Software Design Document

for

Space Shooter

Team: Team Diwash

Project: Space Shooter

Team Members:
Peyton Urquhart
Connor Rosentrater
Diwash Biswa

Last Updated: [4/4/2021 8:10:33 PM]

Table of Contents

[Update the Table of Contents]

Table of Contents....2
Document Revision History....3
List of Figures....4
List of Tables....5
1. Introduction....6
1.1 Architectural Design Goals....6
2. Software Architecture....7
2.1 Overview....8
2.2 Subsystem Decomposition....8
3. Subsystem Services....11

Document Revision History

Revision Number	Revision Date	Description	Rationale
1.0	3/28/21	Initial Document	Initial Document
2.0	4/4/21	Class Changes	Changes to class structure

List of Figures

Figure #	Title	Page #
2-1	Entity Spawn Sequence Diagram	7
2-2	Player Movement and Shooting	7
2-3	Grunt Movement and Shooting	8
2-4	MVC Component Diagram	8
2-5	Factory Design Pattern	9
2-6	Observer Design Pattern	9
2-7	Singleton Design Pattern	10

List of Tables

Figure #	Title	Page #
3-1	View Subsystem Table	11
3-2	Model Subsystem Table	11
3-3	Controller Subsystem Table	11

1. Introduction

The product is a bullet hell shooter spaceship game. The game will be customizable using a json script to allow for customization of levels.

1.1 Architectural Design Goals

The design qualities that we had in mind while we were designing this project were usability and modifiability. We designed the product with usability by making our user interface easy to use. The level design will also use json objects to allow for easy modification of the levels in the game. The program is designed for modifiability because the enemy entity is designed to allow for different movement classes to be added in and different bullet spawners to be used to allow the enemies to be modified to change the different levels. The program will be using json scripts to allow the different waves of the to be modified and changed easily.

2. Software Architecture

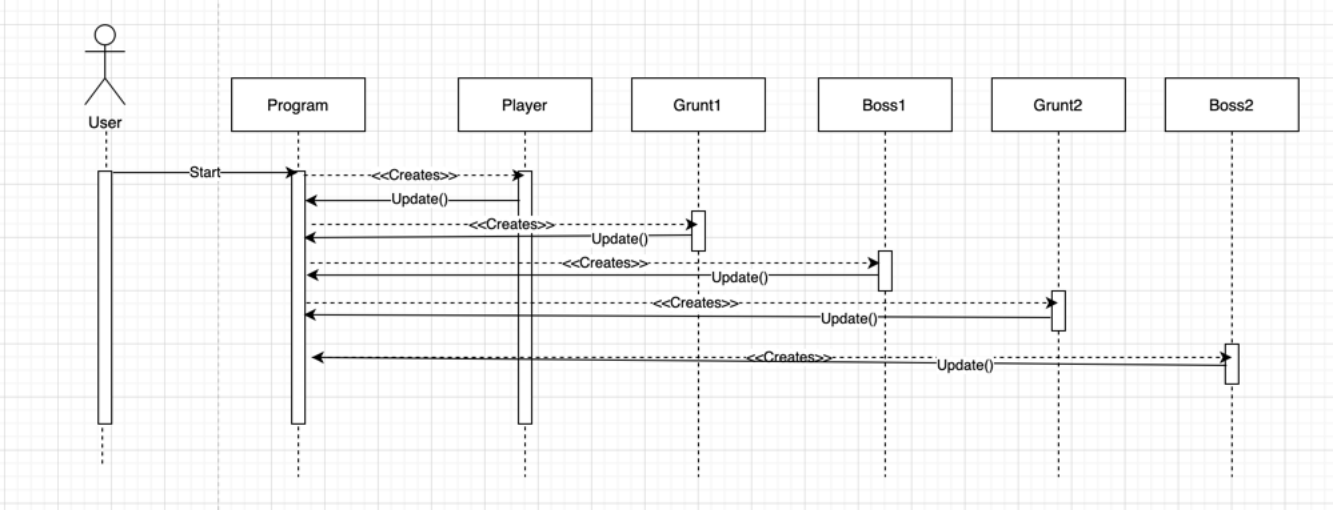

Figure 2-1 Models how the player and the enemies are created by the program. All the players and enemies inherit the sprite class and the sprite class's update function. This update function calls back to the main program and updates the location and status of each of the different sprites.

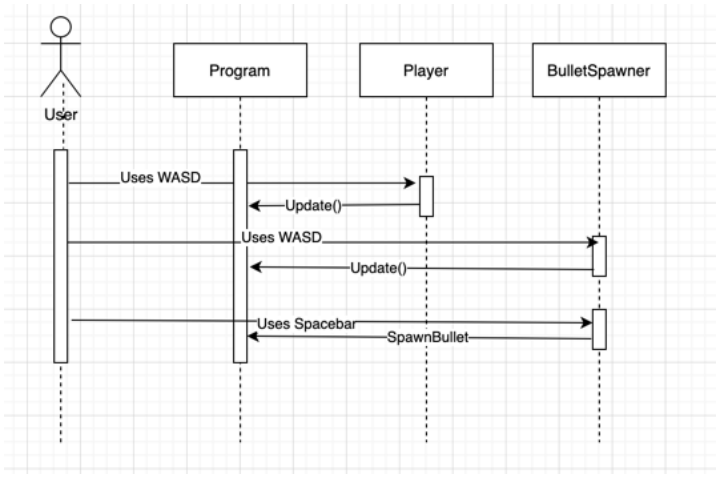

Figure 2-2 Models how the player and the bulletspawner for the player move and fire bullets. When the player hits the wasd keys it causes the player and the bulletspawner to both move so that they are always in the same location. When the user wants to fire a bullet for the player they hit spacebar which causes the bullet spawner to call the spawnbullet function.

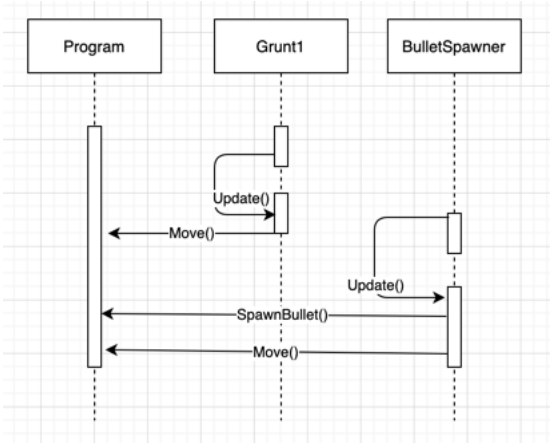

Figure 2-3 Models how the grunt1 enemy moves and fires. The grunt1 enemy has a bullet spawner that moves in the same pattern as the grunt1 does. This makes it so that the grunt1 and the bullet spawner are in the exact same place. When the update function is called the grunt1 moves and its bullet spawner moves and spawns a new bullet using spawnbullet.

2.1 Overview

We went with a Model View Controller architecture for our project. The model subsystem is comprised our sprite classes that make up the different enemies and the player class. The view subsystem is the MainGame of our project. The controller subsystem is made up of the CollisionObserver class. The view subsystem is responsible for displaying the sprites for the user. The controller subsystem is responsible for handling the collision of the bullets and damage. The model subsystem is responsible for all of the different sprites in the game and updating their movement to the view.

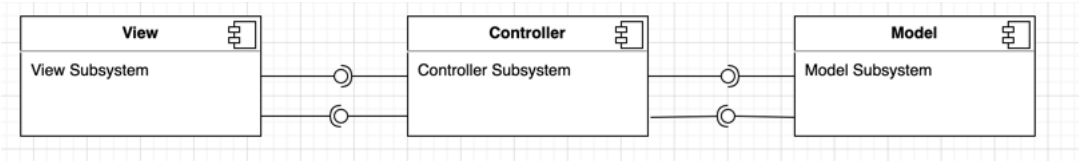

Figure 2-4

2.2 Subsystem Decomposition

2.2.1 View Subsystem

The view subsystem is the MainGame of our program that displays the game to the user. The view starts and creates the entities in the model subsystem using the controller subsystem.

2.2.2 Controller Subsystem

The controller subsystem is the CollisionObserver class that is responsible for finding when a bullet has collided with either an enemy or the player. This class is also responsible for causing the bullets to be disposed with once they have hit their target.

2.2.3 Model Subsystem

The model subsystem is the sprite class and all of the classes that are inherited by it. The model subsystem is responsible for the movement of the different sprites.

2.2.4 Design Patterns

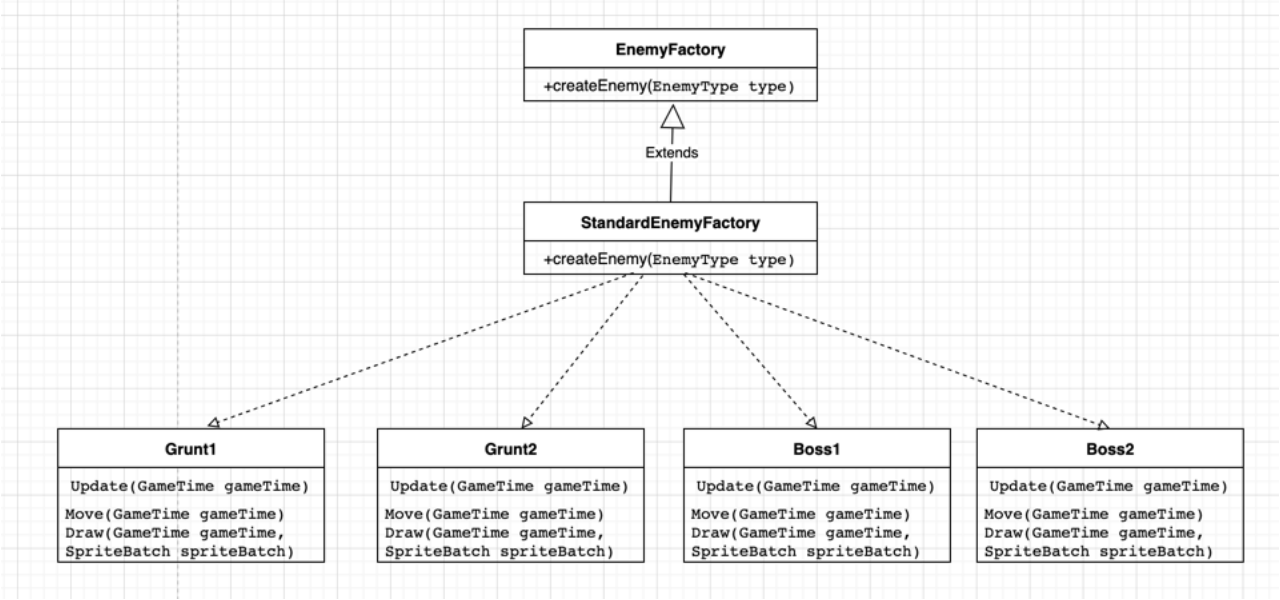

Figure 2-5 Shows how we use the abstract enemy factory to produce all of our enemy class sprites.

Figure 2-6 Shows that the CollisionObserver observes the Sprite class to see when it has collide with a bullet object.

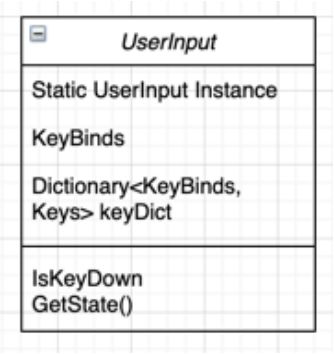

Figure 2-7 Shows the design pattern singleton for the UserInput Class. It has the static UserInput Instance.

3. Subsystem Services

Table 3-1 View Subsystem

Services Provided:	• N/A
Services Required:	• Sprite Update: This is used to update the view of the sprites when they move.

Table 3-2 Model Subsystem

Services Provided:	• Sprite: The sprite class is used for creating the player and all enemies. They are used for the movement of sprites. • BulletSpawner: This service is used to spawn bullet sprites. • Sprite Update: This service is used to move and fire bullets in the sprite class.
Services Required:	• Collide: This is used to detect when enemies and the player have been hit with a bullet.

Table 3-3 Controller Subsystem

Services Provided:	• Collide: This is used to detect when a bullet and another sprite have collided with each other.
Services Required:	• N/A