Contents

General Specification

Unity version: 2021.3.18f1

Target Platform: PC

Executing the Game: The game should be started (played) from the “MainScene”.

Inputs

Rotate Left: A

Rotate Right: D

Accelerate: W

Shoot: Space

Third Parties

Following third-party packages are used.

DOTween:

A tweening library used for simple animations

TextMeshPo:

Used for text components.

Internal Packages

I have used some packages that I have developed personally.

Service Locating:

A simple implementation of the Service Locator pattern. It is used to access some global entities, especially between scene transitions.

UI Management:

A simple package for handling UI windows and popups.

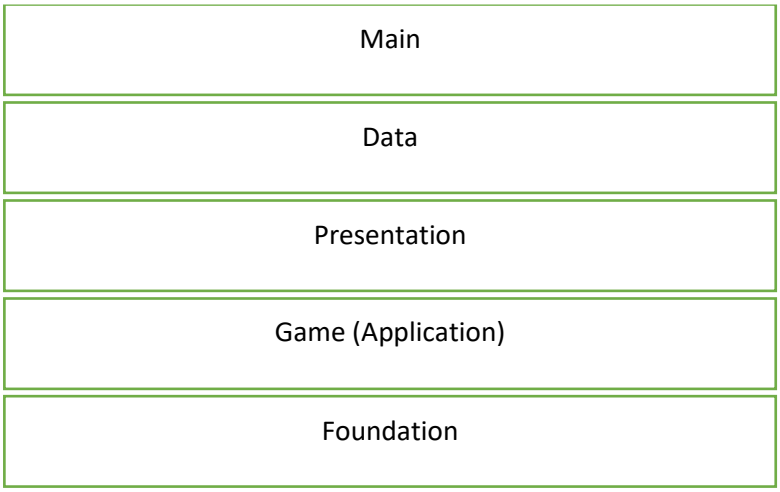
Package Basics:

A package that has some shared utilities for other packages

General Technical Architecture

Generally, the source code has two types of partitioning. The first type is feature-based.

First, modules are defined by features and inside the modules, the partitioning is based on technical concerns, mainly the layered architecture used. It is generally defined by the following layers (some modules may not have all the layers).



Ideally, I would have followed this layering but due to lack of time and some behavioral interconnection with Unity, I wasn't able to fully follow this architecture. You can see volitions when classes of the Presentation namespace are used in classes of the Game namespace.

Also, due to a lack of time, there is no data layer for persistence and configuration.

Game Structure

Overall Structure

The overall game structure is as follows. *GameManager* is the root of the game and mostly coordinates the different game states by creating the *MainGameController*. MainGameControllers has the responsibility of controlling a specific game state. Here we have `MainMenuMainController`, `LevelMainController`.

Level/Gameplay State

`LevelMainController` is the root controller of this state. It mostly has the responsibility of initializing and wiring the other sub-controllers. It also handles scoring logic and spaceship lives (which can later be moved to other sub-controllers).

This state is divided into the following sections.

Spaceship

This section is about handling the spaceship. It only contains SpaceshipAvatar which handles the physics, input, presentation, and logic of the spaceship (ideally I would have separate classes for each responsibility)

Asteroid

The two classes here are AsteroidAvatar and AsteroidsController. AsteroidAvatar is similar to SpaceshipAvatar and manages different concerns of an asteroid. AsteroidsController tracks how many asteroids are in the level and handles the splitting of an asteroid into smaller asteroids.

Ending

This is about the end condition of the level. EndConditionController checks the two end conditions of the level, namely, losing all lives and destroying all asteroids. LevelResultPopup is just a simple popup for showing the end of the level.

Screen

ScreenWrapper handles a simple way of wrapping the Spaceship and Asteroids when they go out of the screen.

OutOfBoundObjectDestroyer is just a simple component to destroy some objects when they are out of the screen, mainly bullets.

Shooting

BulletAvatar handles different aspects of a bullet, and BulletSpawner handles the spawning of bullets initiated by Spaceship.

Scene Structure

There are 3 important scenes in the game.

The MainScene is used for the root initialization of the game. This consists of creating and initializing the UIManager and GameManager. After the first initialization, it will transition to the MainMenuScene.

The MainMenuScene is a simple main menu and pressing Start will load the LevelScene.

And the LevelScene is for the level/gameplay state. It contains the environment for the level, consisting of Asteroids, Spaceship, Screen handling, UI, and other level related objects.

Design Notes and Decisions

Architecture and Testing

I'm an advocate of good architecture and testing. I start the project with the intention of separating the concerns, to enable testing. But due to a lack of time and not having a specific way to achieve this goal in the context of physic-based object, I decided to not focus on this goal. You may see some odd naming like `Avatar` which originally meant to separate the presentational aspect from the logical aspects.

General Structure

The general structure of *GameManager* and *MainGameController* may seem overengineered. But I had this structure prepared (from my other projects) so it was easier to reuse.

Gameplay Simplifications

I decided to make some simplifications to the gameplay. Mainly I did not implement the saucers and I defined only one level.

Spaceship Handling

For simplicity, I used RigidBody2D to implement the behavior of the spaceship, since a lot of features are provided out of the box, like torque and force.

Configuration and Persistent Data

Due to lack of time, I haven't implemented any data-related aspects. The main part is the configuration, which for simplicity I either use some hardcoded values or just expose them through presentational components. Usually, I would have a unified configuration system.

Also for simplicity, no Persistent Data behavior is implemented (e.g. for leaderboard).

Screen Wrapping

For screen wrapping, I used a very simple solution. The wrapper just used a simple Box Collider 2D for the screen size, and will "teleport" objects to the opposite side, when objects move out of the collider. There is a known bug with this approach, which is when new smaller asteroids are spawned right outside of the collider, which will not be detected. I used an ad-hoc solution by creating a second larger one to capture this "rogue asteroid".

One big problem with this approach is the jarring teleportation issue, which I manage to cover with some visual trickery. But a better solution is needed to handle the visual aspects of this wrapping correctly.

Note also that currently, the setup for the screen (the wrappers and visual aspects) is pretty manual, but it's possible to make most of it automated based on just a desired screen size.