SE450 - Object-Oriented Software Development

Final Project

2017-3-19

Project Report
Asteroid Game

Shuonan Yang

Background and Object

The goal of this project is to use object-oriented software development skills learned in class to develop a simple Asteroid Game.

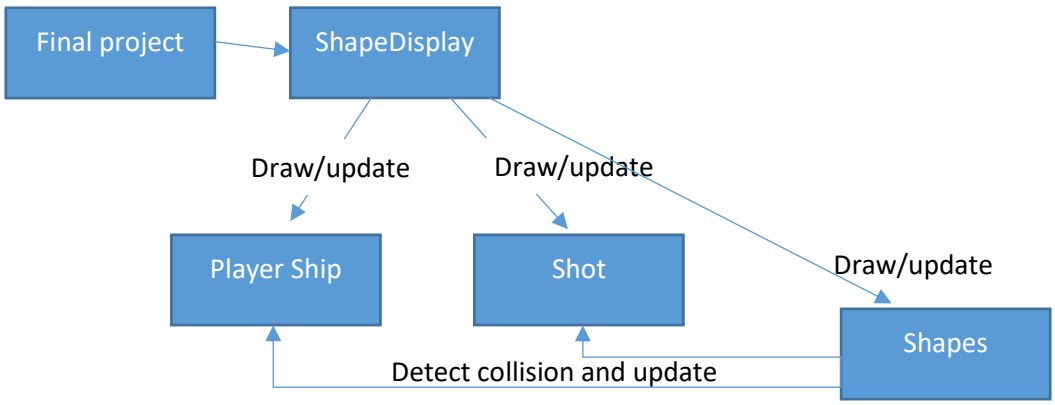
The game area will have floating shapes which generated from 'json' file. The player can use keyboard to control a space ship and the ship can fire shots. If the shots hit shapes, the shape will explode. If the shape hit shapes, the player ship explodes. A score panel will count the overall score for player. Each player has 3 ship life.

Discussion of the solution

This Game logic is simple. But due the design patterns, we need to plan the overall structure first. We need a class run the game as whole and many other parts to complete it. I think the game as several important parts:

1. Main – hold game UI setup and run the whole game
2. Keyboards – implement key control and pass to other class
3. Sounds – import and use when game status changes
4. Shapes – automatically generated from 'json' file
5. PlayerShip – the user contorted object which can fire shot
6. Shot – the object fire from player ship and hit shapes
7. ShapeDisplay – holds all objects and update their status

Conceptual diagram would like this (for detail see UML at end):



I managed to fulfill most of the requirements.

Configuration part:

I use *ConfigurationParser* to parse configuration from json file to the data type we need. *CondigurationManager* set all parameter into our models. Especially for player ship control: rotation, thrust, dimension…and shots control: life time, shot speed… I heavily import configuration data to PlayerShip to setup configuration.

Shapes part:

I use ShapeParse to parse json data to java recognizable data type. And then use Shape factory automatically generate shape type and dimension based on their own data (For a better play experience, I change all shape types to filled color circles. They are more popup and nicer on screen).

In the Shape.json I add one more String parameter- shape size: “larger”, “median” or “small”. When large shape be hit, it will split to 3 median shapes and new split shapes will add to shape array list and display on game panel. Same logic, when median shape be hit by shot, 2 small shapes created, original median shape was removed from shape array list. Same thing, small shape will be remove from shape array list (disappeared in panel) if hit by shot.

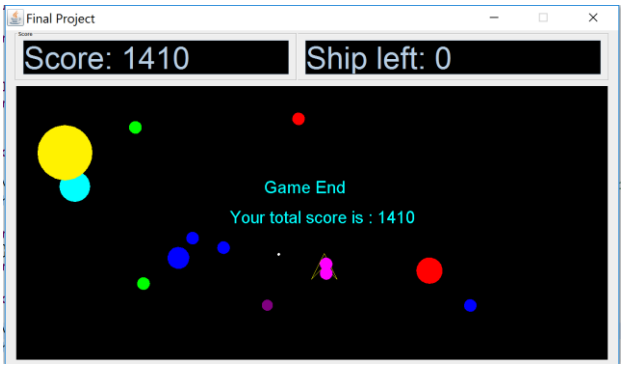
Hit test:

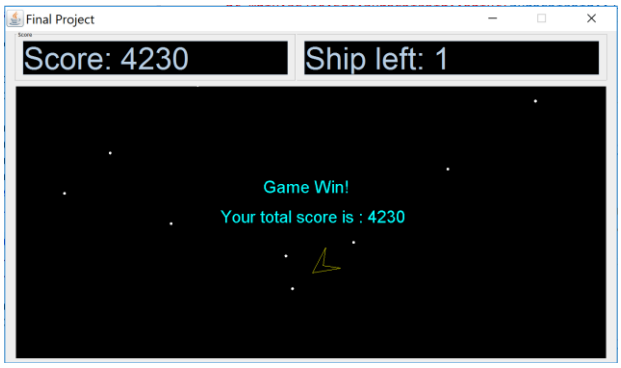
Instead of bounding box collision test, I use shape distance to test if 2 shapes are overlapped to each other. If the distance between a shot and a shape is smaller than the shape radius, we can consider the shape is hit by shot. The shape collision test return true. Same logic for ship and shape collision test.

Player Ship part:

Each player has 3 life. Each time the ship hit by shape, it will lose 1 life until 0. If no life left the game will stop. “game over” and total score will appear on screen. If all shapes are killed by the player, the “Game win!” and total score will appear on screen. Also the victory sound will play.

Player ship alsp implement all control movement and move algreithm: thrust, left, friction and so on.





Discussion of patterns

The mainly these 4 OOP Patterns:

Factory:

I use shape factory pattern to generate different shapes base on their own character. All shape has different shape, color, dimention, split number and so on. With factory pattern we don't need to create shape one by one. Each shape can figure out their own peoperty by using factory pattern.

Strategy:

Strategy pattern allow class algorithm or behavious changing at run time. For our case, the Strategy factory let different shape figure out their own strategy. If it is rebound strategy, when create this shape, the shape will apply rebound stategy. If it is passthru strategy, shape will apply passthru stategy.

Singelton:

Singelton pattern will make sure only one object get created each time. In my project, multipul objects are create almost at same time. Using singelton pattern, shapes are maintained in ShapeList and shots are maintained in ShotList class. Each time we are get access to single object and don't need to instantiate the object of the class.

Observer:

Observer pattern are used to solve one to mant relationship between objects. In my project I use observer pattern to notify multipul objects to change status or do something. For example, Motion class, not only shape but also all shots need to move on the screen. Whenever we need to stop the game, we will need to notify the obsevering objects to stop motion. Whenever the palyer push fire button, the ship start, because it start to observing fire mathod.

UML Class Diagram