Fall 2024

COMP 302

gameName

D2

Group: groupName

Enes Ak – 80090

Muhammed Babelli - 84342

Yusuf Cemâl Karataş - 83639

İbrahim Cebecioğlu - 79906

Caner Kösem - 80246

Cemal Nişan – 79158

Table of Contents

Sequence Diagrams

SD 1: Wizard Behavior

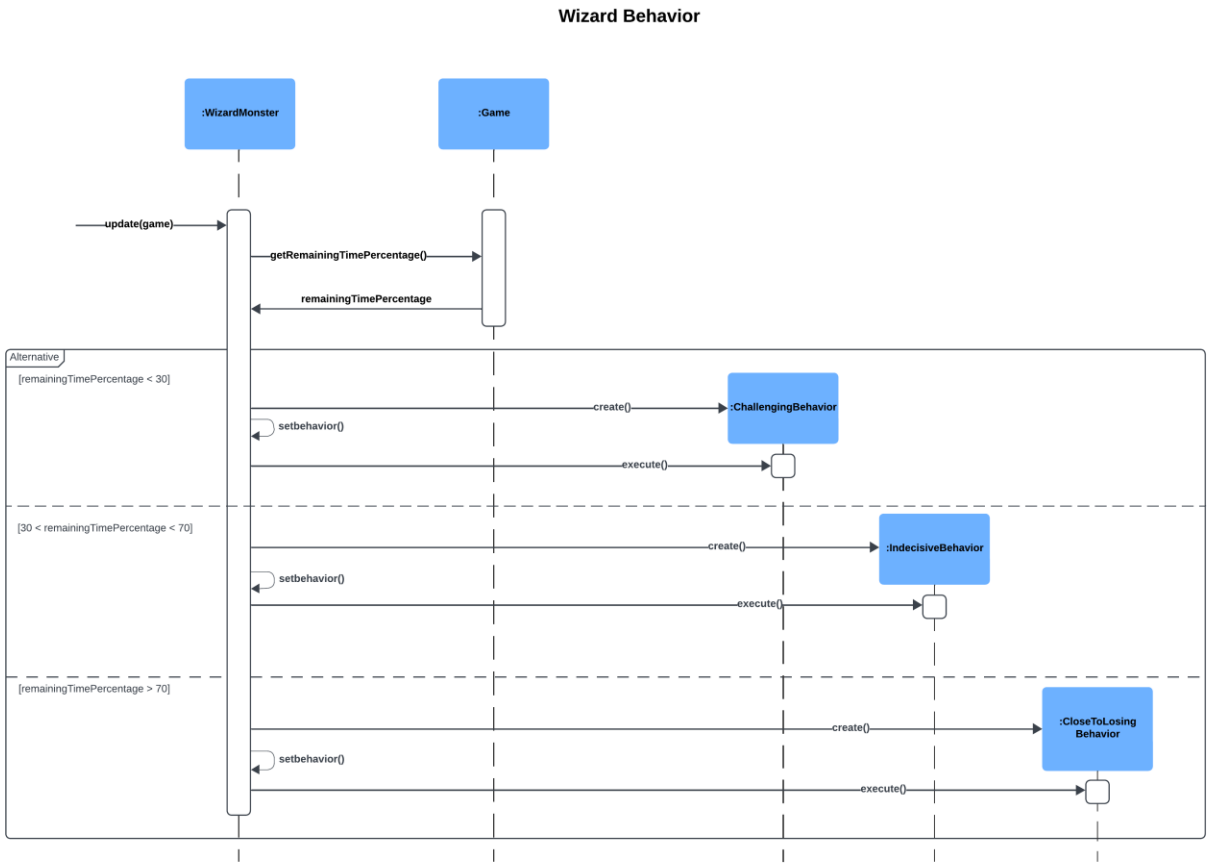


Figure 1: SD 1 – Wizard Behavior

SD 2: Wizard Disappear

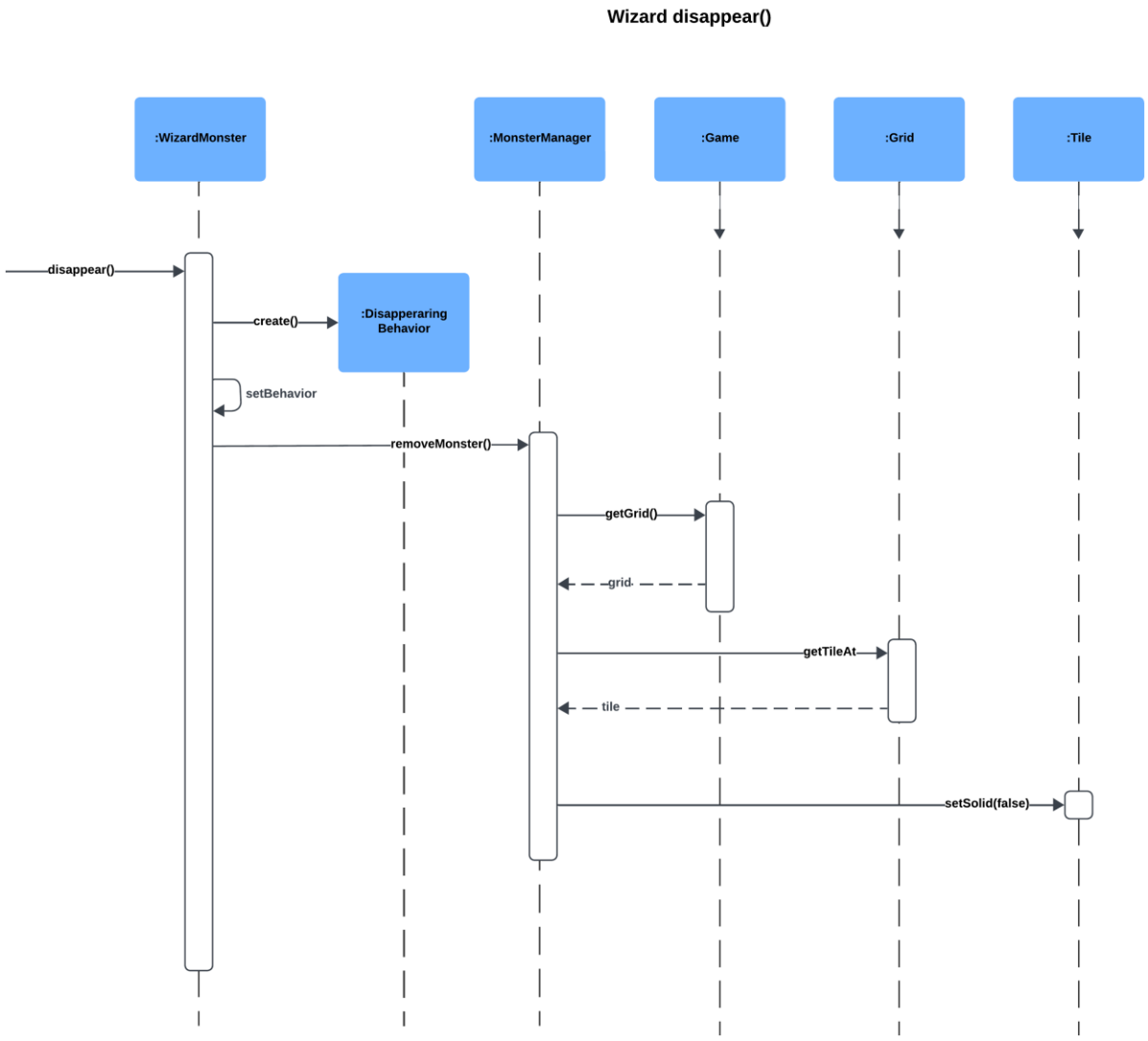
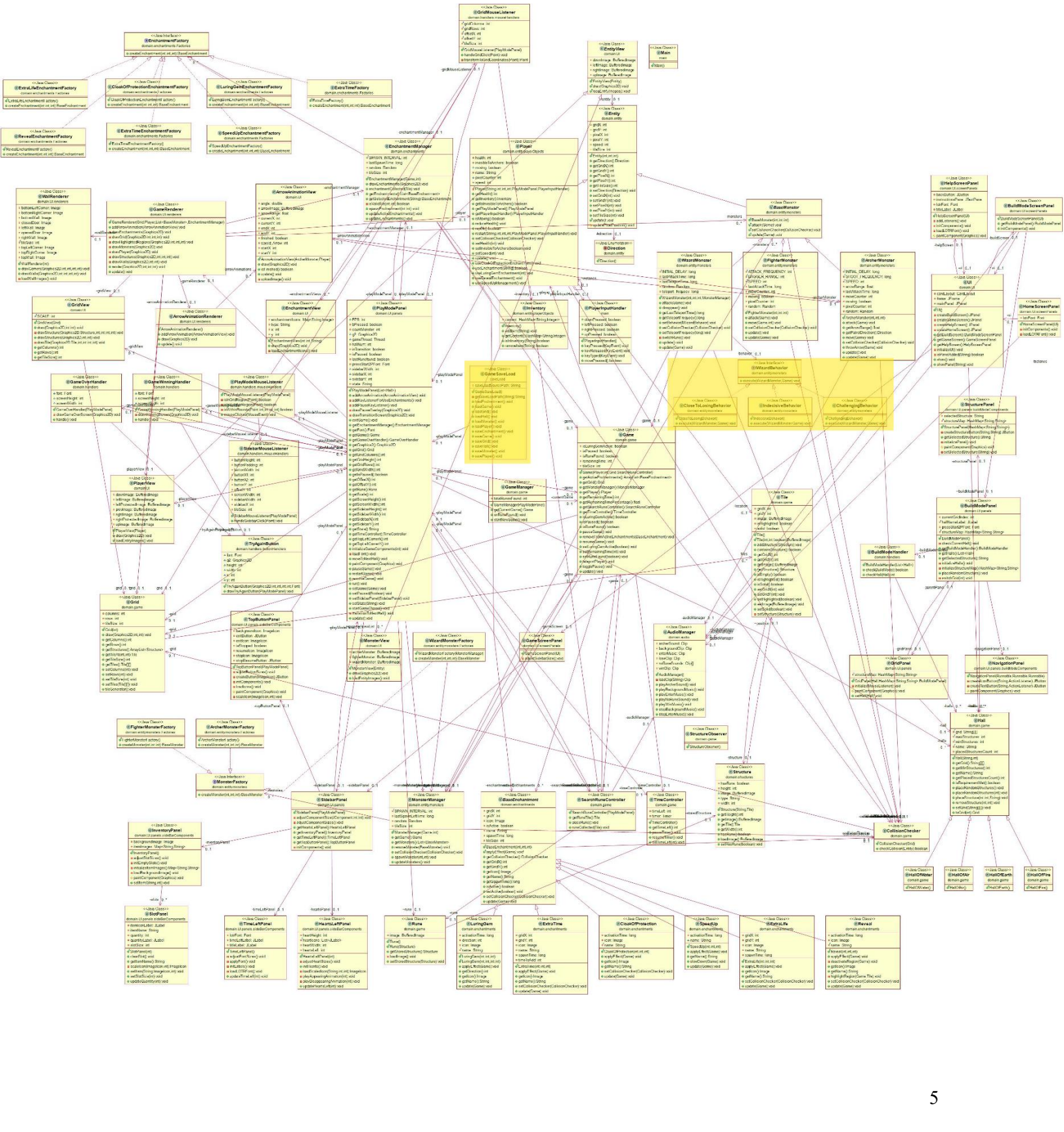


Figure 2: SD 2 – Wizard Disappear

Class Diagram



Design Alternatives

Controller Pattern: In our design, we used the controller pattern to design the `Game` class. The UI elements delegate the inputs to the classes at the domain level. The `Game` class controls the flow of the game, delegates the work of performing checks to the appropriate classes and then controls the game accordingly by calling certain methods of the classes. As a drawback, using the `Game` class to control too many things may make it bloated so we may add other controller classes to our design.

Creator Pattern: We will use the `Game` class to create the halls and the player. Because the `Game` class closely utilizes the halls and the Player object, it is responsible for the creation of these classes. The `Game` class also handles the creation of `Monster` and `Enchantment` objects since it holds the initializing data for them such as the spawning frequency.

Another class in which we applied the creator pattern is the `Hall` class. Each `Hall` will create its own grid based on its size. Since every `Grid` is contained in a `Hall`, the `Hall` has the responsibility of creating the Grid according to the creator pattern.

Strategy Pattern: We used the Strategy Pattern to implement the Wizard monster's dynamically changing behavior. Different behavior classes (ChallengingBehavior, IndecisiveBehavior, …) implement the IWizardBehavior interface, and the Wizard monster uses this interface to change its behavior according to the time left.

Information Expert Pattern: For collecting and using enchantments, we will use the Inventory class as an Information Expert. As the information expert, the Inventory class holds the number of each item in its inventory and other classes ask the Inventory for item counts and usability.

Low Coupling: The `Game` class does not shoot the arrows of the Archer monster directly, instead it triggers the Archer monster's `shootArrow()` method. Therefore the changes in the arrow shooting behavior do not directly affect the `Game` class. If we had the `Game` class be responsible for shooting the arrow, it would result in high coupling.

Factory: We used the Factory pattern to create Monsters and Enchantments. This way we can create similar types of objects efficiently.