyecto primero se tuvo que aprender a utilizar la librería gráfica SFML, a su vez , se decidió: el crear las texturas que tendrían los sprites del juego y el fondo del juego, se buscó una fuente de texto para los menús y, por último, se buscó un archivo de cabecera para poder mejorar el tema de colisiones.(todos lo que se mencionó anteriormente se puede observar en el apartado

“Texturas, archivos y fuentes utilizadas”).

Para el jugador y los enemigos/obstáculos se realizaron clases para cada uno en la cual se encuentra su comportamiento.

Para implementar listas se decidió realizar una lista enlazada de enemigos los cuales se van creando a medida que el jugador se mantiene con vida y, estos mismos, se van borrando de la lista a medida que salen de pantalla. A su vez, cuando el jugador pierde la lista se borra con el destructor de la clase.

Con respecto a las pilas y colas se decidió implementarlas de la siguiente manera. Mientras que el jugador se encuentre realizando partidas todas las puntuaciones que haga, en cada intento, se van introduciendo a una pila la cual, en el momento que el jugador decide volver al menú principal, son cargadas en un archivo “.txt”. Luego de esto, cuando el jugador decide ver el menú de puntuaciones, se lee el archivo y las puntuaciones se introducen a una cola la cual se encargará, posteriormente, de ir devolviendo esas puntuaciones para que les cambien el tipo de variable(en este caso de tipo “int” a tipo “string”) y, por último, sean mostradas en pantalla.

- descripción del juego

Como se mencionó anteriormente el juego es del género “bullet hell”. Este consiste en esquivar todos los elementos que aparecen en pantalla.

- descripción del jugador

El jugador sería cualquier persona, la cual esté interesada en los juegos de este género ,pero teniendo en cuenta que la dificultad del mismo es baja/media si es que nunca jugaste a uno con anterioridad. De ser el caso contrario el juego es bastante fácil.

- descripción del personaje y escenario del juego

La persona principal, el cual maneja el jugador, es una nave espacial la cual está en el espacio esquivando asteroides.

- descripción de las mecánicas del juego

la mecánica del ujuego consiste en esquivar todos los asteroides que van apareciendo para asi lograr una mayor puntuación.

- clase correspondiente al jugador

```
class Spaceship {
private:
    int x;
    int y;
    Texture tex_spaceship;
    Sprite sp;
    int animation;

public:
    //Constructor del objeto
    Spaceship();

    //Seteo de la posicion del jugador
    void setPosition();

    //Delimita hasta donde se puede mover el jugador
    void delimitation();

    //Mueve al jugador
    void move();


    //Devuelve el sprite del objeto
    Sprite getSprite();

    //Dibuja el objeto en pantalla
    void draw(RenderWindow &window);
};
```

- clase correspondiente al obstáculo/asteroide

```
class Asteroid {
private:
    int x;
    int y;
    Texture tex_asteroid;
    Sprite sp;
    int animation;

public:
    Asteroid(int pos);


    bool move();

    Sprite getSprite();


    void draw(RenderWindow &window);
};
```

- Implementación de lista

```
//Crea una lista enlazada de objetos del tipo asteroid
LinkedList<Asteroid *> asteroid_list;
```

```
//funcion encargada de la aparicion de asteroides
asteroids(score, &clock, &asteroid_list);

//funcion encargada de las colisiones entre el jugador y los asteroides
colisions( &player, &asteroid_list, &life);
```

```
void colisions(Spaceship &player, LinkedList<Asteroid *> &asteroid_list, bool &life) {

    for (int i = 0; i < asteroid_list.size(); ++i) {
        if(Collision::PixelPerfectTest(asteroid_list.get(i)->getSprite(),player.getSprite())){
            life = false;
        }
    }
}
```

```
void asteroids(int score, int &clock, LinkedList<Asteroid *> &asteroid_list) {
    if (score <= 500 && clock == 0) {
        int aux = rand() % 3 - 0;
        for (int i = 0; i < aux; ++i) {
            asteroid_list.push_front( dato: new Asteroid( pos: rand() % 9 - 0));
        }
    }
    if (score > 500 && score < 1000 && clock == 0) {
        int aux = rand() % 4 - 0;
        for (int i = 0; i < aux; ++i) {
            asteroid_list.push_front( dato: new Asteroid( pos: rand() % 9 - 0));
        }
    }
    if (score > 1000 && score < 1500 && clock == 0) {
        int aux = rand() % 6 - 0;
        for (int i = 0; i < aux; ++i) {
            asteroid_list.push_front( dato: new Asteroid( pos: rand() % 9 - 0));
        }
    }
    if (score > 1500 && clock == 0) {
        int aux = rand() % 8 - 0;
        for (int i = 0; i < aux; ++i) {
            asteroid_list.push_front( dato: new Asteroid( pos: rand() % 9 - 0));
        }
    }
    for (int i = 0; i < asteroid_list.size(); ++i) {
        if(!asteroid_list.get(i)->move()) {
            asteroid_list.erase(i);
        }

    }
    clock++;
    if (clock == 50)
        clock=0;
}
```

```
if (!life) {
    //si el jugador pierde se borra la lista, se pasa la puntuacion al stack y muestra la pantalla de meurte
    asteroid_list.~LinkedList();
    stack_score.push(score);
    game = false;
    dead = true;
}
for (asteroid_list.loopStart();!asteroid_list.loopEnd();asteroid_list.loopNext()) {
    //Recorre la lista de asteroides para dibujarlos
    asteroid_list.loopGet()->draw( &: window);
}
```

- implementación de cola

```
while (menu_score) {
    //se crea la cola en donde se van a estar las puntuaciones
    std::queue<String> score_queue;
```

```
   //pasa las puntuaciopnes del archivo a la cola
   extracData( &: score_queue);
```

```
void extracData(std::queue<String> &score_queue) {
    //TOMAS LOS DATOS QUE SE ENCUENTREN EN EL ARCHIVO
    //LOS PASA A UNA COLA

    String scr;
    std::ifstream reed;
    int aux;
    reed.open( s: "score.txt");
    while (!reed.eof()) {
        reed >> aux;
        scr = std::to_string(aux);
        score_queue.push(scr);

    }
    reed.close();

}
```

- implementación de pila

```
//Crea una pila de puntuaciones
std::stack<int> stack_score;
```

```
if (Keyboard::isKeyPressed( key: Keyboard::S)) {
    load_score( &: stack_score);
    menu_score = true;
    break;
}
```

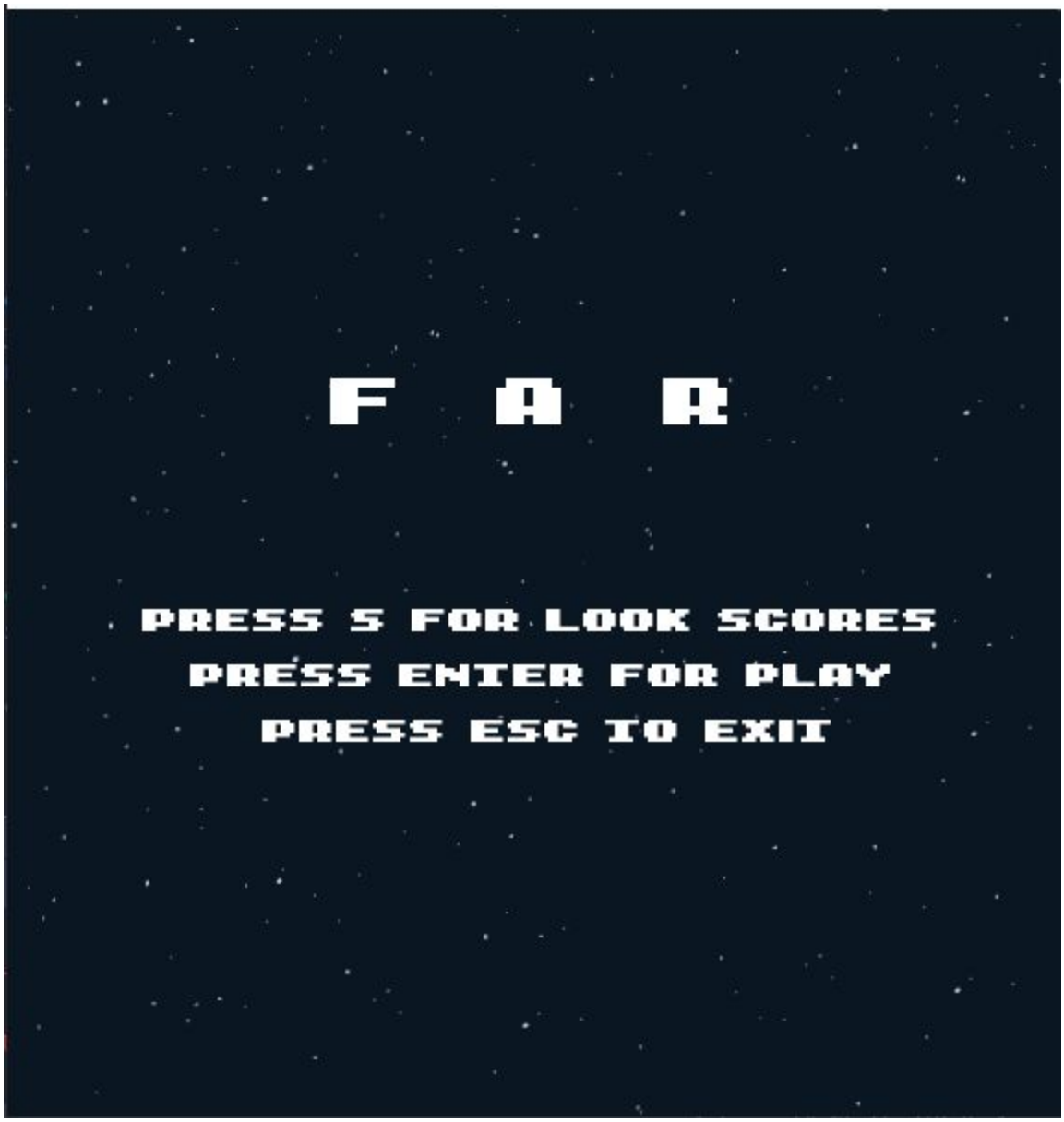
```cpp
void load_score(std::stack<int> &stack_score) {
    //ABRE EL ARCHIVO EN DONDE SE GUARDA LAS PUNTUACIONES
    //PASA LOS DATOS QUE SE ENCUENTRARN EN EL STACK AL ARCHIVO

    std::ofstream write;
    write.open( s: "score.txt");
    while (true){
        for (int i = 0;!stack_score.empty(); ++i) {
            int a = stack_score.top();
            write << a << std::endl;
            stack_score.pop();
        }
        break;
    }
    write.close();
}
```

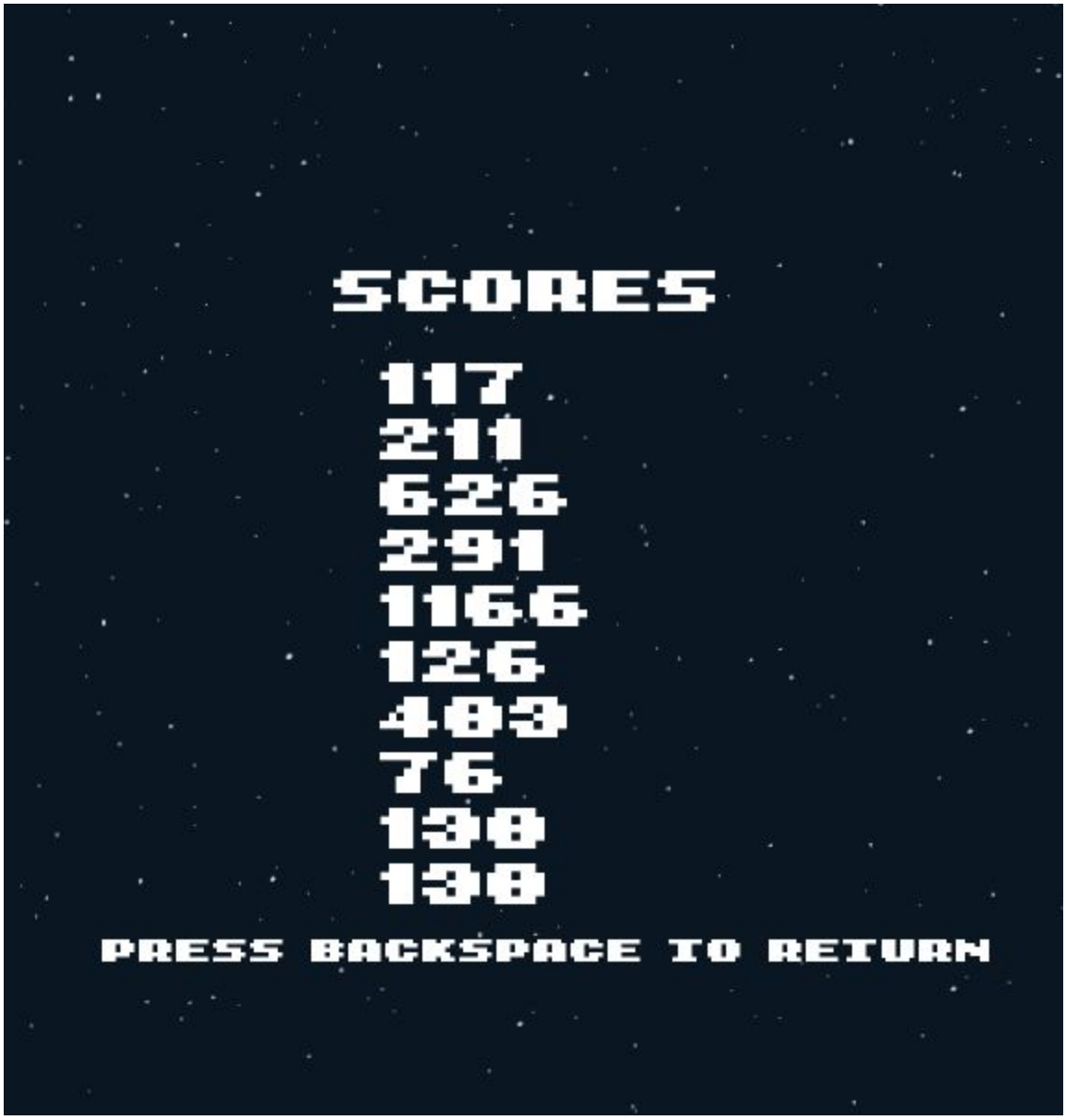
```cpp
if (!life) {
    //si el jugador pierde se borra la lista, se pasa la puntuacion al stack y muestra la pantalla de meurte
    asteroid_list.~LinkedList();
    stack_score.push(score);
    game = false;
    dead = true;
}
```

Resultados

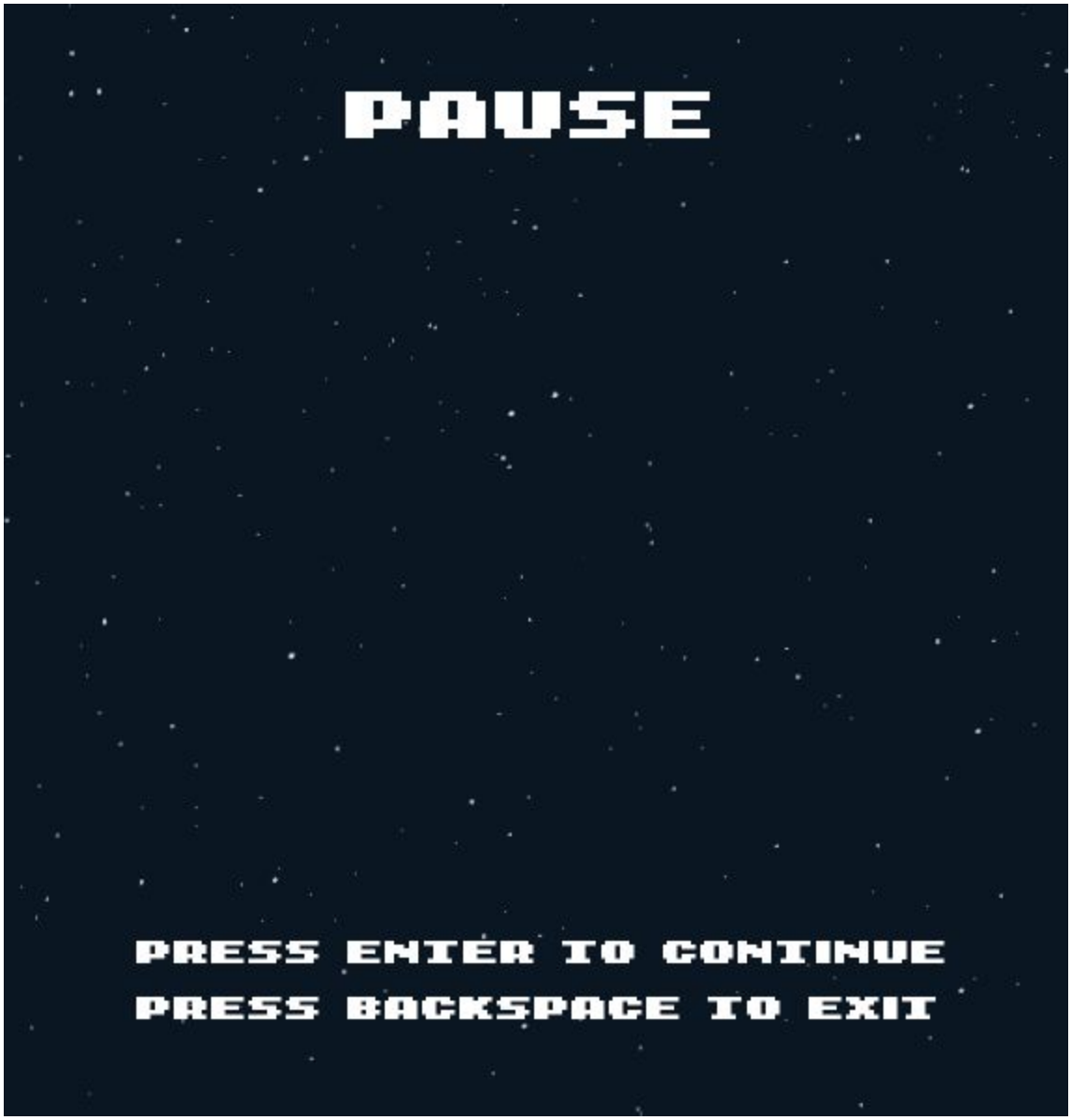
- Menú principal



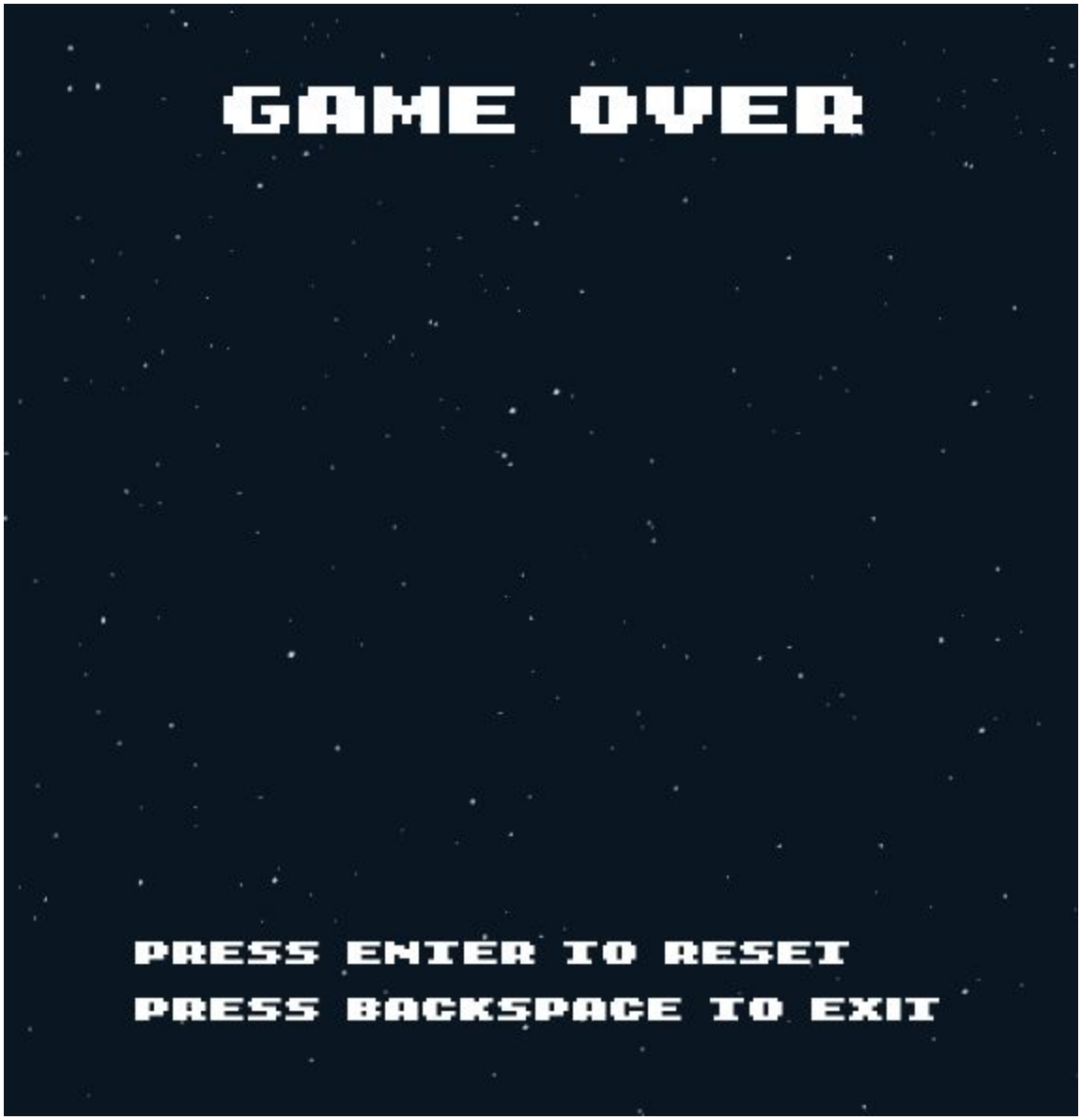
- Menú de puntuaciones



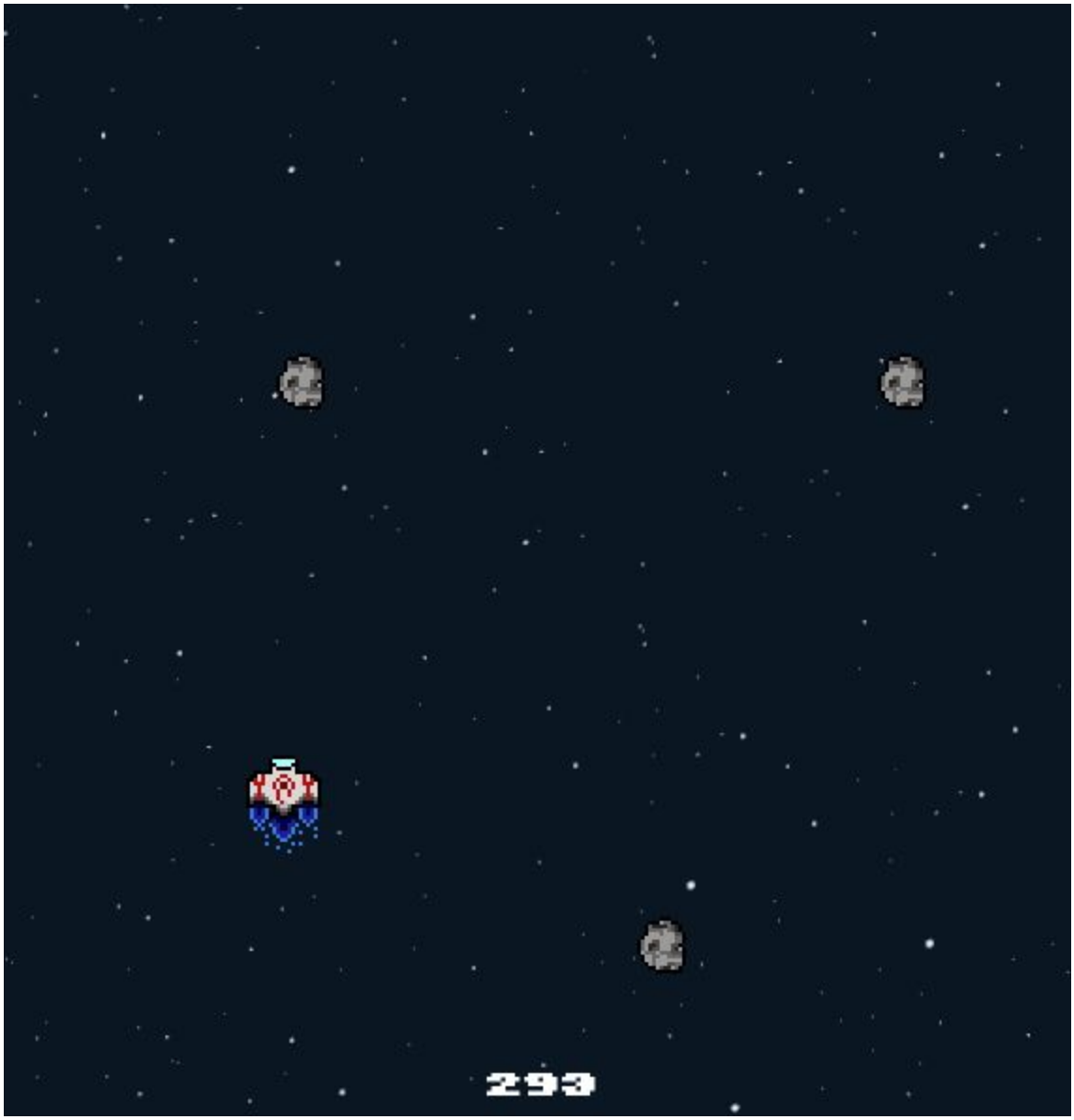
- menú de pausa



- Pantalla de muerte



- pantalla del juego



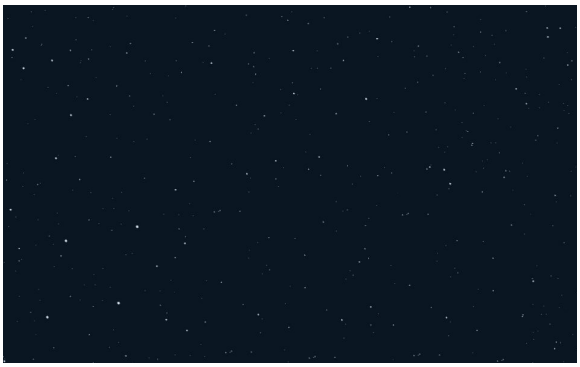
- Archivo generado

```
117
211
626
291
1166
126
483
76
138
```

Texturas, archivos y fuentes utilizados

- Textura utilizadas
    - Nave principal(Jugador)Obstáculo principal (Asteroide)

    - Fondo del juego



autor:https://opengameart.org/content/space-background-7

- Archivo de cabecera de terceros utilizados
    - Colisión.h y colisión.cpp

      Autor: Nick Koirala (original version), ahnonay (SFML2 compatibility)

- Fuentes utilizadas
    - 8-bit Arcade In

      autor:Damien Gosset-https://www.dafont.com/es/8-bit-arcade.font