### Game Around UI Documentation Structure

**Author**: Timóteo Cruz

#### 1. ****Introduction****

* **Project Overview**: This is a small test to the Game Around showing how: I code, use the new UI Toolkit and folder structure.
* **Purpose of the Document**: This is to speak about some points that could be obscure during the developers test evaluation.

#### 2. ****UI Architecture Overview****

* **UI System Structure**: The UI is divide in 3 parts:

- Main UI the leaderboard itself that holds all the UIs.

- The custom button that when pressed will fill the leaderboard list.

- The card that will be instatiated inside the leaderboard list.

* **Key UI Components**:

- The custom button was created as a class because it would hold some data and a delegate reference that will be used when the button is pressed.

- The card is a simple UXML but in the future It could become a Custom card if some action was need, like in the custom button.

#### 3. ****UI Design and Implementation****

* **Design Guidelines**: It just follow the design that was provided.
* **Prefabs and Assets**: We have 3 UXML files: LeaderboardTemplate, ButtonTemplate and CardTemplate.  
  We also have the corresponding USS styles to each one.
* **Script Overview**:
  + **USS files:** Those files have comments to divide the USS so it can be read from top to bottom and make the changes easy in a long run.
  + **UXML files:** Those files have names and classes that reflect a good structure to make changes easy in the future.
  + **C# files:** All scripts have good names, methods names and strategical comments to help the interpretation of each file. **OBS.:** Scripts that start with SO are scriptable objects, it make easy to flow in the project.
* **Responsive Design**: The design is 99% responsive it is missing this when using the scroll from the listView component.

#### 4. ****UI Navigation****

* **Event System**: The UI is using the **obsever** pattern using delegates when fetching data.

#### 5. ****Good Practices Implemented****

* **Coding Standards**:
  + I follow the Unity standad **naming convenstions**.
  + Use **scriptable objects** to make dependency injection and avoid using **singletons** or even the **obsever** pattern too much.
  + Use the **single responsibility principle** from **SOLID** to make each class easy to read and **open** to new related functionality.
* **Reusability**: Using scriptable objects make the code more testable and we can easy reuse it in other scripts.
* **Accessibility**: Using atlas to make the project to use less memory.

#### 6. ****Challenges and Solutions****

* **Challenges Encountered**:
  + As UI Toolkit is new I do not implement sprites and even less using atlas, those were chanlenges that made me study the Unity documentation and make several try and error.
  + Implementing a Task to handle the requests. I just did this I few times and I am not used to it.
* **Solutions**:
  + In the end I’ve converted the code into a scriptable object to just put sprites into image visual element and also add those to my own documentation.
  + I also had develop a scriptable object just to handle this situations.

#### 7. ****Future Improvements****

* **Feature Wishlist**:
  + Add a popup with proper message to handle when the data fecth does not returns nothing or some error.
* **Refactoring Plans**:
  + Recreate the listview to make it more custom.
  + Add more fields to the **SOAddCardToListview** to make is more custom.
  + Also make it more flexible refactoring the code that handle image setup.