Gram Games – Game Developer Take-Home Technical Test

Solutions by Eduard Lavrishchev

## 1. Adding the Unity project to a git repository.

I have previously worked with version control software for my personal, academic and commercial projects, I had no problems setting up a new repository on my GitHub. One important thing to note, I always use Unity. gitignore so that unnecessary files are not uploaded.

The project was initially made with an older version of Unity that I have on my computer. The guidelines suggest that one can upgrade the project instead of downloading the old Unity version. That's what I did. When I opened the project for a first time, I had a bunch of errors in the Unity Editor itself, as well as in my IDE (I use Rider). Clearly there was something wrong with the imports. So, after a bit of researching, I found out that UIElements in not under Experimental namespace anymore. I fixed the imports and project loaded without any errors.

## 2. Fix all compiler warnings.

There were two warnings. The first one said that there are .meta files for directories that do not exist in the project. Unity fixed this warning itself by creating these directories.

The second warning was about an unused variable quantityFound in ItemUtils class. After deleting the variable, the warning disappears.

## 3. Refactoring the Game class into a singleton.

Singleton is a programming pattern that allows only one instance of a class. This is quite a useful pattern when it comes to game managers. I used a private static variable, a public getter, and singleton initialization in Awake method.

DishEffect and GridCell are two methods that uses Game class. So, I changed the reference to Game class from expensive FindComponentOfType to the static instance of Game.

**4. Fixing the UI canvas.**

Changing canvas's render mode to World Space fixed the issue of scaling icons.

**5. Pattern control.**

This one was fun, I really enjoyed doing this bit. I went with ScriptableObject implementation because it is an agile and versatile way to manage custom data types. They are easy to edit, easy to swap them to test different variable values.

Item density is a number between 0 and 100 that determines the chance of each GridCell spawning an item. Recipe range is an array of NodeContainers, which is a data structure for recipes. A designer can easily add and remove recipes in the array.

Basically, spawning of items happens in 4 steps:

1. First, you have a difficulty parameter in Game class, that determines how many recipes are spawned to a player per grid. Let's assume the difficulty is 2, that means we need to get 2 random recipes from the Recipe range (and also make sure that these 2 random recipes are not the same recipe).
2. Then, for each grid cell, you use Item density parameter to determine whether the cell will be filled or empty.
3. If filled, we need to choose a random recipe from our 2 recipes
4. Finally, we choose a random ingredient from the recipe.

**6. Main Menu.**

I have created a separate scene with a background and a canvas. There is a button on the canvas that says 'play' and when you press it, it changes the scene. I needed to add both game and main menu scenes to the build settings, give main menu an index of 0 (so that it is the first scene that loads up when the game is launched), and game scene an index of 1. The ‘play’ button executes a method ‘SceneManager.LoadScene(sceneNumber)’ with sceneNumber being 1. The ’back’ button on the game scene is there to go to the main menu, it executes the same method but with sceneNumber of 0.

**7. Settings**

By the time that I am writing this, I have realised that I mistakenly added a settings menu to the main menu scene, not the game scene. I will update GitHub by the time you received this document. The idea stays the same – there is a volume slider that changes the value ‘volume’ in PlayerPrefs (so that it persists the value between sessions) and changes the value of ‘volume’ in the Audio Source component. It also updates the text component that displays the current value to the user. The actual value is from 0 to 1, but the player sees it as from 0 to 100 for better readability.

**8. Build-time compiler errors**

Editor scripts are not available in builds at runtime, they should be put in a special Editor folder. Otherwise, they cause compiler errors.

**General feedback on the tasks.**

I think the tasks perfectly show the daily routine of a Unity game developer. The bugs, the missing features, etc. are a huge part of the development process. As a programmer with some experience in commercial products, I didn’t find the tasks particularly challenging (thanks to a good project codebase), however I still learnt quite a lot, especially about how Gram Games manages data in the app – I specifically found NodeContainers very fascinating. I never really used raw GUIDs in data structures before, let alone in ScriptableObjects, and I will research more into that.

Thank you for this amazing opportunity to check out and work on the project made by Gram Games employees. After this task, I even more look forward to becoming one myself!