Software Design Document

for

Bullet Hell Shooting Game & Level Interpreter

Team: Team Reptile Ninjas

Project: BH-STG

Team Members:
Colin Willis
Miranda Mccoy
Tanner Dryden
Denish Oleke
Minh Nguyen

Last Updated: [3/28/2022]

Table of Contents

Document Revision History

Revision Number	Revision Date	Description	Rationale
1.0	3/28/2022	First draft	First draft of the design document.

List of Figures

1. Introduction

1.1 Architectural Design Goals

1.1.1 Performance

To improve the performance of the game, two tactics are available: “Control resource demand” and “Manage resources”. As the game progresses, there would eventually exist a large number of entities that need to be kept track of and used by different classes. To remedy this issue, the project will adopt the “Control resource demand” tactic. By utilizing the Singleton design pattern, classes would have one central entity storage to operate with, thereby reducing (or eliminating) the need of passing the entire entity list to certain methods and thus, increasing resource efficiency.

1.1.2 Testability

To improve the testability of the game, two tactics are available: “Control and observe system state” and “Limit complexity”. As there is a lot of interaction between objects of different classes (for example, updating the health of the enemy when the player hits it with a bullet), classes should avoid directly manipulating the attributes of other classes, in order to prevent unexpected behavior or errors from happening and making the testing process easier. The project will adopt the “Limit complexity” tactic. Using encapsulation, classes would keep their attributes private, and only make it accessible to external classes via their respective methods; this would help prevent unwanted changes that would otherwise make locating the error point difficult and thus, limiting the structural complexity of the project.

2. Software Architecture

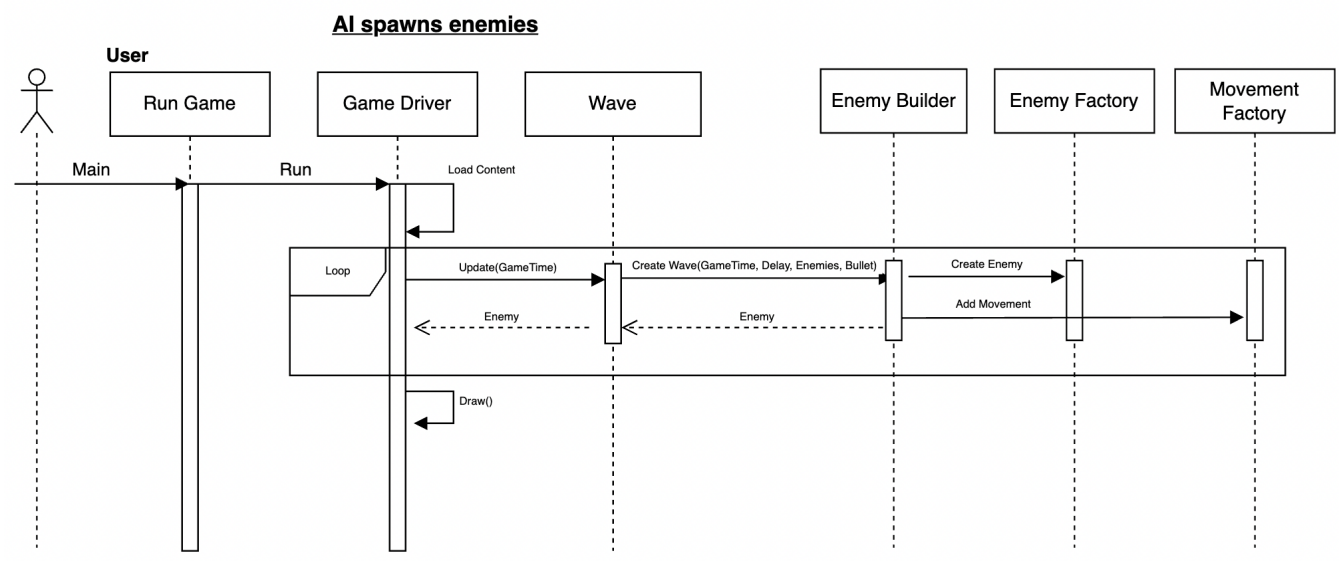


Figure 2.0.1: Enemy spawn

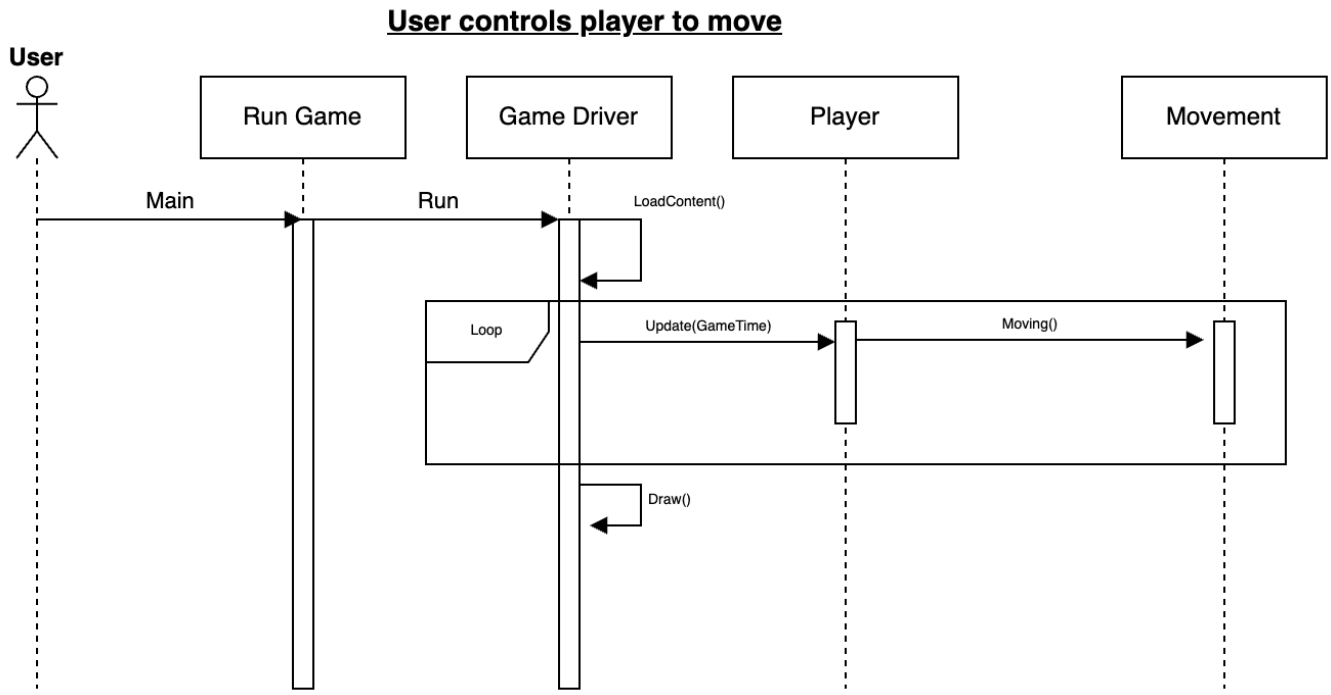


Figure 2.0.2: User moves player

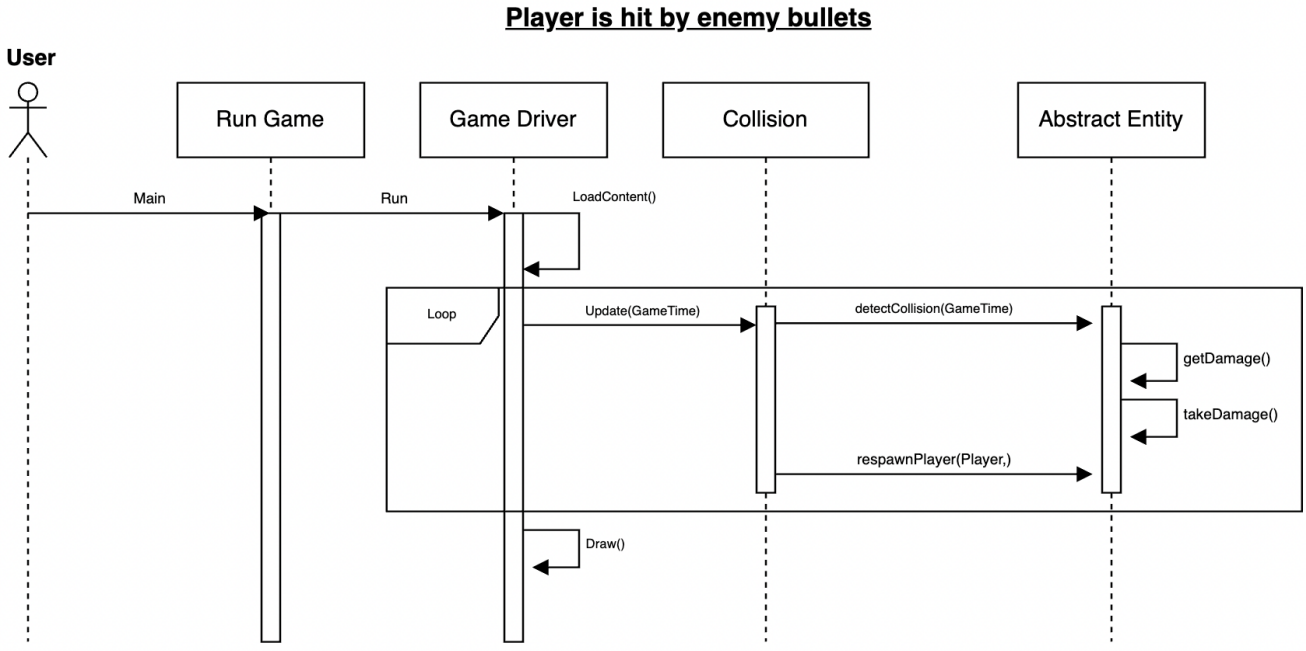


Figure 2.0.3: Player hit by bullet

2.1 Overview

The architectural pattern that was chosen for our system was the Model-View-Controller pattern. The model component consists of all the entities that we have created and utilized throughout the program. The controller classes manipulate and update model component attributes. The controller classes also update the presentation layer which includes where the entities are located on the screen.

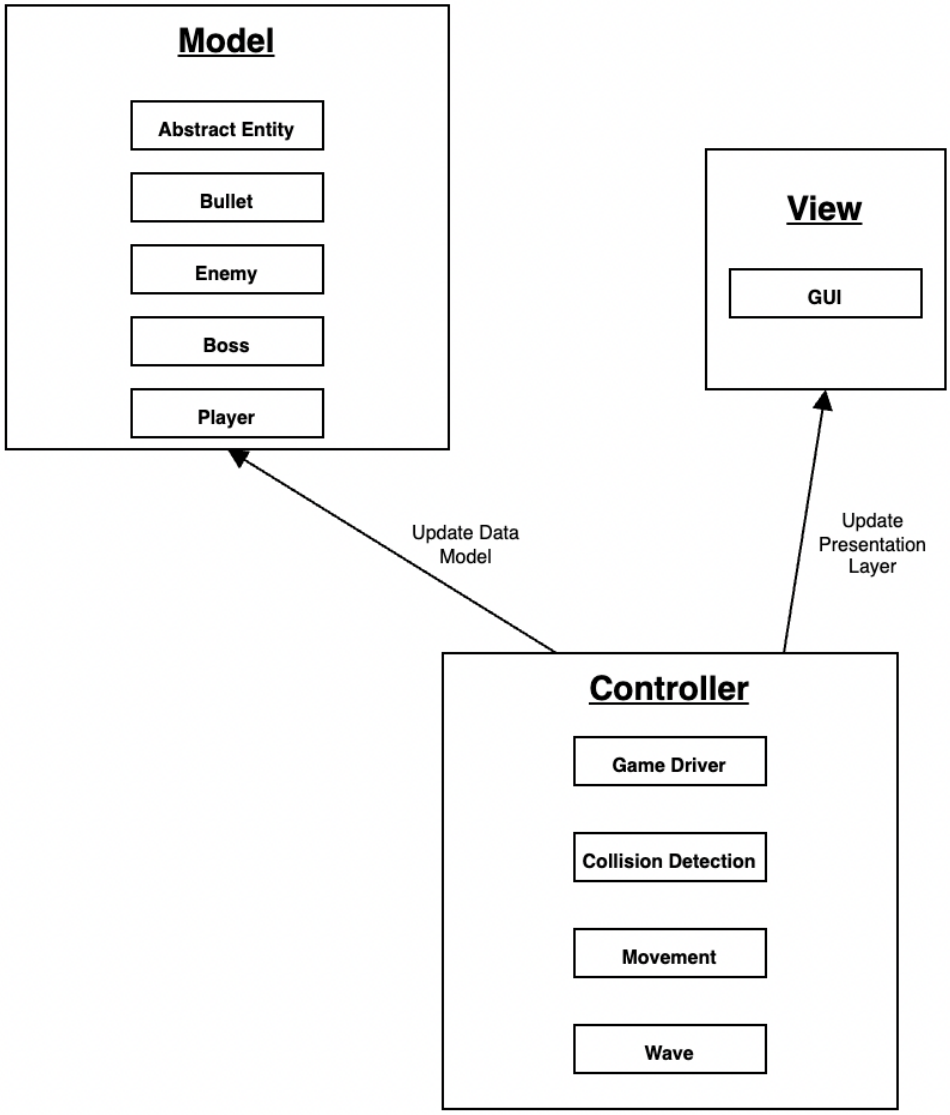


Figure 2.1: Model View Controller

2.2 Subsystem Decomposition

The Model subsection is in charge of modeling the data that we create and are storing. It takes in the command from the controller section that tells it to create the entity and it stores it in the entity dictionaries. The entity dictionaries update frequently with removing and adding entities and this how the controller tells the view what to display on the screen.

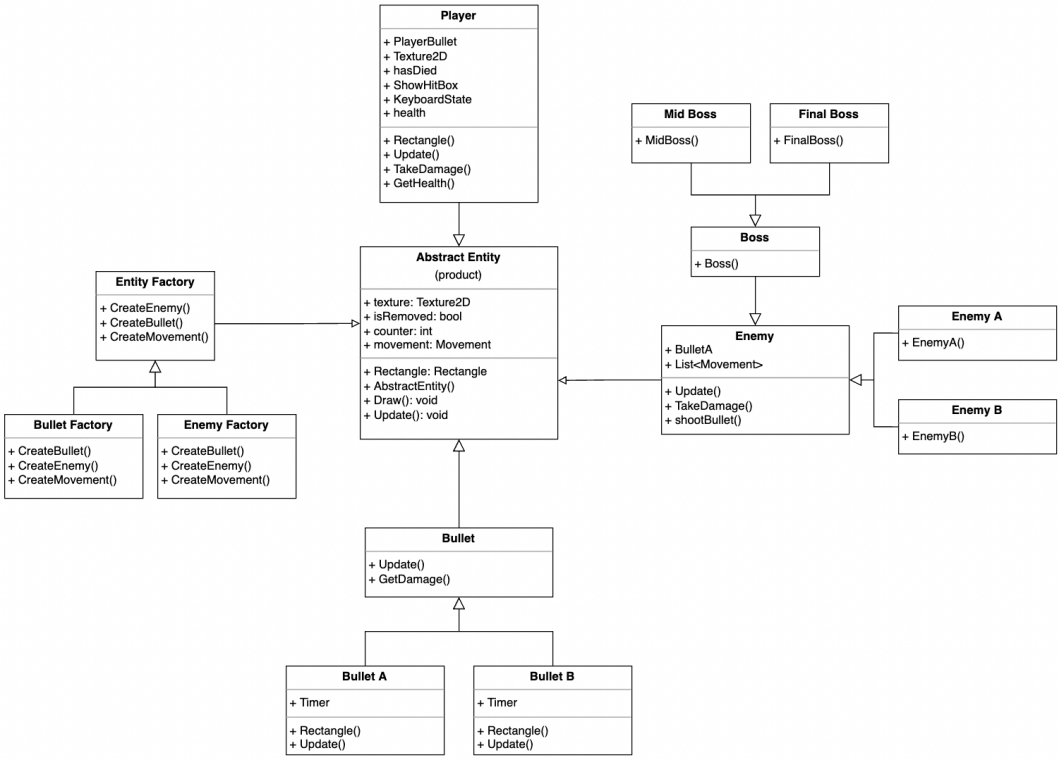


Figure 2.2.1: Entity Model

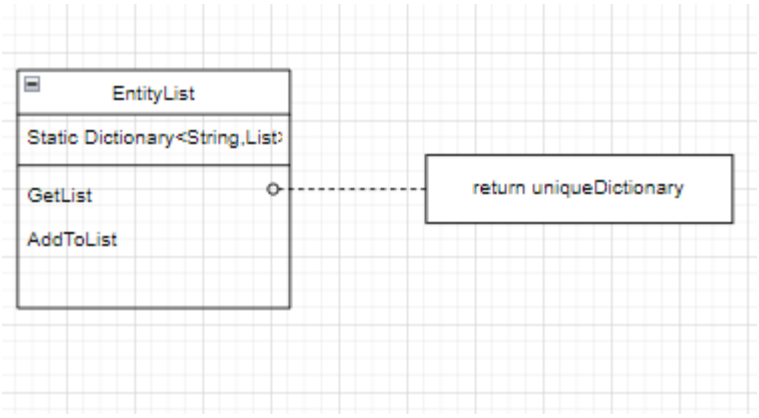


Figure 2.2.2: Singleton Model

The Wave class controls the enemies quantity, type of enemy and, when they will be on screen. This information is processed and updated in the EntityList class (Model) and then the game is updated to show the enemies on the screen (View)

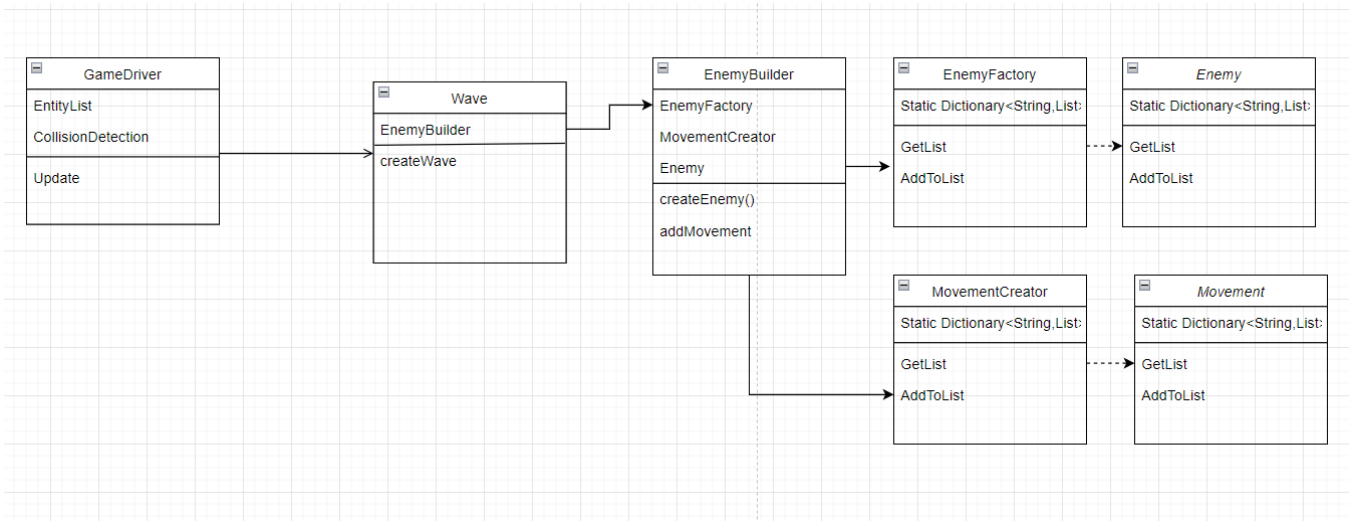


Figure 2.2.3: Wave Controller

The CollisionDetection is a controller class that receives information from EntityList (Model) which contains all the entities data like the enemies location, player location, and bullets location. If the different types pass through each other, then they will be removed from the EntityList and from the screen(View).

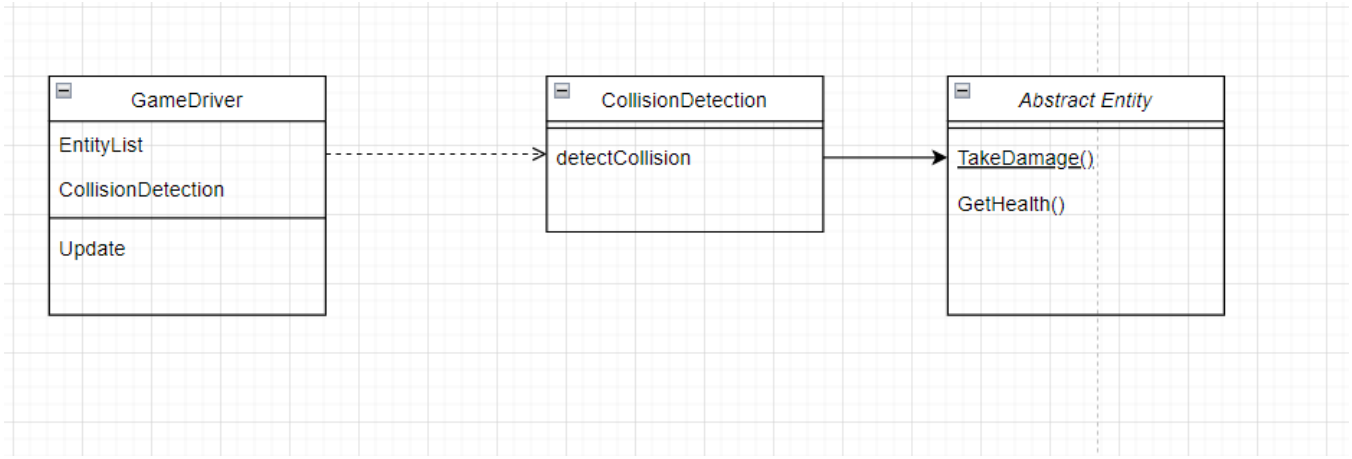


Figure 2.2.4: Collision Controller

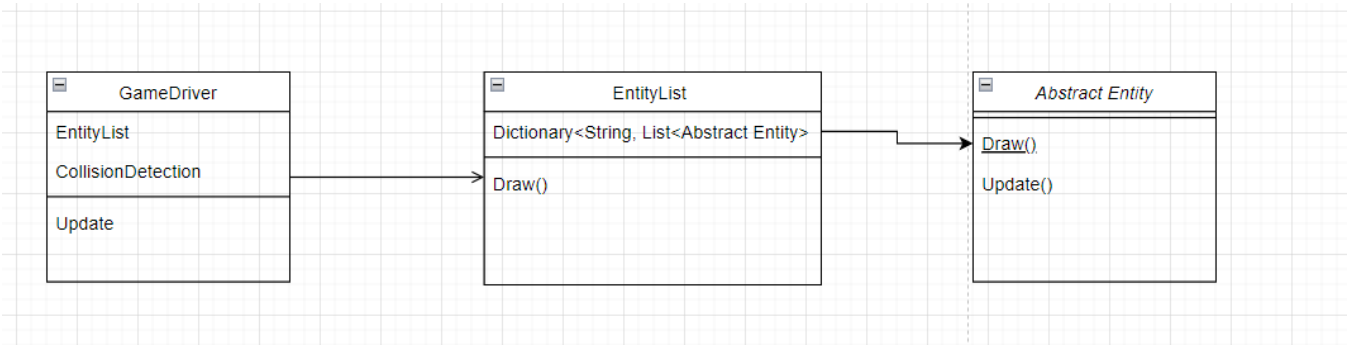


Figure 2.2.5: View

2.2.6 Design Patterns

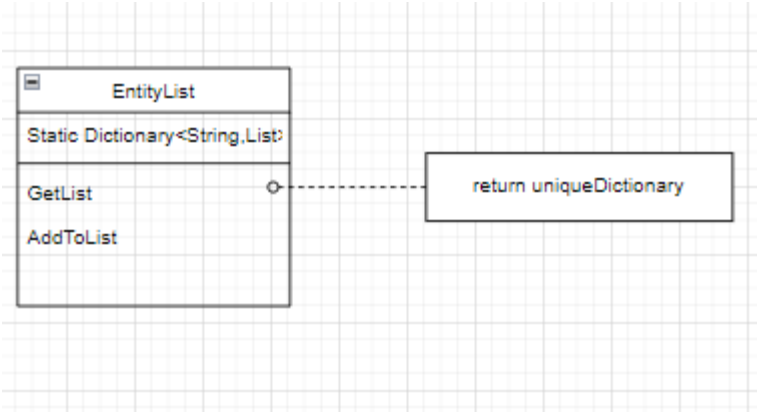


Figure 2.2.6.1: Singleton Pattern

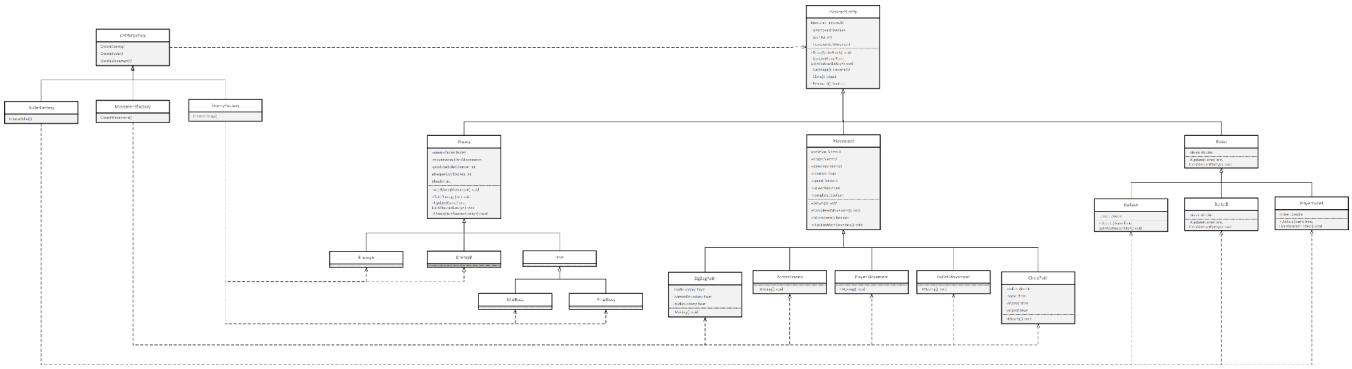
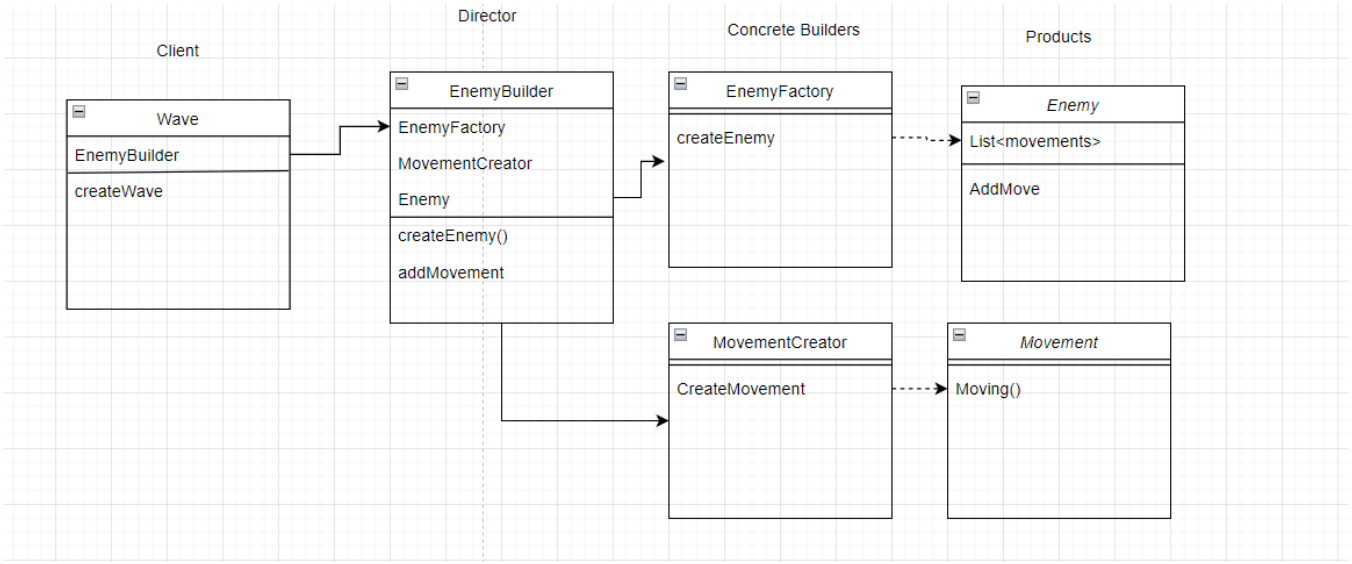


Figure 2.2.6.2: Factory Pattern



2.2.6.3: Builder Pattern

3. Subsystem Services

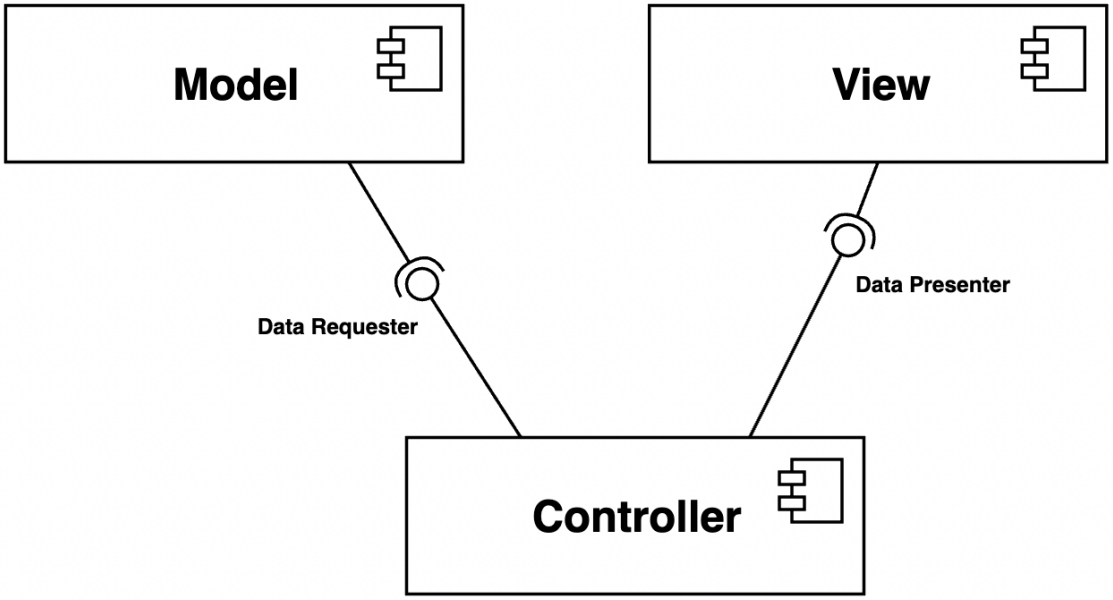


Figure 3.0.1 MVC

The first dependency in our model is between the Model and Controller subsystems. The service that is provided here is the Data Requester service. This requests the entity data that is needed to be created and returned. The next dependency is between the Controller and View subsystems. This service is the Data Presenter service. This service is for telling the view what to update on the screen.