Balloon Shooting project

Project Introduction

Introduction

The purpose of this project is to create a graphic game of balloon shooting.

Scope

At the current scope, it is a simple single player game that requires user to move the cannon and shoot all the balloons that are moving randomly on the screen. The player is considered to won the game if they can shoot all the balloons using no more than 5 bullets.

The intended audience is the developers and the lecturers who are going to evaluate the project.

Design Goals

In the game development, the requirements for the game may change throughout the process, or some functionalities are needed to be change or added into the game. Moreover, as the current scope is only limited to single-player, single-level game, future development can include the extension of the game's scope. Taking these into consideration, the following principles guide the design of the project:

Maintainability:

With requirements that were often in flux, it is important to be able to easily change functionality that had already been written without disturbing other related components. Therefore, all major components of the game such as the player and game object generation should be separated into different module (such as each class should be saved into a single **.cs** file)

Moreover, the classes in the project should follow the dependency inversion principle, which means they should be dependent on abstract classes and interfaces.

Optimized Performance:

The game should be able to update and render with the frame rate of 60 times per second. Therefore, the costly components are only run when necessary, and the methods applied are needed to be as efficient as possible to provide best experience.

Developer Friendly:

To make other developer who wish to review the code without difficulties, the program should follow the basic coding style and naming conventions.

Game Design Specification

Technology used

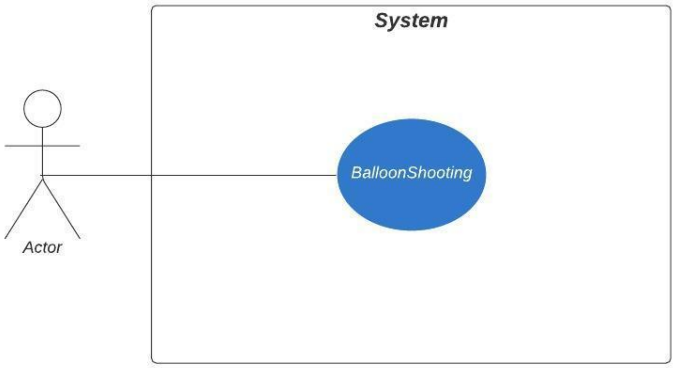
The game was built using Visual Studio, with the source code written in C#. For graphic integration, the third- party library Splashkit is used.

Graphic resource

The text font, background image, and the sprites for all game object are downloaded and edited from the Internet.

UML Representation

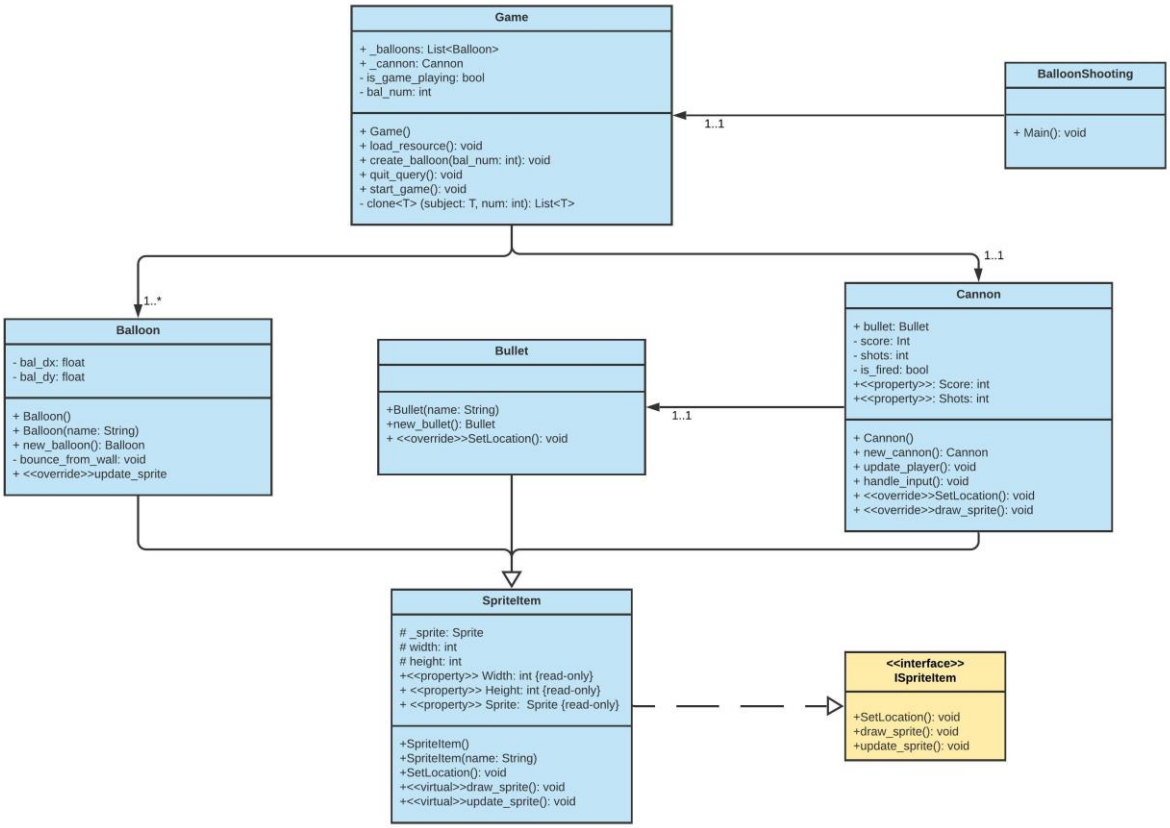
Use-case diagram



BalloonShooting (Game)
1. Player start the game
2. The system load the resource and generate a graphic window
2a. The system generates balloons that move randomly
2b. The system allow user to move the cannon vertically and shoot bullets
3. The player finish the game (either shoot all the balloons or run out of bullet)
2c. The system displays the game over screen when all balloons are shot or no bullet left.
2d. The system asks the player whether to replay or quit the game

To make the game easier to maintain and develop in the future, the project has implemented the Facade design pattern. It can be seen through the UML Class-diagram that all the user interaction with the system is through the **Game** class. Using facade pattern not only simplify the interface and make the system easy to use, but also minimize the dependencies on the subsystem, which is in line with the dependency-inversion principle that the project wishes to pursue.

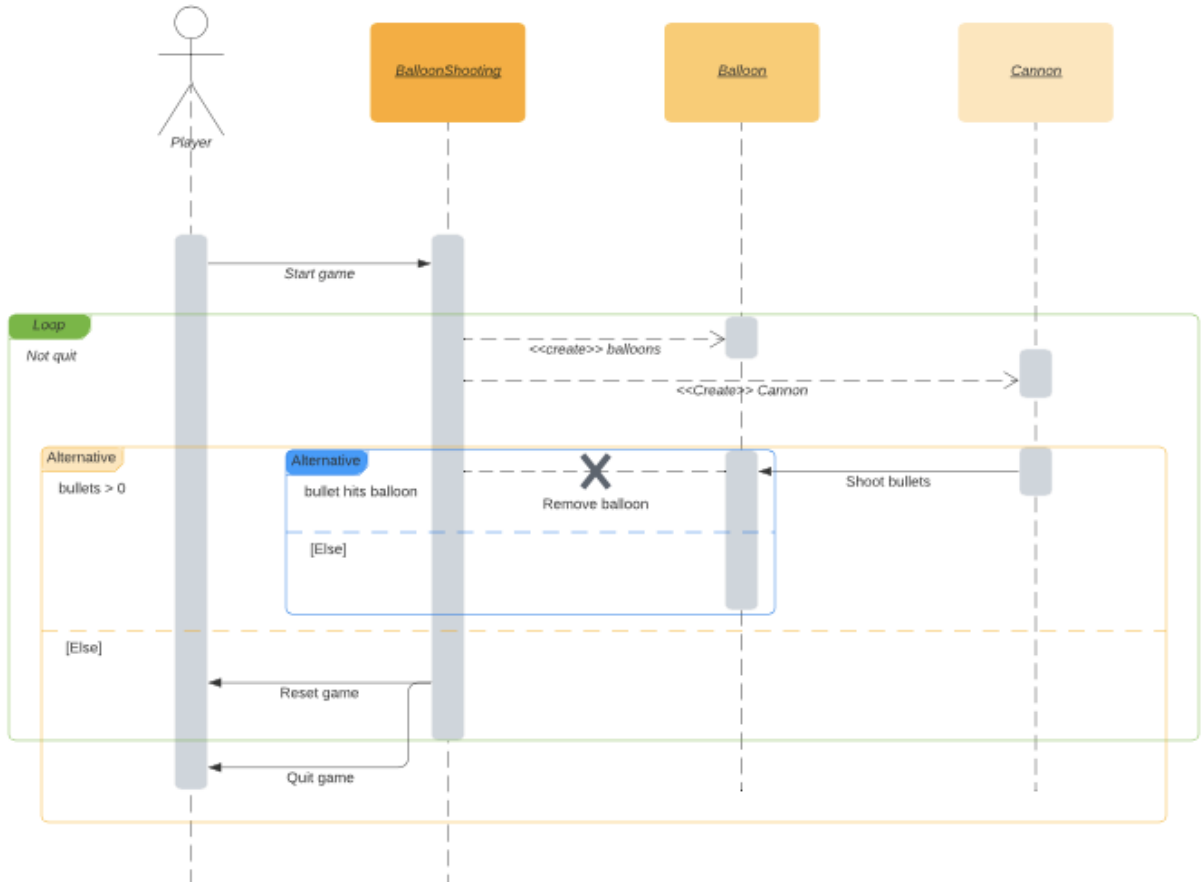
Class-diagram



Class's identification and purpose:

- **SpriteItem**: this is an abstract class that inherit from the **ISpriteItem** interface. It acts as the base class for all the items in the game. This class contains all the essential properties and methods that all other classes needed.
- **Balloon**: A class that focus on the "target" object of the game. This class contains the method of generating the object, and the method to deal with the issues of object collide with the border of the game.
- **Bullet**: A class that focus on the "ammo" object of the game. Like the **Balloon** class, it contains the method of generating the object and positioning the sprite in the screen. Although the movement of this object is dependent on the **Cannon** object, it is still recommended to separate this into an independent class, so that future maintenance or extension will be easier.
- **Cannon**: A class that focus "player" object of the game. Its task is to generate the object and move in accordance to the user input. It also responsible for calculating the score.
- **Game**: This class follows the facade pattern and acts as the unified interface of the project. Its purpose is to receive the input from **Balloon** and **Cannon** object and delegate required tasks to those classes.
- **BalloonShooting**: The main game of the project. It main tasks is to initiate the game using **Game** object.

Sequence diagram



Custom Program Design Specification

	Requirement description	Justification
X	Project proposal identifying what it will be doing, scope and the outcomes	Executive summary content
X	Identifying use-cases — building use-case diagrams	UML representation content
X	Identifying class-diagram (conceptual level)	
X	Building sequence diagrams	
X	Building class-diagram (specification level)	
X	Depiction of a) open-closed principle, b) dependency-inversion principle, c) Interface-segregation principle, or any other principles	The system strictly pursue the dependency-inversion and principle and open-closed principle, as demonstrated and explained in the design goals and UML representation content.
X	Implementation or usage of any Design pattern	In this project, the **Facade** design pattern is been implement and can be seen via the Game class
X	Justification of Design	Game design specification content
X	Implementation and Functionality achieved	The game is completely functional and the player can enjoy the game without any hinder.
X	Coding style and documentation	The program strictly follows the basic standard coding style and indentation
X	Interview with tutor	An interview with the tutor about the finished product has been conducted.
X	Use of anonymous functions, lambda expressions in code	In **Balloon.cs**, the lambda expression has been implemented in the form of function that calculate the movement of the generated balloon.
X	Demonstrating the use of Generics	In **Balloon.cs**, a function has been implemented in the form of function that calculate the movement of the generated balloon.
X	Video demonstration of the program that demonstrates its power to non-technical audience, something to post on Youtube	A video about the game demo was created. Access Link: https://video.deakin.edu.au/media/t/1_krrnw12s

Wow Factor:		
	Design of a new design pattern	
X	Implementation of highest quality and standards	The program is integrated with Splashkit for good graphic experience, and the program strictly follow the open-closed and dependency-inverse design principle