POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Project 2

Xenon Clone

Project Report

1st Examination Period

Advanced Game Programming Topics
Undergraduate in Games and Multimedia

2025/2026

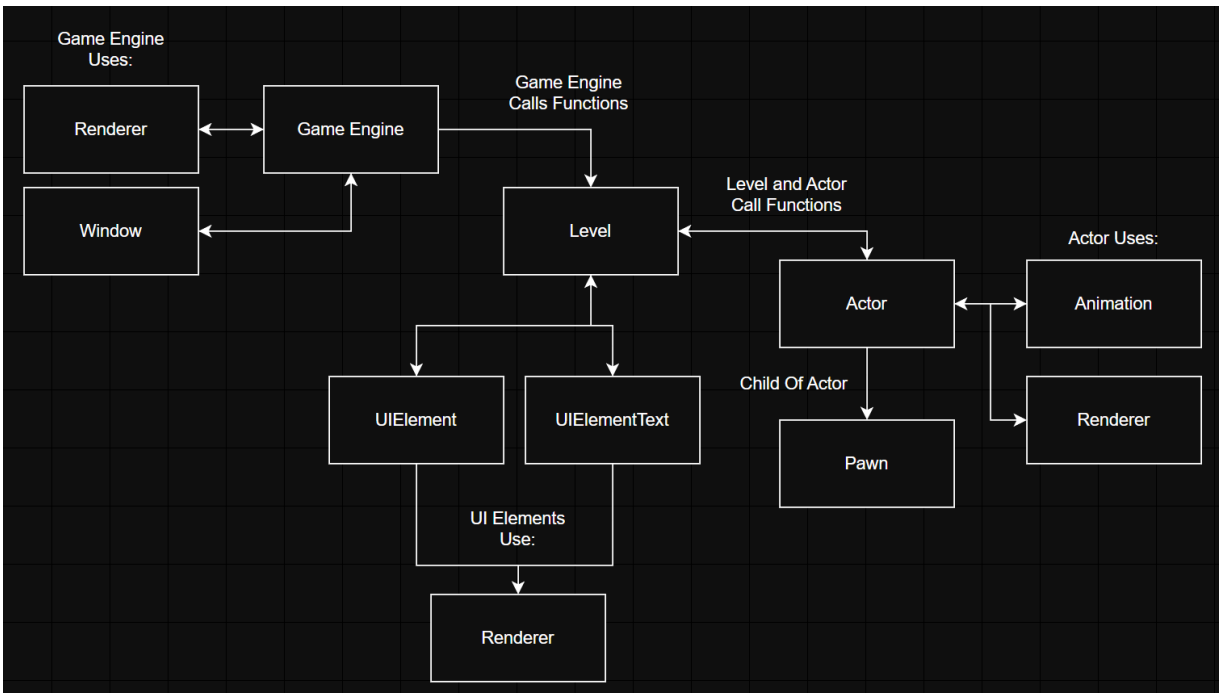
Professor Gustavo Reis

2232222 | Daniel Pitt Moradi

2232860 | Gonçalo Pedroso Pires

Leiria, 4 February of 2026

1. Class Diagram Game Engine



2. Computer Graphics

2.1 Renderer

In the renderer class the constructor initializes the OpenGL, creates the shader program from the vertex and fragment shader and initializes the quad, setting the vertices, indices, generating vao, vbo and ebo.

The function LoadTexture is used to store the texture given by the actor to be used to render, in this function it checks if the texture already exists in the array if it does it returns the handle a int value for the actor to use when saying which texture to use when it needs to be rendered, if the texture isn't in the array it will convert the texture surface to be RGBA, then if the actor sends usesColorKey boolean is set to true it filters the magenta color from the texture else the whole texture is stored and returns the handle(int value) of where the texture is in the array for the actor to use.

A UnloadTexture function was made to be able to unload textures from the array of textures the renderer has if needed, the function isn't used due to the game's small amount of textures and game implementation.

Then the DrawTexture function has dependencies that the actor needs to pass when calling it, the texture handle which is the int variable received when its texture is loaded, the position of the actor, scale, rotation, a vec4 responsible for handling which part of the texture is used, a boolean of useSolidColor for object that use the whole texture and color if the actor uses solid color variable is true. It creates an identity matrix called model to set the translate, rotation which places the object in the center to rotate it on its axis and return it to its position and scale, then the useSolidColor variable is used to set a uniform as well as the color in the fragment shader that controls if the shader paints the texture with the color or simply uses the texture.

The model identity matrix is then used to set the uniform model in the vertex shader. With the vec4 we override the vertices positions, having to then rebind the vbo because we altered the vertices, activating the textures and finally drawing the

texture.

2.2 Animation

For the actors that had animations the actor calls a function in the animation class that now because we are using OpenGl instead of returning the current frame of the animation it returns the position of the frame in the texture that is being used by the actor in a vec4 to be used in the renderer when the actor is rendered.

2.1.3 UI Bases

2.1.3.1 UIElement

This class is the base for the health bar and life count ui elements. It has functions to position the object and be able to set it visible or not if needed, then it has virtual functions for the update and draw for the children to override it.

2.1.3.1.1 UIHealthbar

On the constructor it is set the size, current health and max health and position of the health bar and calling to the renderer to load a texture without using the color key to be able to use the full texture without filtering.

Then it has a SetHealth function to be able to alter the current health of the player whenever damage is taken or healed.

The Draw function starts by clamping a percentage between the current health and max health to 0 and 1, getting the color of the healthbar with the GetHealthColor function, then it draws the background of the healthbar and then the healthbar on top of it with scaling the width with the health percentage and cropping the texture depending.

The GetHealthColor returns a vec3 which is a selected color which is green if above 66%, yellow if above 33% and red when below 33%.

2.1.3.1.2 UILifeCount

On the constructor the number of lives, the size and spacing between icons is set. Having a function to be able to set the lives whenever the player loses a life to be called. The Draw function draws the UISpaceship texture as many times as there are lives with the designated spacing between them.

2.1.3.2 UIElementText

This class is used for text elements, on the constructor it is able to set the char width, height, the spacing between letters, characters per row on the texture sheet, characters per column on the texture sheet, text color, text position and scale.

Having functions to add score to a variable or set the score if the text has numbers, a function to store the text to be displayed by the object which takes a dependency of a string to be able to store a text in our case store something like Player Score and then the score.

Then the draw function breaks the variable with the text to write into char, then for each char if it’s an end line it sets the position for the letter or number to the next line, if it's a space it gives a space between the next letter or number, then it gets the uvcoords of the char in the texture using the GetCharUV function and then calls the DrawTexture of the renderer.

The GetCharUV firstly uses the CharToFrame function to know the int where the char is placed in the texture, calculates the grid positions, getting texture dimensions, calculating the character size in the texture itself and then calculates the uv coordinates of the char using row and column calculations with the texture width and height.

CharToFrame is a switch case function that receives the char and returns an int, each char from the texture chosen has an int.

3. Advanced Game Programming Topics

3.1 Xenon

In the main of this class it uses the engine to call the initialize function to create the window and everything related to the setup of the game, then it uses a reference of the Xenon Level to set the current level of the engine as the xenon level and then tells the engine to start the game.

3.2 Xenon Level

The Xenon Level is a child of the Level class, setting every actor that will spawn in the beginning of the game as the Background, ParallaxEffects, Spaceship actors and the UI elements.

On this second project we added an Update function so we can spawn waves every 3 seconds of different types. It can be the Rusher, Loner, Companion, Drone, Asteroids or Companion Wave.

The Drone spawning function while isSpawningDrones is true this function will be called every update tick also counting a timer to spawn each a certain time after one another, this drones always spawn in the middle of the screen height with and outside of screen width.The last drone spawned will become special which means that once he dies he will drop a power up.

3.3 Spaceship

The Spaceship is a child of the pawn class, setting on its constructor its speed, the size which the spaceship is gonna be, divides its spritesheet into frames, indicates that it is an idle animation and that it isn't loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is a player and can collide with enemy and enemy missile.

In its update depending if it is moving to the left or right it will tell the animation to change to a specified animation using specific frames from the ones that were divided in the constructor and use it, if the player isn't moving to the left or right it will use the default idle animation. Also is responsible to check if the player has shot a missile and count the cooldown to be able to shoot again, the Shoot function is responsible for spawning a missile class actor on top of the spaceship. Once the spaceship shoots or moves, if the spaceship has any companion attached to him it will move the companion and shoot as well.

Whenever the spaceship collides with something it takes damage once its health is below or 0 if the spaceship has any companions those companions will be destroyed and the spaceship loses one life out of three , in case the spaceship loses all its lives the spaceship will be destroyed.

There are two functions that are called by the powerups actors one of which adds health to the spaceship and it clamps the health to the maxHealth variable so the spaceship doesn’t have more health and the maxHealth defined. The other

function is a simple method to change the type of missiles, once the spaceship collides with the missile powerup a simple change of variables from false to true and the other way around and when the player shoots verify which variable is true and it spawns the upgraded missile.

Since we needed to attach companions to the spaceship we decided to create variables of the type Companion where we can add them and remove them easily once added they already spawn locations defined.

3.4 Missiles

The Missile class is a child of the actor class and parent to other missiles classes Light,Medium and Heavy classes where they set on their constructor the damage the missile does, the size that it will take, divides its spritesheet into frames using specific frames for the animation, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is a player missile and can collide with enemy.

Since the game now it is now a horizontal scroller In its update it will move to the right at a certain speed until it reaches a bit past the window limit where if it didn't collide with anything it will destroy itself, but if it collided with an enemy, it will on the OnCollision function that is called when it collides to call the function TakeDamage of the object it collided with and create an explosion class actor and destroy itself in the process.

3.5 Loner

The Loner is a child of the actor class, setting on its constructor its health, his value of points, the size that it will take, divides its spritesheet into frames, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is an enemy and can collide with player and player missile.

In its update it will move in a vertical direction due to the game now being a horizontal scroller, when spawned the loner can have different speed(75, 100, 200, 300) until it reaches the window limit where it will move the opposite direction, meanwhile it will shoot projectiles every two seconds creating an enemy projectile class actor in front of its position.

Since the loner has health it means this actor can be damaged so since we already had a TakeDamage function on Actor class we override the function and check if the loner has less than or 0 hp and if so the actor is destroyed and points are added to the score.

3.6 Enemy Projectile

The enemy projectile is like the missile class in the constructor with the exception that the collision categories are different.

In its update it will move to the left at a certain speed due to the game now being a horizontal scroller until it reaches a bit past the window limit where if it didn't collide with anything it will destroy itself, but if it collided with the player, it will on the OnCollision function that is called when it collides to create an explosion class actor

and destroy itself in the process.

3.7 Rusher

The Rusher is a child of the actor class, setting on its constructor the size that it will take, divides its spritesheet into frames, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is an enemy and can collide with player and player missile.

In its update it will move in a horizontal direction at a certain speed depending on which variable is true. If movebackwards is true it means that the rusher will be spawning the right side of the the screen and moves to left and if it is the moveforward variable that is true it is the the other way around

3.8 Drone

The Drone is a child of the actor class, setting on its constructor the health, the size that it will take, divides its spritesheet into frames, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is an enemy and can collide with player ,player missile and companion.

In its update it will move in a horizontal sinusoidal direction at a certain speed from right to left on the X axis we only add the normal speed but to give the sinusoidal aspect on Y axis we have to have the starting position where they start and adding an offset where we use the trigonometric function sin to get a value that goes from -1 to 1 after that we add the wave length so the value that the drone move is between negative wavelength value and the positive wavelength value.

These drones can be special where in case they are special they spawn the powerups but this drone can only spawn the powerup when it dies.

The drone also has the TakeDamage where it checks if the drone gets destroyed or not.

3.9 Stone Asteroids

The Stone Astoroids are a child of the actor class, setting on its constructor the health, size that it will take, divides its spritesheet into frames, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is an enemy and can collide with player ,player missile and companion.

These asteroids can be divided into 3 smaller asteroids except the small asteroids. With that in mind on the update when being divided into 3 smaller asteroids they all have different movements but they all move sideways to the left , one of them move a bit upwards and the other downwards, but when spawning there is only one asteroid that moves sideways to left only.

The only asteroids that can be divided are the Big Asteroids and Mediums ones where they have another function that once the asteroid is destroyed it will

spawn 3 new smaller asteroids. For example the Big asteroids turn into 3 medium asteroids.

3.10 Metal Asteroids

The Metal Astoroids are a child of the actor class, setting on its constructor the health, size that it will take, divides its spritesheet into frames, indicates that it has an animation and that it is loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is an enemy and can collide with player ,player missile and companion.

The Astoroids only move laterally to the left and cannot be destroyed.

3.11 Companion

The Companion is a child of the pawn class, setting on its constructor its speed, the size which the spaceship is gonna be, divides its spritesheet into frames, indicates that it is an idle animation and that it isn't loopable, then it indicates its actor type and collision category it is part of and can collide with, in this case it is a companion and can collide with enemy ,enemy missile and powerups.

The companion is like the spaceship, it can also get upgrades and shoots.

Unlike the other actors the only thing on update is calling the parent update function because on the companion we added a function called move that is called by the spaceship so when the spaceship moves it calls the function move of the companion with the exact coordinations.

We decided to make the two different classes, one that is the pickup of the companion and the companion itself. On the pickup companion class when it collides with the spaceship it gets destroyed due to everything being done inside the spaceship code.

3.12 Power Ups

The power ups are exactly the PickUpCompanion class where they move to the left and when the player collides with it, it gets destroyed.

The spaceship does all the work where it calls the function depending on which superpower it collided with.

4. Design Choices

4.1 Computer Graphics

We decided to implement a way to store the textures in the renderer in order for the actors that use the same textures don't have to load a texture separately and instead just retrieve the texture handle from the array to have a reference to it, saving memory in the process. We also decided to make it possible to filter the magenta color off textures if wanted in the case of most textures except the health bar which we needed to use a full texture.

4.2 Advanced Game Programming Topics

We decided to implement every spawn that isn’t spawned by another actor in the level due to being easier to spawn whenever we wanted since the level is always running, and also it made it way easier to spawn different spawn types every second to simulate a wave system. We could’ve added a math library to our game engine but we decided against it because there weren't many functions that would go in that class.

5. References

Rendering sprites. LearnOpenGL. (n.d.). https://learnopengl.com/In-Practice/2D-Game/Rendering-Sprites

Shaders. LearnOpenGL. (n.d.-b). https://learnopengl.com/Getting-started/Shaders

Transformations. LearnOpenGL. (n.d.-c). https://learnopengl.com/Getting-started/Transformations