IPL
escola superior de tecnologia e gestão
instituto politécnico de leiria

Department of Computer Science
www.dei.estg.ipleiria.pt

Advanced Game Programming Topics

Your own 2D Game Engine 2.0

Project

Using the SDL2 library, GLSL for creating vertex and fragment shaders, the Box2D physics engine and the C++ programming language (C++11 specification), you should create a new version of your previous two-dimensional (2D) Game Engine.

Game: Xenon 2000

For testing your Game Engine, you must use the assets of the **Xenon 2000: Project PCF** and create a very simple horizontal scroller clone of the Xenon 2000 game. Therefore, you will need to load the assets corresponding to the spaceship, missile and enemy's sprite sheets, backgrounds, etc.

Figure 1: spaceship sprite sheet ("Ship1.bmp").

Spaceship

The spaceship should move according to the keyboard or gamepad input. Also, when the ship is moving to the left, it must have the animation of turning to the left (see the first three tiles of Figure 1) and when is turning right, an animation of turning right (last three tiles of Figure 1).

Missiles

The spaceship should also fire missiles when the player presses the Space bar on the keyboard, or the X button on the PlayStation controllers or the A button on the Xbox controllers.

There are three types of missiles: light, medium and heavy, each has a corresponding row on the "**missile.bmp**" sprite sheet:

Figure 2: missile sprite sheet ("missile.bmp").

Every time the Spaceship gets a weapon power up, its fire power improves from light to medium or from medium to heavy. Medium missiles have twice the fire power of the light missiles. And the heavy missiles have twice the fire power of the medium missiles.

Every time a missile collides with an enemy, the missile is destroyed and should happen an explosion. You must use the following sprite sheet for the explosion:

Figure 3: explosion sprite sheet ("explode16.bmp").

Loner

The Loner is an enemy and is represented by the following sprite sheet:

Figure 4: enemy loner sprite sheet ("LonerA.bmp").

This enemy moves horizontally at a certain speed. The loner fires at each 2 second interval a projectile in the Spaceship direction.

Enemy Projectiles

The enemy projectiles are fired by Loners and represented by the following sprite sheet:

Figure 5: sprite sheet of the enemy projectiles (“EnWeap6.bmp”).

Enemy projectiles also explode when they collide with the Spaceship.

Rusher

The rusher is also an enemy and is represented by the following sprite sheet:

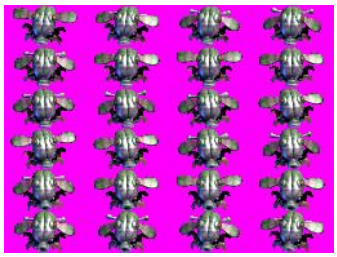

Figure 6: sprite sheet of the rusher enemy ("rusher.bmp").

This enemy moves vertically at a certain speed.

Drone

The drone is also an enemy and is represent by the following sprite sheet:

Figure 7: sprite sheet of the drone enemy ("drone.bmp").

Drones have a horizontal sinusoidal move and appear in packs (see reference video).

Stone asteroids

Stone asteroids are hazards represented by three different sprite sheets, according to their size:

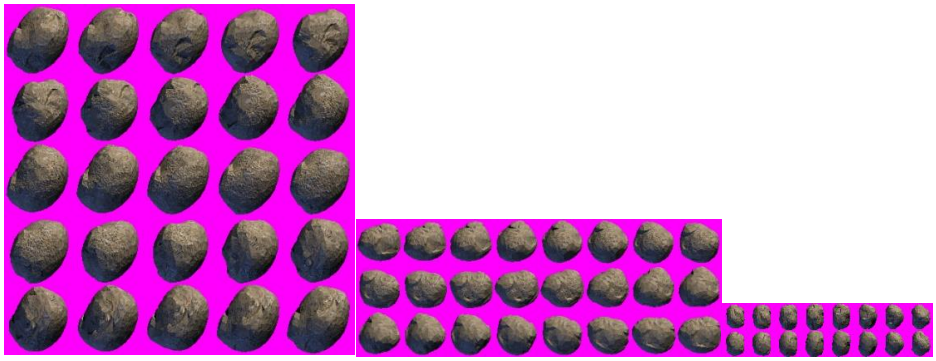

Figure 8: sprite sheets of the stone asteroids ("SAster96.bmp", "SAster64.bmp" and "SAster32.bmp").

When a big or medium size asteroid receives enough damage, it is split into three smaller asteroids (see reference). When the smaller asteroid is hit, it explodes.

Metal Asteroids

Metal asteroids are hazards that are indestructible and are represented by three different sprite sheets, according to their size:

Figure 9: sprite sheets of the metal asteroids ("MAster96.bmp", "MAster64.bmp" and "MAster32.bmp").

Companion

The Space Ship can have up to two companions, which are represented by the following Sprite Sheet:

Figure 10: sprite sheet of the companion ("clone.bmp").

Companions are like “clones” of the spaceship, they also have fire power, can die and get power ups. Basically, they have the same behavior as the Ship, but they are close to the Spaceship. When the Spaceship fires, their companions also fire. When the Spaceship dies, the companions also die.

Shield Power Up

The shield power up is represented by the following sprite sheet:

Figure 11: sprite sheet of the shield power up ("PUShield.bmp").

The shield power up restores the energy (life) of the Spaceship or the Companion, depending on who “catches” the power up.

Weapon Power Up

The weapon power up is represented by the following sprite sheet:

Figure 12: sprite sheet of the shield power up ("PUWeapon.bmp").

The weapon powerup upgrades the fire power (missile type) of the Spaceship or the Companion, depending on who captured this power up.

Game Engine

The implemented Game Engine must follow the Object-Oriented paradigm. Furthermore, it should be as generic as possible, so it can be used for creating other kinds of 2D games. Thus, the Game Engine and the Xennon clone game should be created in separate XCode/Visual Studio Projects. The project corresponding to the Xennon clone game should include the game engine has an external library.

Also, resource management must be considered: every time an allocated resource is not required, it should be freed. You must avoid resource/memory leaks at all costs.

For the **Game Loop** inside your game engine, you must separate the **game logic** from the **rendering** (the rendering part will be evaluated in Computer Graphics class). During each loop, first all the actors or game objects have their game logic updated (positions, etc.), and then all actors are rendered. For more information, please go to:

https:// docs.unity3d.com/560/Documentation/Manual/ExecutionOrder.html. Here, you have a short description of how the Unity[1] Game Engine works

Reference

You can use the following video as a reference for your implementation of the Xennon 2000 clone: https://www.youtube.com/watch?v=JRNxocsPOvY.

Report

Together with the project, there must be a report in PDF format with a description of the Game Engine, including a class diagram of the Game Engine, the description of all the implemented

[1] https://unity3d.com/pt/

algorithms and a justification of the choices made. The report shall site all references (books, websites, etc.) in which the algorithms were based. The report should also explicitly include the objectives achieved and not achieved.

Extra Credits

For the extra credits, you should use SDL 3.0 instead of SDL 2 and implement audio using the new audio system of SDL 3.0.

Note: The extra credit has a Mark of 10% (2 in 20). However, the final grade of the project cannot be higher than 100% (20 in 20).

Marks

This part of the project has the following marks:

Criteria	Mark
Event Handling (engine)	5%
Game Loop (engine)	10%
Resource Management (engine)	5%
Engine Architecture	40%
Xennon 2000 Clone	30%
Report	10%
Extra Credits	10%

Rules

The practical project must follow the following rules:

1. Deadline for submission: 9th January 2025
2. The work can be performed by groups of two students (projects with groups larger than two, will not be accepted).
3. On 9th January, the project will be presented on an oral defense.
4. The project should be submitted to github and shared with the professor as project collaborator.