Project Xenon 2000 Clone

Undergraduate of Games and Multimedia

Advance Game Programming Topics

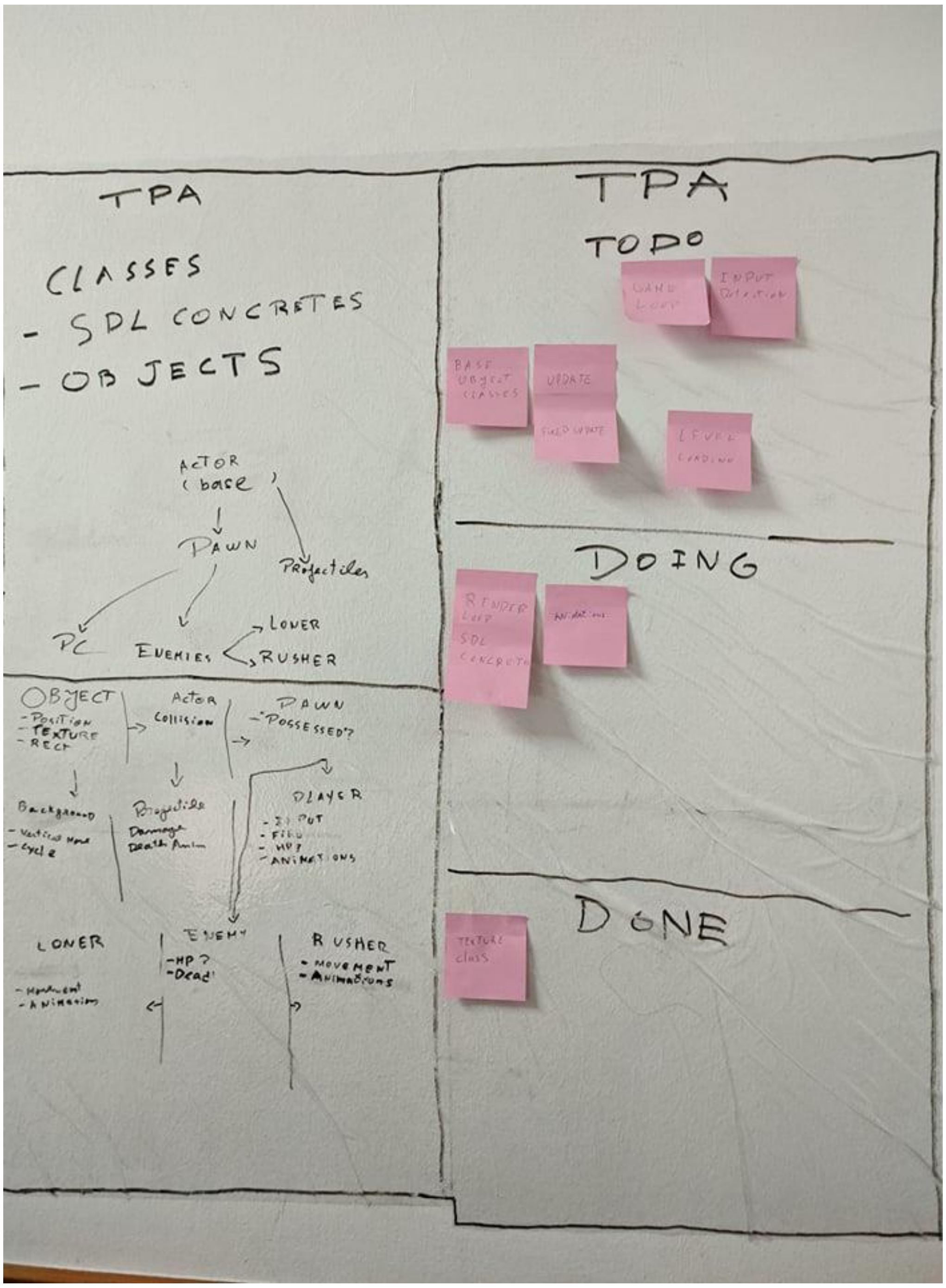
João Martins nº 2181750

João Pereira nº 2181289

Project

This was the hardest project we have ever done. It was really challenging and unfortunately, we got stuck and could not finish it. We were only able to do a part of the engine, but we are proud what we have done, because we think it is well planned and would work well.

Following, the image of our initial scheme of how we were thinking the game engine and our scrum to control our work. The diagram suffered some changes, but we did not update them on the board.



Engine:

We use the Engine class as the main class where we initialize all the other necessary components to build the game on and where we store the class instances that process all the game logic.

On the init method of the engine we initialize all the necessary components, in this case, the SDL Wrapper, the Window, the Renderer, the Level, the Texture Manager and the Input Manager classes.

The startGame method is where we run our game loop, in here we divide the loop in 3 different steps: handling inputs and events, the update method and the render.

The update method would be where we update all the physics, movement, etc, being this step mainly linked with the level update method

The handleEvents method as of now was only implemented as a basic version only allowing to quit the game, our idea was to use the inputManager Class to wrap this step and manage all events and tell the engine what inputs happened and call the respective methods.

The render method is where we do the implementation of the render loop by using methods in the rendererClass and the level class, first clearing the render, use the level class to draw everything on the level and then sending this to the screen.

To try and help make everything look cleaner we also created a clean method where we clean the memory that was in use by variables managed by the Engine class, being used only in the destructor.

Texture:

About the texture manager, we create a class and to tried to make a faster travel of the information to spear memory and performance, we created an instance of the class on init in the engine script were we pass through an argument the renderer.

On the texture class, we used the code provided on the class's pdf of SDL except for the renderer. We also used GetSDLRend() to get the actual SDL renderer.

Inputs:

For the inputs, we found a solution we considered amazing from Ather Omar (a game engine programmer working for Ubisoft that has a YouTube channel). Basically, we created a class where we implemented a singleton, a bool KeyDown method with an SDL_Scancode as an argument and with the return of the scancode, which means it would return the code of the key pressed.

We also created a void Update method that would update the variable that controlled the key. Finally, a void Release method that to delete the instance variable.

After this, we thought about creating a singleton to every class, to guarantee we would only have one instance for each, but we were trying to implement them first so we didn't do it.

Animations:

For the animations, we created 2 scripts, Tilemap and Animations. The tilemap is the script were the textures would be cut to then be used on the animations, we based part of our code on the provided on the class's pdf with the exception of the number of the columns and rows that instead of being hardcoded are arguments passed on the class.

We did a destructor to turn every variable to NULL, and 2 methods, on to get the height and width in pixels of the texture and another of each frame of it. We did these 2 methods as Vector2 to pass both height and width at the same time.

We did not finish the animation script, we only implemented some encapsulated if's.

SDL Wrapper Class:

The SDL Wrapper Class is a class that we used to make SDL a concrete class and that we can also use it as an include to avoid including SDL everywhere by including the SDL Wrapper instead.

It is also where we Initialize the SDL Subsytems to use everywhere else in the code.

Window Class:

The Window class is where we create the main window for the game and where we keep the base SDL_Window variable, being only accessible by a getter to try and prevent from messing with this window in non-intended ways otherwise.

On the Constructor of this class we create the window and save it on the variable and check if everything goes smoothly on this creation by using a throw error.

On the getSurface method we send the surface of the window so it can be used somewhere else, and we do the same with the SDL_Window itself as well in the getSDLWindow.

On the updateSurface method we send the info written to the WindowSurface to the Screen.

On the Destructor we free and the variables and destroy the window to liberate all the memory that this class used.

Vector2 Class:

We decided that to help us manage 2-dimensional vector (positions, velocities, axis on the inputs, etc) we could use a specific container, similar to the Vector 2D in Unity.

This class has 2 float variables (x and y) and we also added some basic maths to it to help when using them later, by overriding operators and creating a normalize function.

Init Error:

We based our code on the provided class's pdf, there is not much to say. It has 2 constructors, one of them have a string passed as an argument and both give errors back in cause of something goes wrong.

Actor:

We could not complete this class, we only did the header file where we created a constructor with a Vector2 argument to pass the location.

A render where we would send the information about the actor to rend.

An update to update the physics of the actor.

The set level that would give the information of the level where the actor would be, to ease the change of information between it.

The set animation would give the right animation to the actor.

Renderer Class:

We made this class to manage the SDL_Renderer and all methods that involve it directly, trying to contain most of the necessary methods to do the render loop in this class

On the Constructor of the class for this class we create the SDL_Renderer using the window created previously and save this variable on the class.

On the ClearRender method we call the SDL method to clear the render.

On the SetRenderBaseColor method we set the color to use on drawing operations.

On the AddRender method we send the provided Texture to the Render Stack to be drawn at a later point in the render loop.

The updateRender method is where we tell the render to send everything on the stack to the screen.

The GetSDLRend method is where we can send the pointer of the SDL_Renderer to anoher class that might need to use it.

Level Class:

On the Level Class we wanted to use it to manage everything related to the level itself, meaning and physics management, actors and bodies creation, keeping an array of all actors in the level, etc.

In the constructor we initialize the box2d world with the given gravity.

We also have the createBody method to put the actors that we're adding to the level to the physics world.

Then on the logic loop we have manage adding and removing actors from the main array by using too helper arrays to make sure we dont miss any calls and that we delete everything.

On the render method we send to the renderer class all info from the actors to renderer using the actors array.

With the timeStep method we advance the physics simulation by a fixed amount of time.

This was one of the parts where we got stuck, having a bit of hard time trying to understand how the contac listening system works.

References

- https://www.youtube.com/channel/UCCKlrE0p4IZxqBpq98KFBmw
- https://www.youtube.com/user/TheChernoProject
- https://www.youtube.com/user/creaper
- https://www.libsdl.org/
- https://www.youtube.com/watch?v=pJ_M_fACtB8&list=PLRqwX-V7Uu6Zy4FyZtCHsZc_K0BrXzxfE&index=12
- https://stackoverflow.com/questions/4926935/find-what-bodies-are-colliding-in-box2d-using-c
- https://box2d.org/documentation/index.html
- https://stackoverflow.com/questions/7501192/vector2-class-in-c
- https://www.thegeekstuff.com/2016/06/friend-class-cpp/
- http://www.cplusplus.com/reference/algorithm/find/