Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Computación e Informática

CI-0136 Diseño de Software

Proyecto MARDA

Profesor:

Alan Calderón Castro

Estudiantes:

Daniela Quesada Aguilar B25247

Jennifer Villalobos Fernández B67751

Cristopher Ulloa Jiménez B16649

Adrián Vargas Martínez B57576

6 de diciembre, 2020

Justificación del diseño

El Marco Adaptable y Reutilizable para el Desarrollo de Aplicaciones que se ha creado, se ha basado en dos patrones de diseño: Fábrica Abstracta y método plantilla.

El patrón Fábrica Abstracta se considera adecuado para la creación de este MARDA debido a que dicho patrón proporciona una interfaz para crear familias de objetos sin especificar sus clases concretas y considerando que el problema que se busca resolver es el cómo ensamblar objetos compuestos con diferentes representaciones de manera que no haya que referenciar explícitamente a los distintos tipos de partes, fabrica abstracta es un patrón adecuado para la creación del MARDA y es el patrón principal utilizado en el diseño de este proyecto.

El segundo patrón utilizado corresponde al método plantilla. Este patrón ha sido utilizado en el método denominado *execute()* que se ubica en la clase padre controlador, clase de la cual los controladores específicos heredan y utilizan el método plantilla mencionado, esto debido a que todos los controladores específicos tienen la misma secuencia principal de ejecución(inicio del juego, durante el juego, ganó o perdió, fin y reinicio del juego). Este patrón resulta útil en la realización del MARDA debido a que se busca definir una plantilla sin usar la parametrización y además, es especialmente en *Python* (lenguaje usado en este proyecto) debido a que no existen mecanismos que permitan parametrizar las clases en dicho lenguaje.

El diagrama de clases final del MARDA se incluye en un archivo externo a este documento (imagen) debido a sus dimensiones, pero se recomienda primero visualizar el diagrama de la figura 1 de este documento para comprender con mayor claridad el diagrama final. Dicho diagrama (el de la figura 1) muestra las clases del proyecto, las relaciones de herencia, composición y asociaciones utilizadas, pero no se incluyen las propiedades y métodos de cada clase (dicho detalle se puede consultar en el diagrama final).

Ahora bien, es importante recordar que el patrón fábrica abstracta está compuesto por la fábrica abstracta, las fábricas concretas, los productos abstractos, los productos concretos y el cliente, los cuales están presentes en la figura 1 y se detallan a continuación:

1. Fábrica abstracta: clase *GameFactory*
2. Fábrica concreta: clase *BomberManFactory*
3. Productos abstractos: clases *Maze, Player, Weapons, Goals, Sounds.* Acá es importante recalcar que la función de la clase *Goals* es implementar metas u objetivos de los juegos, como por ejemplo, llaves y diamantes del juego *BomberMan*, cofre de *DigDug* o bandera de *Battle City*.

4. Productos concretos: *BomberManMaze, BomberManAvatar, BomberManEnemy, BomberManWeapons, BomberManGoals, BomberManSounds.*
5. Cliente: el cliente del patrón fábrica abstracta en este caso equivale a las clases controlador que se implementan(*Controller* y *BomberManController*), y además, es en los controladores en donde se implementa el patrón método plantilla, como se mencionó anteriormente.
6. Vistas: clases que se encargan de dibujar en pantalla lo que se puede crear mediante el MARDA. Dichas clases son *GameMenuView* y *GameView* para el MARDA y las vistas específicas de *BomberMan* son *GameViewBomberMan* y *GameMenuViewBomberMan.*

Es importante mencionar que las clases de color azul corresponden al MARDA y las de color blanco son las clases propias del juego *BomberMan*.

Una vez explicado el diseño que se ha implementado para la creación del MARDA y la del juego *BomberMan* con la figura 1, se puede observar el diagrama final (imagen externa a este documento) el cual incluye además las propiedades y métodos de las clases y a modo ejemplo, cómo deberían ser usadas las clases de otros juegos de tipo laberinto en el MARDA.

Principios SOLID

Single Responsibility:

En patrón fábrica abstracta: este principio se ve reflejado en el proyecto debido a que cada fábrica podría especializarse para ensamblar una única familia de componentes.

En patrón método plantilla: cada controlador concreto implementa los métodos invocados desde el método plantilla de forma específica.

Liskov Substitution:

En patrón fábrica abstracta: cada controlador puede funcionar igual con cualquiera de las fábricas concretas.

En patrón método plantilla: el controlador puede funcionar igual con cualquiera de sus clases derivadas para obtener diferentes implementaciones del método plantilla.

***Dependency inversion*:**

En patrón fábrica abstracta: el controlador padre puede seleccionar con cual tipo de componentes se debe ensamblar el objeto compuesto.

En patrón método plantilla: facilita que el controlador padre pueda “inyectar” una implementación específica del método plantilla en tiempo de ejecución.

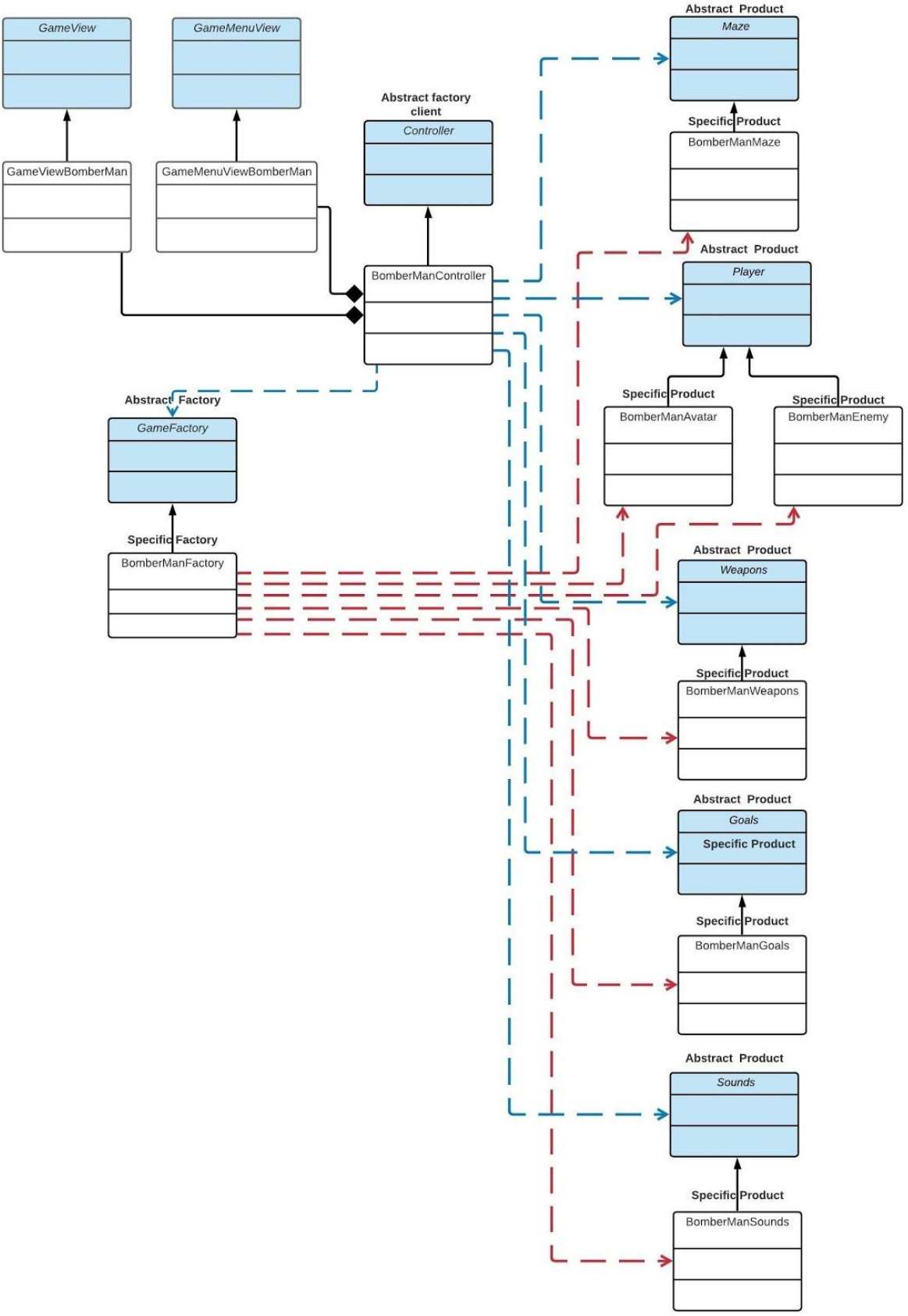

Figura 1: Diagrama del MARDA