CC3501-2 Modelación y Computación Gráfica para Ingenieros - Tarea 2D

BattleSpace

Manuel Olguín

El Desafío

Reglas del juego:

Es un juego para 2 jugadores.

Cada jugador controla una nave.

Cada nave tiene un blanco en la parte de atrás y un arma en la parte de adelante.

El objetivo es golpear con el arma el blanco oponente.

Reglas del Juego

Para mover la nave se utilizan 3 botones, el primero la gira a la derecha, el segundo a la izquierda y el tercero hace que avance.

En la etapa hay obstáculos que dañan a las naves.

Puntajes

(1.0 ptos) dibujar las naves y los obstáculos.

(2.0 ptos) Programar el movimiento de las naves y los eventos.

(2.0 ptos) Programar el algoritmo para las colisiones.

Puntajes

(1.0 ptos) Definir las condiciones de victoria y derrota, mostrando mensajes en el terminal.

(Bonus) (1.0 ptos) implemente un sistema de “combustible” que permita a las naves moverse, y los elementos para rellenar el estanque durante la batalla, puede presentar la información del estanque en el terminal.

La Solución

Implementación

Juego escrito 100% en Java.

Se utilizaron dos librerías adicionales:

a. Slick2D - Set de utilidades para simplificar la creación de juegos en Java. Utiliza LWJGL.

b. LWJGL - Librería que otorga acceso a OpenGL, OpenAL y OpenCL en Java.

Implementación

Además, dado que el juego está escrito en Java, es inherentemente multiplataforma.

Para su distribución, se ha utilizado la herramienta JarSplice, la cual es capaz de crear ejecutables para todos los grandes sistemas operativos de escritorio a partir de archivos JAR.

Clases Implementadas

Asteroid.java
Bullet.java
FuelBottle.java
InGame.java
Life.java
Menu.java
MultiShot.java
ObjectHandler.java
P1Win.java
P2Win.java
Pause.java
Ship.java
ShipGame.java
SpaceObject.java
Tie.java

Objetos

SpaceObject.java

"Superclase" genérica que engloba todos los elementos móviles del juego, los cuales luego la extienden.

Implementa métodos para movimiento inercial (a través de Vector2f), de representación gráfica (imágenes y figuras), de detección de colisiones y para marcar que un objeto debe ser destruído, entre otros.

Ship.java

Esta clase se encarga de dibujar la nave, actualizar su posición y velocidad en cada instante, y además se encarga de agrupar sus balas cuando dispara.

Posee métodos para acelerar, girar, disparar y para cambiar su estado de combustible, vidas, etc.

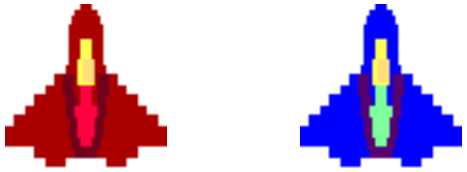
Ship.java - Geometría

La geometría de las naves se implementó como la unión de un círculo en la parte delantera (que representa un escudo) y un triángulo para la parte posterior.

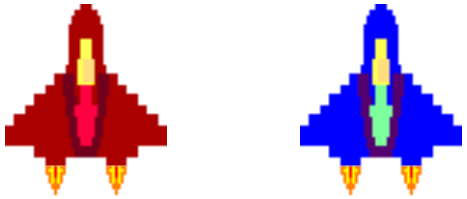
De esta manera, sólo los impactos en la parte posterior causan daño.

Ship.java - Sprites

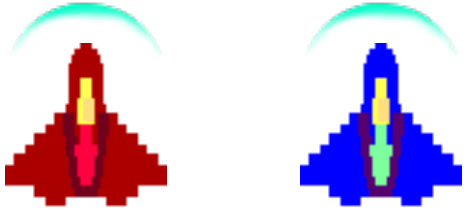
Normal:



Acelerando:



Escudo:



Bullet.java

Representa una bala en el campo de batalla.

Básicamente implementa el movimiento inercial de SpaceObject y agrega un contador temporal que destruye la bala luego de un periodo determinado de tiempo.

Asteroid.java

Clase asociada a los asteroides del campo de batalla.

Implementa el movimiento inercial de SpaceObject, e implementa métodos para calcular la "integridad" (hitpoints) del asteroide de acuerdo a su tamaño.

FuelBottle.java y Life.java

Implementación de los powerups de combustible y vidas adicionales.

Son clases básicas que implementan el movimiento inercial, y que al colisionar con algún jugador, generan un cambio en una variable.

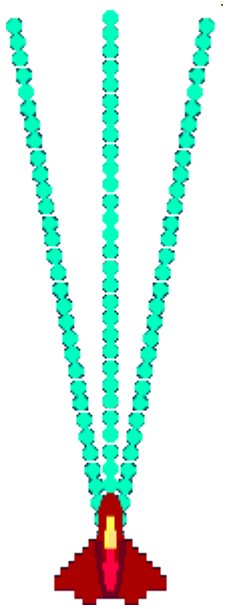
MultiShot.java

Clase asociada a otro powerup, el de disparo múltiple (MultiShot). Al igual que las clases anteriores, extiende SpaceObject.

Cuando este objeto colisiona con un jugador, cambia temporalmente su manera de disparar.

MultiShot.java

Normal



MultiShot

Estados del Juego

ShipGame.java

Clase Main del juego.

Esta clase es invocada al iniciar la aplicación; se encarga de iniciar los "estados" del juego y crear la ventana y fijar sus dimensiones.

Finalmente, invoca el estado inicial del juego.

Menu.java

Clase que contiene el estado inicial del juego.

Muestra una imagen con el nombre del juego y espera a que el usuario presione ENTER para iniciar.

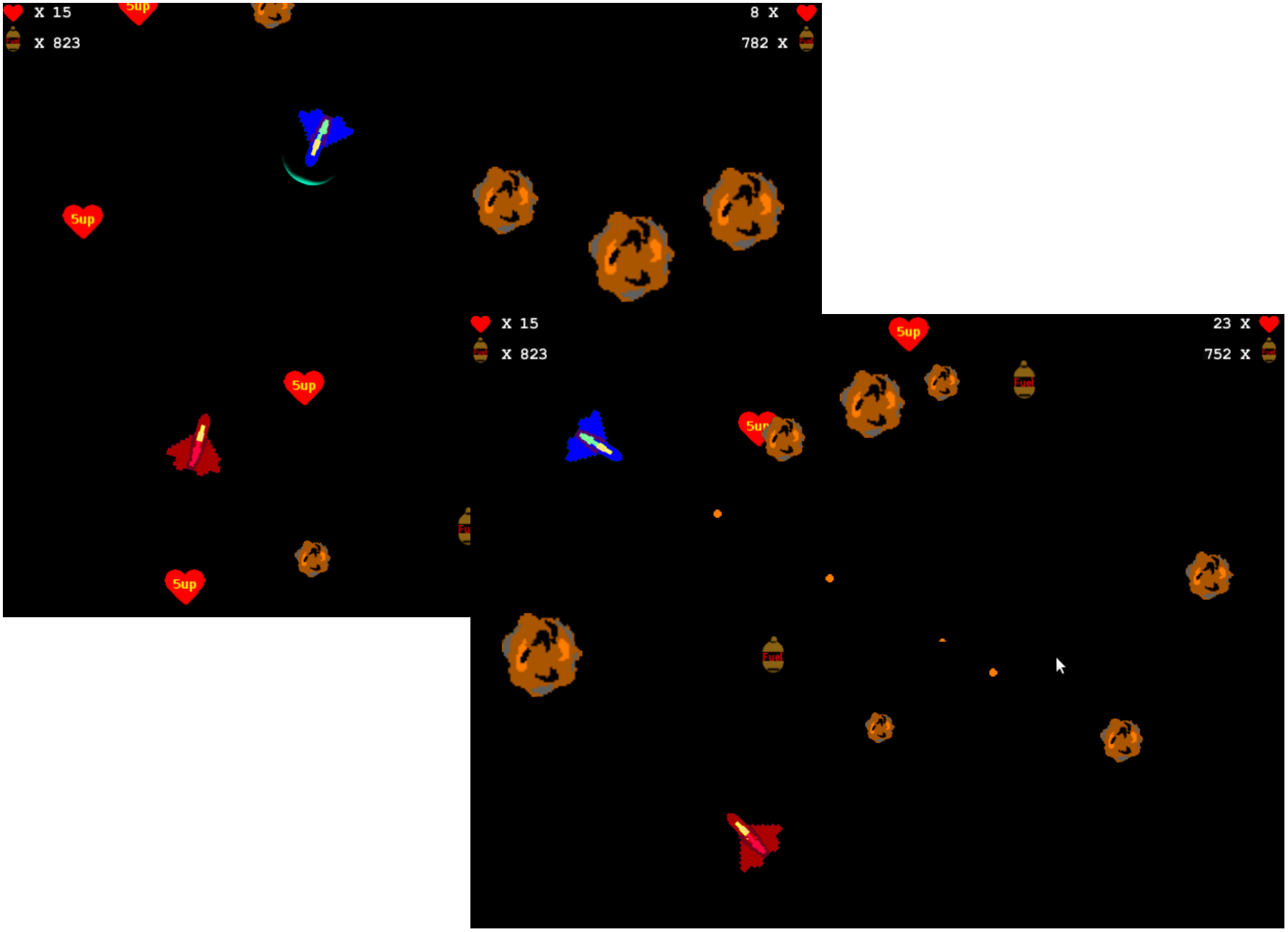
Por otro lado, si se presiona ESCAPE, finaliza.

InGame.java

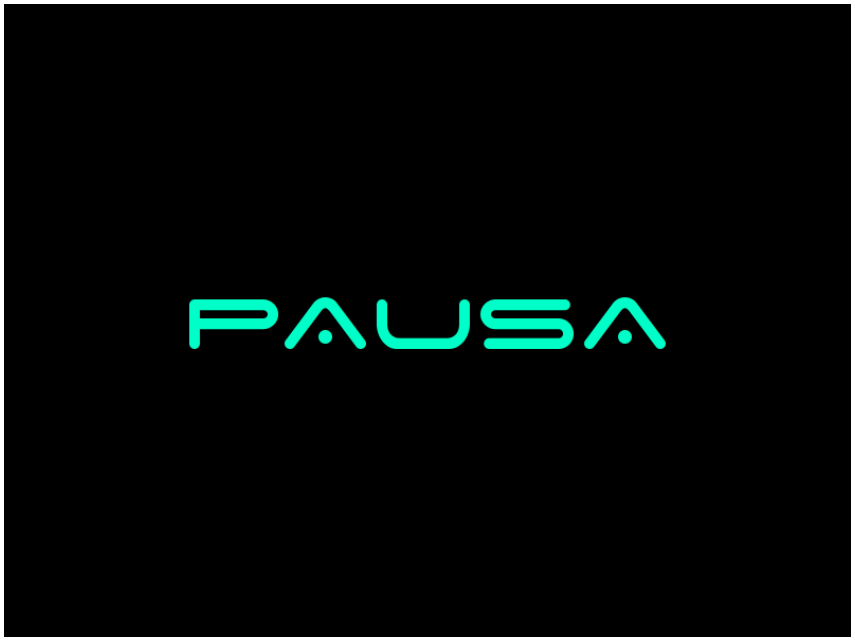
Estado principal del juego.

Se inicializan los jugadores, los asteroides, y se actualiza el estado del juego en cada instante. Además, llama a los métodos de de cada objeto para dibujarlos en pantalla.

Finalmente, también implementa métodos para detectar las colisiones entre los objetos.


X 15
X 823
8 X
782 X
X 15
X 823
23 X
752 X
5up
Fuel

Pause.java



Estado al que ingresa el juego cuando se presiona "P" en InGame.

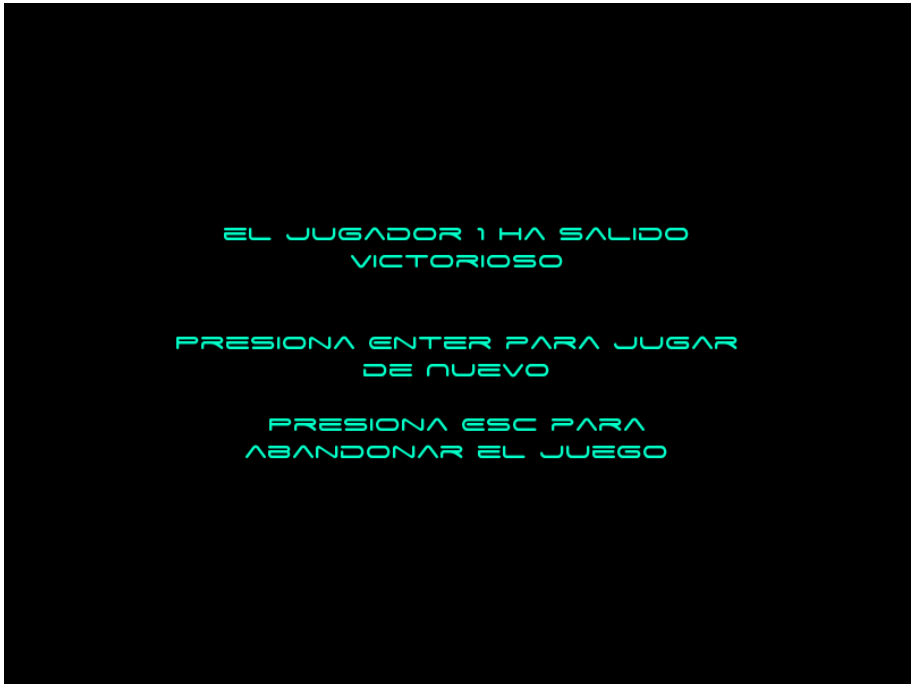
Congela el estado InGame y muestra una imagen.

Si se presiona ENTER, reanuda el juego. Si se presiona ESCAPE, finaliza.

P1Win, P2Win y Tie

Corresponden a estados invocados en caso de victoria del jugador 1, 2 o en caso de empate, respectivamente.

Presentan una imagen y opciones para reiniciar el juego o finalizar.




X 15
X 823
23 X
752 X
8 X
782 X
BATTLESPACE
PRESIONA ENTER PARA COMENZAR

Conclusión

Conclusión

Todas las imágenes utilizadas en el juego fueron creadas utilizando GIMP 2.8.

La música y los sonidos fueron obtenidos desde FreeSound.org, y corresponden a archivos licenciados bajo Creative Commons.

El juego se encuentra disponible en GitHub, y está licenciado bajo la GNU GPL v3.