Introduction to the project

This project is a 2d-engine tool created for the edition and test of two dimension games. It operates with four basic graphical entities: Rectangles, Labels, Sprites and Backgrounds. These graphical entities are integrated using the library SFML (Simple and Fast Multimedia Library) created by Laurent Gomila. To operate through all these entities the project uses the library IMGUI (Immediate Mode Graphical User Interface) created by Omar Curnat. These libraries need to work together. Therefore, we use the binding for SFML-IMGUI created by Elias Daler.

The project also presents it's own window and scene class. The window class is mostly a wrapper of the sf::RenderWindow that implements the main methods of this class. Meanwhile, the scene class is the one in charge of storing the diverse entities that the UI will create and is able to save them and load them, to accomplish this purpose it uses the library JSON for Modern C++ created by Niels Lohmann. In short, the scene class is in charge to store the status of the graphical elements in it and is able to load them.

It's also important to notice that the application implements the use of native windows for Windows. This is done through the library tiny file dialogs created by Guilleme Vareille.

Finally it has a game class, that stores the main functionality of the game loop: (process input, update and render). Additionally, it uses two singletons as support. These singletons are a Game Manager, in charge of storing different status and variables of the editor. The other singleton is a Pool manager, that is the one in charge of giving the graphical entities to the game when he needs it.

Data structure

The data structures can be found at the doc folder, it has been created with Doxygen.

Strategies and Solutions to the Development Problems

First of all, to create the different graphical entities we decided to use SFML, thus allowing us to create the multiple entities using it's primitives. Is important to note that the project is heavily dependant with SFML. These classes are created through a POO hierarchy using inheritance. To the basic class model defined we have added a Drawable_Entity with the objective of group the basic operations of the different graphical entities. The reason behind not adding this functionalities to entity is to not have transformations and rendering methods linked to entity, as this is the core class of the hierarchy and it's possible that in the future we would like to create new entities not related to render ones, like code scripts. Furthermore this class could help in the future to virtualize the different entities with a common interface and improve the different data structures we use.

All drawable entities can have transformations (rotation, scale, translation), we implement them using the sf:Transform class from SFML. This class works directly with matrix consequently allowing us to set different origin points for every kind of transform, note that this would be impossible with the basic SFML functions (like Rotate) of shapes and others.

This graphical entities also use the design pattern of factories to manage their creation and establish a maximum of every kind of graphical entity that can't be exceeded. To accomplish this the base constructor and copy constructor of these classes have been privatized. Making it mandatory to pass through the factory to create new elements.

The UI is implemented through IMGUI, and is integrated at the game class. It has it's own functionalities of update and render, this making it easier to separate through edit mode and game mode in the main loop. The UI allows to create, delete and edit the different graphical entities in the scene. We will further develop the UI at the user manual section.

For the creation, reuse and manipulate of these graphical entities in the scene specific structures, using as base some defaults of the standard C++11, have been created. To manage the creation, reuse and final liberation of the graphical entities we use the pattern of Pooling. Our Pool is a singleton that has an std::vector for every kind of entity that behaves like a stack, it adds and removes elements at the end thus making a more efficient result. This Pool is in charge of creating the different entities and the game class will communicate with it whenever it needs a new instance of any of them. The pool will initialize a base set of entities, not being used, and will return one of them whenever it's possible. If by the other hand, the pool doesn't have more items ready to be used it will create one using the factory. When items are returned to the Pool they are reset and ready to use again whenever someone else asks for a new item.

To store the different graphical entities in the scene we use std::unordered_maps that allows us to implement a hash functionality to access the different entities faster. This map uses the id of the entity as its hash, the entity class assures that two entity won't have the same id.This unordered maps are used to fast access the entities hence the reason why it's an unordered map, the order doesn't matter.

We have another map that is in charge of drawing the scene and it's implemented through a std::map. The hash used in this case are the different z-orders that exist in the scene. Note that in this case we implement a map because the order matters as we wish to draw first the elements with a lower z-order. Every position of the map also stores an unordered map that contains all the elements that share that z-order.

Meanwhile, to accomplish the save and load of the scene, the JSON for Modern C++ library has been used. Using it to serialize the status of the scene we want to save and load the json of a saved scene to be able to set it to it's correct state. To save and load the json files the library tiny file dialogs has been used. It allows us to invoke, select a file of the type we have indicated and returns us it's path.

Next, the implementation of the window class is a wrapper of the sf:RenderWindow. Implementing commonly used functions like draw, display, close… It's also important to notice that this class also works with a factory.

For the implementation of the Game manager we have opted for a singleton, as it is remarked. This singleton follows the singleton pattern, it uses a getInstance function that will create the game manager if it doesn't already exists or will return a pointer to itself, the instance, if it exists. This class is mostly used by the game to store status and other useful data.

At last, we have the game class. This class is the one in charge of iterating the main loop. The primary functions of a game loop have been clearly separated in functions for the game mode and edition mode. Therefore making it easier to separate between game mode and edition mode.

Personal Tasks and Workgroup Plan

To divide the work between the two members of the team we have implemented agile methodology, SCRUM to be more accurate. The kanbas used to monitorize the workflow can be found at: Scrum Kanbas.

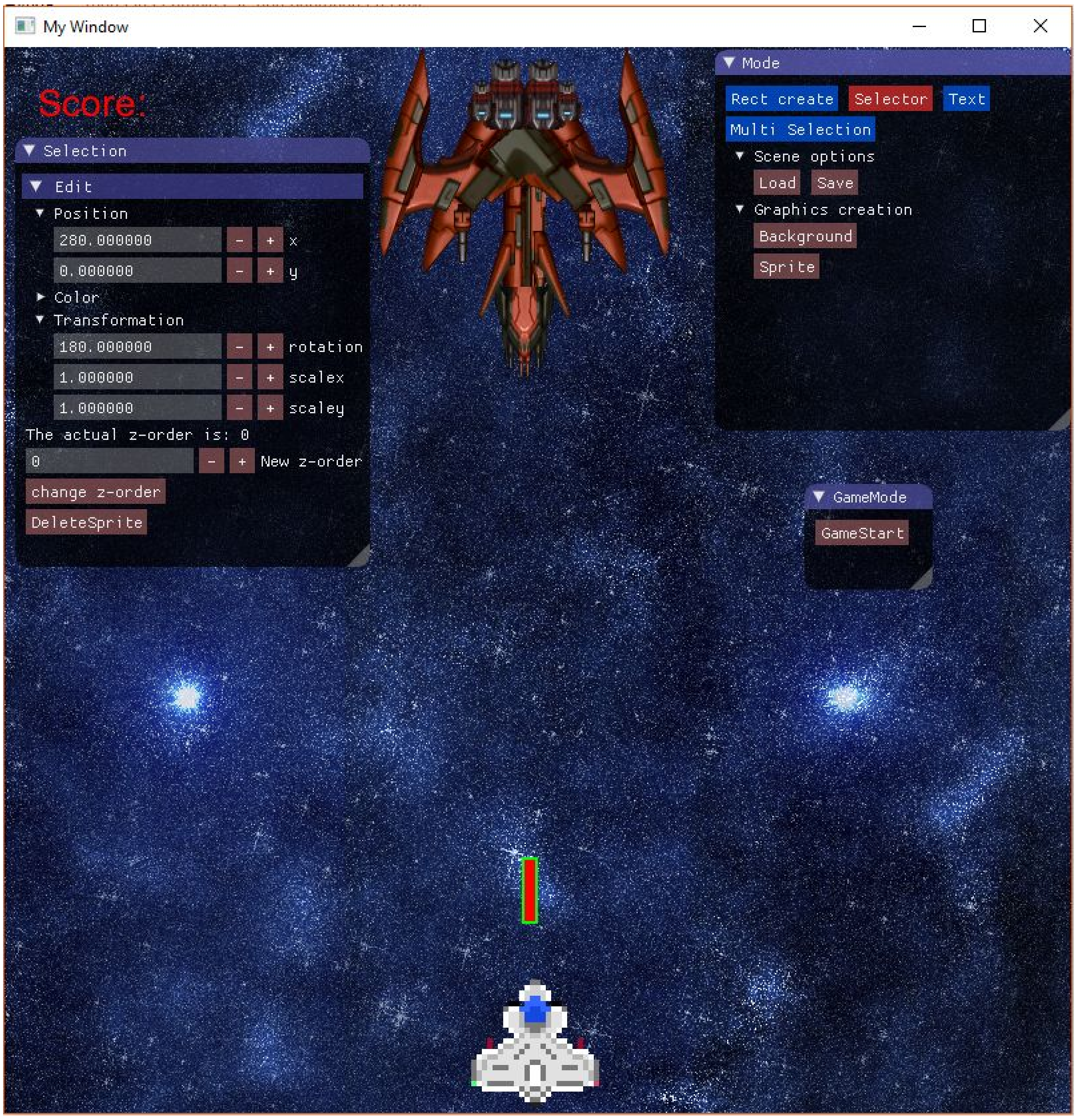
To summarize the distribution of the multiple tasks has been:

Sebastián Adrover Pedrosa : Task 1, Task 4,Task 5, M1(Especial efficiency through mapping for selecting entities and draw with z-Order using json library) and M2 (implementation of native windows through tinyfiledialogs)
José María Martínez Carvajal: Task 2, Task 3, Task 6, M1 (efficient creation of graphical entities through Factories and Pooling) and M2 (alternative grahpic library use with SFML).

Short User Manual

First of all to create the executable file two bat files can be found at the build file. compilacion_d.bat in case you wish to compile in debug mode and compilacion_release.bat for release mode. In case they are not present the following dll must accompany the exe file: sfml-graphics-2.dll, sfml-system-2.dll and sfml-window-2.dll. These dll can be found at the 2d_engine\deps\SFML\bin path.

Once we execute the exe we will see the main window of the application. It’s important to notice that the UI can be altered as you wish, let’s start seeing the example of an scene:



280.000000 x
0.000000 y
180.000000 rotation
1.000000 scalex
1.000000 scaley
The actual z-order is: 0
0 New z-order

As can be observed at the image the UI consists of three menus. Let's start with the mode menu.

In this menu we can choose in which mode we are at the moment, and the selected mode button will be highlighted as red. The different modes are:

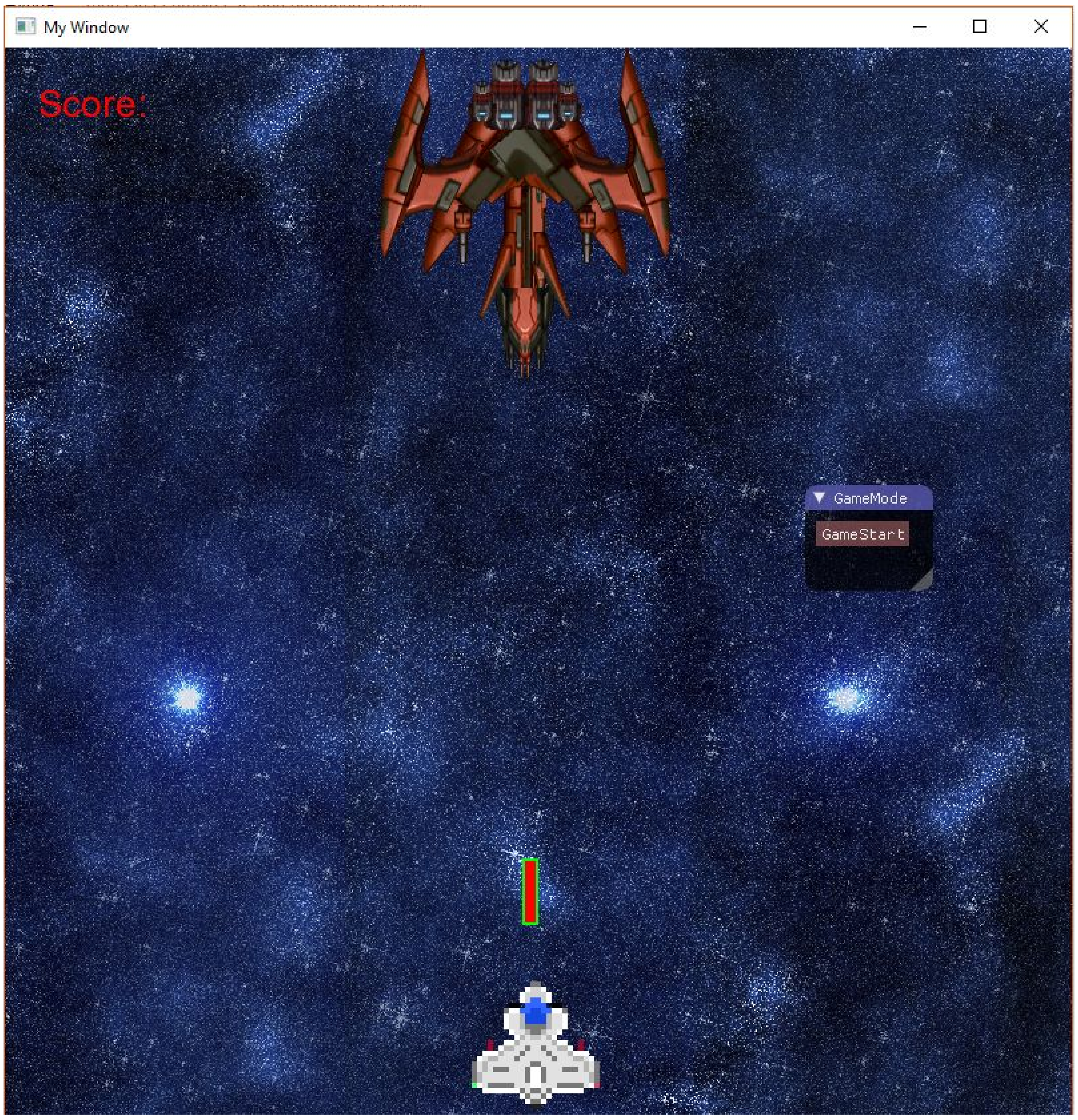
- Rect create mode: in this mode we can use the mouse to create rects in the scree. Once we press the button with this mode active it will start to create the rect. This will continue until we release the mouse button.
- Selector mode: in this mode we can select an element from the screen. Once we select an entity it's information will appear at the edit mode menu, we will check it further lately.

- Text mode: with this mode we can create default strings, that will say “Hello world” at the position of the screen we click.
- The multi selection mode lets us select more than one entity and move them using the edit menu, we will make more clarifications later as this is mode is still in beta and can be a little tricky.

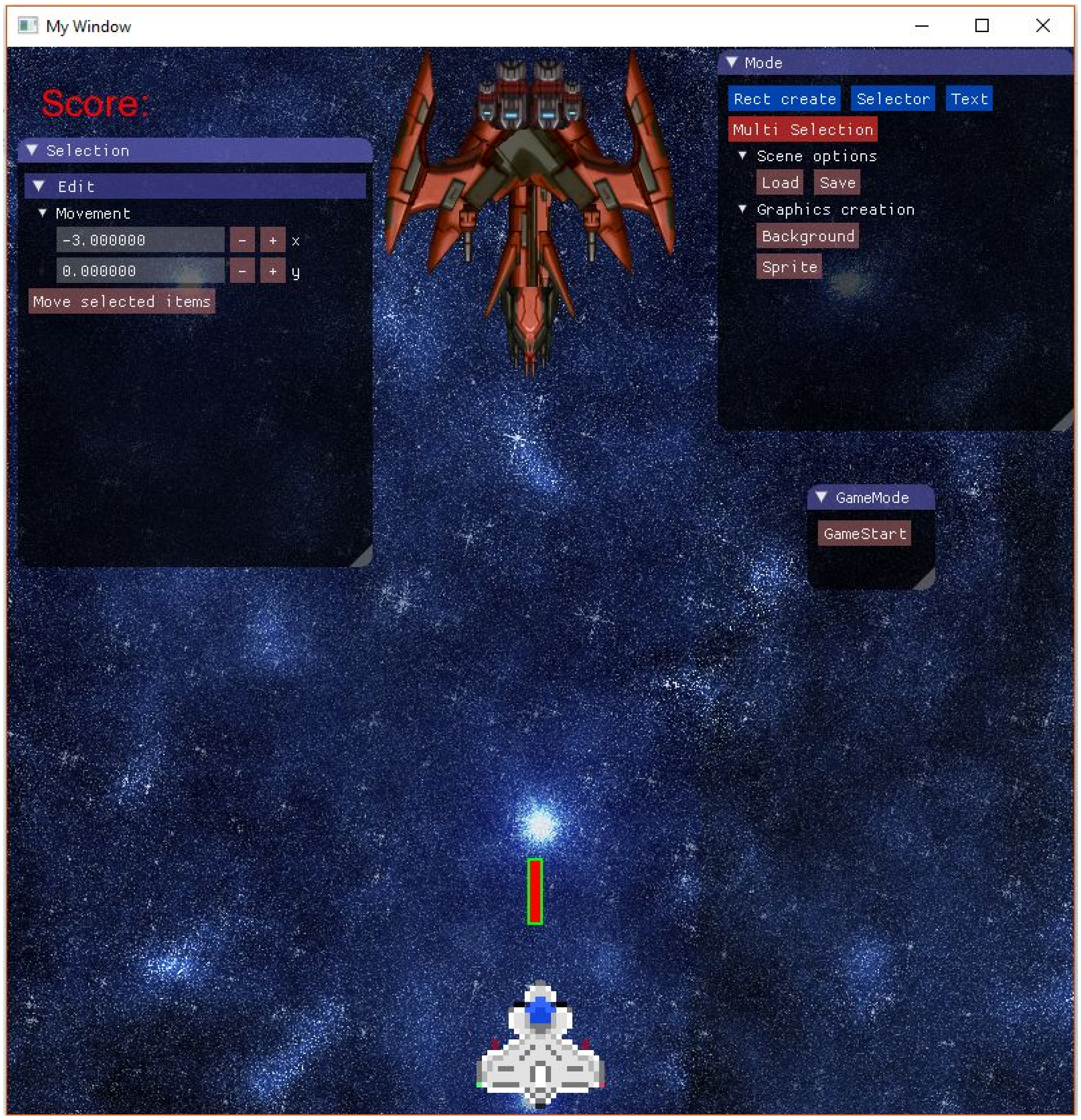
The mode menu has also some utility functions like saving and loading the scene, in json format and creating backgrounds and sprites. Note that the sprites will be created at the top-left corner of the screen.

The following menu we will check is the selection one. In this menu depending of the entity we have selected, or if we are in multi selection mode, the information will vary. Here we can modify all the values of the entities until we are satisfied. Some examples of common things we can change are the positions, transformations, color…
It’s also possible to change the z-order of the entities. This z-order is used to indicate a depth sensation and this way we indicate which entity has to be drawn over which. Note that for making the change effective is necessary to press the button change z-order. At last we can find the delete button. With this one we can delete an entity and remove it from the scene.

To end we have the The game mode menu. If we press the button Game Start we will activate the logic of the game (thus exiting edit mode). This menu of the UI will be the only one showing in play mode. Right now this button doesn’t change it’s color while active like the ones at the mode menu. The reason is because it will be pretty clear when we are in game mode as half of the UI will disappear as you can observe at the next image:



As a last note, let’s talk a bit about multi select mode.



When we are at multi selection mode the selection menu will change to a movement operation. While we are in this mode we can select multiple items and operate with them independently if they are labels, sprites or rectangles (note that the background can't be selected as moving its position would be strange). Clicking on them again will deselect them, but it's not clearly seen at the screen. Adjustments are being taken to improve this mode and made it more intuitive with the time.