Ayberk Çam
Unity Engineer Assignment Report

General Information

The project was developed primarily by planning to ensure a comfortable transition between two different mechanics. First of all, after the desired mechanic was developed, other mechanics were also developed to indicate how the transition would be made. In terms of technical design, the first difference between them is the mold used to create the elements. In the main mechanic, elements were created by moving around the borders of the matrix, while in the second mechanic, elements were created by walking around the matrix in a way that I call a snake. The second difference between these two mechanics is that after winning the reward, the earned reward disappears or the row containing the earned reward disappears and is recreated. These two mechanics have been developed by technically allowing the renewal of the reward earned in the first mechanic. Technically, the second chance is the possibility of adjusting the number of elements created. The transition between these two mechanics and the adjustment of the number of elements can be adjusted with the component in the "StartManager" object in the starting scene.

To differentiate the roulette mechanics, the mechanics of moving around random elements have been developed. While making this development, the elements were linked to each other using the Circular Linked List structure for a comfortable transition to the roulette mechanics in the examples. The rewards earned come from the structure created afterwards. If a new element is to be added, it is placed in the required position of the structure.

Although the rewards given are standard in this project, the project was developed with an open-ended design using the Interface structure for situations where the reward is not only for the wallet but also for different scenarios. This situation may arise again in future scenarios, such as free rights etc. It also allowed rewards to be added easily. At the same time, a different Interface structure was used for a comfortable transition when different visual effects were desired on the elements.

Local save was used in the project. The project checks the existing save file every time it is opened. Afterwards, the wallet status is updated.

Challenges

The first challenge encountered while developing the project was to make the design open to further development. Within the scope of the project, the aim was to build a structure open to changes, rather than just a development that won awards. For this reason, it was decided to use two different Interface structures. In this way, the project was developed open to improvements.

The second difficulty encountered was the use of Addressables. The difficulty in this matter was not how to use it, but when to arrange the necessary things to be loaded asynchronously with this structure. This situation was solved by loading the necessary assets where necessary and then performing other operations.

The third difficulty encountered was while developing a random roulette mechanic. The difficulty I encountered here was that the last visited element was not selected again when selecting a random element. The difficulty in this case was solved by creating a list, randomly shuffling this list using the Fisher–Yates Shuffle algorithm, and then selecting the elements one by one.

The fourth difficulty encountered was what happens when new elements are wanted to be added to the game and there is no equivalent on the wallet side. This situation was solved by checking new elements and synchronizing Scriptable Objects every time the project was started.

Design Patterns

First of all, a combination of Factory Design Pattern and Object Pooler was used to ensure good performance and to connect object creation to a single location. These structures were used for two different types of elements. One is for the created reward elements and the second is for the flying reward images.

Secondly, Singleton structure was used. Objects derived from a generic singleton structure do not disappear and code repetition is prevented with this generic structure. At the same time, a simple Singleton structure was used for objects that disappeared. This structure was used only where necessary and no improvements beyond its purpose were made.

Model View Presenter structure was used to independently update the improvements in the interface that the user accesses in a scalable manner. View structures are only in contact with the user. It only serves to ensure that what the user sees is updated. The view structure is updated from other places.

Finally, Observer Pattern was used for communication between objects. In this way, development was made without the objects being dependent on each other and a clean code was created.