Movie Listing

# How to run

The entry point of the app is the scene called “*main”* inside the folder “*\_Scenes*”. The app is configurable by some *Scriptable Objects*. They are located inside the folder *“SO”* and referenced in the *GameObject “SceneController”.*

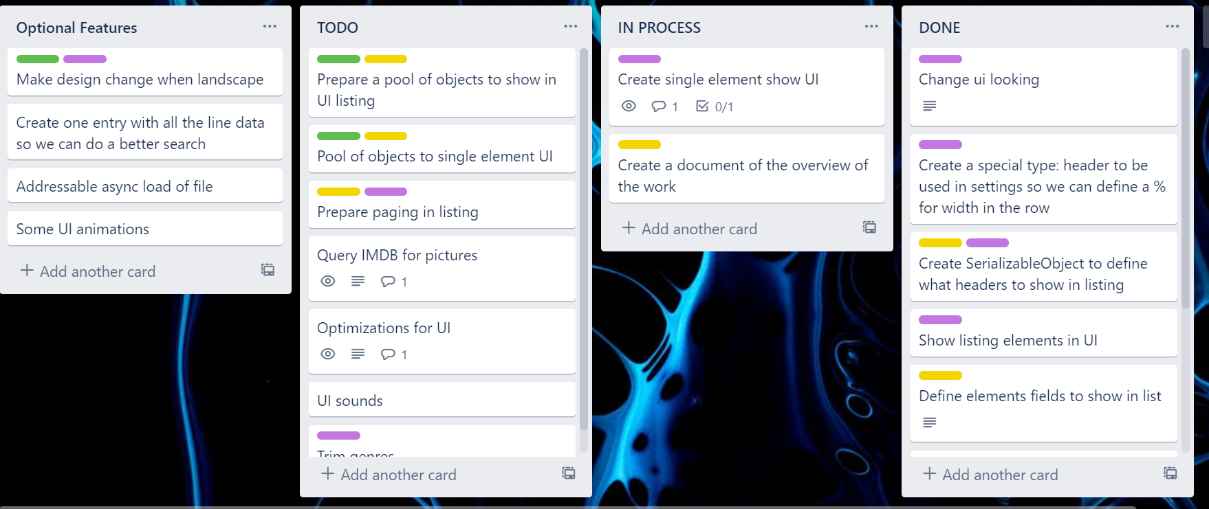
# Intro

The development process starts with a quick identification and overview of the needed pieces: a parser for the csv file and a UI that will show the entries. To keep this super simple, and because time constraints, when the app starts it parses the csv file and will show right away the listing of entries. The listing will be limited to a number so we can keep memory usage in optimal values, and the single element view is simplified with a possibility to check all parse data from that item.

The layout of the app is defined as portrait and the target resolution 1080x1920.

# Development process

For this kind of project, I always start with a quick draw of the overall architecture and trying to identify the pieces that will be needed. Right after that, the setup of the source control and a Kabang board like Trello (image 1).



(Image 1. Trello board)

After the initial setup of the board, I will start implementing the tasks, and as the development evolves identifying new parts or tasks and deciding priorities. The time constraint will force some tasks to not be done, so it is important to have a global picture of what I must implement so I can have a better decision on what to do.

I try always, when a part is not directly related to Unity, to write code independent of Unity (without deriving from MonoBehaviour). This it is easier to Unit test logic parts.

## Parts

1. Parser: Responsible for parse the CSV file. I used the Resources folder for simplicity and because time constraints I did not do an async load of the file (probably using *Addressables*), and for this small csv file that is not a big deal but in a real application it should be an async load.
2. Items Listing: For performance wise I decided to use pages to split the item listing (number of items per pages configurable in a *Scriptable Object*) and use a pool of those entries, so I can reuse the same *GameObjects* for each page. In this way I can reduce the amount of memory usage (by limiting the items), reduce the overhead of creating new *GameObjects* every page and the amount of garbage created by that process. The UI layout is basically a *ScrollRect* with the items listing and buttons to change pages. I have used Unity auto layout to control items placement. One of the UI optimizations that I had in mind, was to replace this auto layout by a script to place the items in the right place but I did not had time to create it.
3. Single element view: Simple UI panel that show an item data. I’ve use rest API from <www.themoviedb.org> to grab movies images to give some color to this panel. I’ve placed some elements that I consider more relevant in a different way, then in a *ScrollRect* (suffers the lack of the same optimization of items listing) I show all the data of an item.

The design of the app is, how can I say it? Hum, a bit poor, but I am not a good designer 😊. I tried to keep it simple and clean.

## Issues/Difficulties

The biggest issue here, is making an UI that works for multiple devices. I simple did not had the time to test in many devices. I had to decided that target resolution is 1080x1920 (only portrait), it will work of course in different resolutions but in extreme resolutions (smaller) it will not work.

## Things that would do if had more time

Because of the small amount of time to create this project I need to cut some features that I had planned, for example: remove auto layout from *ScrollRect*, UI sounds, landscape design, UI animations, search field, async load of csv file.

I consider development process, as an iterative process, so probably if I did some more iterations more things would change, for example, as I’m writing this document, I’m seeing that *BulkInfoPool* and *EntriesPool* share enough to do a refactor to create an abstract class.