SWEN30006 Report - Project 1

Part 1: An analysis of the current design

Based on the GRASP concept, software design should first balance high cohesion and low coupling. However, the current design fails to achieve this goal by placing all walking approaches of different types of monsters in the Monster class. Although this approach lowers coupling, it also reduces cohesion and creates a bloated class. This may make it difficult to identify the corresponding problem if a certain method fails in the future.

Secondly, to ensure an efficient arrangement of game flow, we need an Information Expert class. The Game class appears suitable for this role since it holds all the necessary information such as the game screen, monster locations and information, Pac actor, and items. However, the Game class also has some issues. It contains too much information and repeated statements, which could make modifying the class difficult in the future. Additionally, this design makes it challenging to add new functions in the future.

We also require a Creator, responsible for instantiating required objects, to minimize coupling between different classes. In the current design, the Game class acts as the Creator by creating items, the Pac actor, and various monster objects. While this design meets our requirements, having too many objects in the class can lead to clutter and hinder future development.

Another point is that we also need to ensure polymorphism in our design. Although the Troll and Tx5 have different patterns and actions, they are all monsters in the game. The current design only puts the Troll and Tx5 into the same class, which is insufficient for polymorphism. This could lead to difficulties when adding more monster types later and make distinguishing between different monsters challenging.

Part 2: Proposed new design of the simple version

To improve the design and facilitate future extensions, we first made modifications to the Monster class.

Initially, the walking approaches for all monsters were written in the Monster class. However, this was not ideal for adding more monster types and made the code less readable. Therefore, we used the principle of polymorphism and created Troll and Tx5 as child classes of Monster, with their own walking approaches written in their respective classes, turning the Monster class into an abstract class. This approach will make it easier to add more monster types and rewrite walking approaches in the future.

Additionally, we added a random walk method to the Monster class to reduce code duplication. We noticed that different monster walk approaches often required the random walk method. By adding it to the Monster class, different monsters can easily call this method, improving code reusability. This also reduces the amount of code, making it easier for programmers to read and modify.

Finally, a new class GameVersion was created to store changes in different game versions. According to our understanding of multiverse game versions, we assume that the differences between versions focus on the types of monsters and the effects of various types of pills. Thus, GameVersion should be responsible for creating monsters and providing methods to implement new functions of eating pills. Moreover, as there would be more versions added, the GameVersion was set as an abstract class. The implementation of the simple game version was placed in a child class of GameVersion, the SimpleVersion.

Part 3: Proposed design of extended version

In the multiverse game version, we have introduced three new monsters: Alien, Wizard, and Orion, along with two new features related to consuming gold and ice. The Alien has the ability to calculate the distance to Pacman and find the shortest path. It can move in eight different directions, making it a formidable adversary. The Wizard, on the other hand, does not chase Pacman but has the unique ability to pass through one-cell thick walls. Orion, the third monster, is tasked with guarding the gold. It randomly selects an existing gold piece as its target and, once it reaches it, moves on to the next piece. To make the design of the new monsters easily extendable, we have implemented them as child classes of the Monster class.

The consumption of ice or gold has a significant impact on the monsters in this version of the game. Eating ice will freeze the monsters for three seconds, whereas eating gold will cause them to move an additional cell within the same time period. It's important to note that eating multiple pieces of ice or gold will not extend the freeze or furious time. To accommodate these changes, we have created an enum state to indicate the different states of the monsters, including "Normal", "Freeze", and "Furious".

To implement these changes, we have developed a new class called MultiverseVersion, which is a child class of GameVersion. This class is responsible for creating monsters and providing methods to handle the consumption of ice and gold. A MultiverseVersion object will be stored in the game class to enable seamless gameplay.

Software Models:

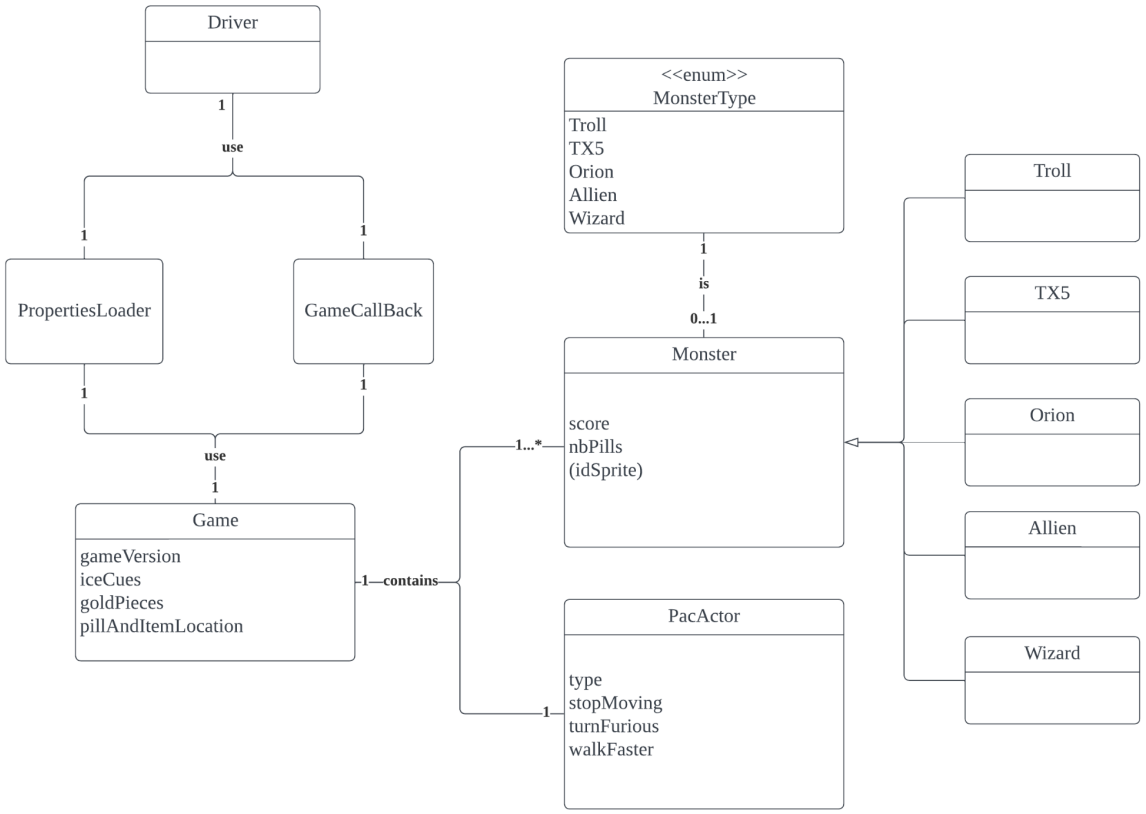

Figure 1: domain class diagram

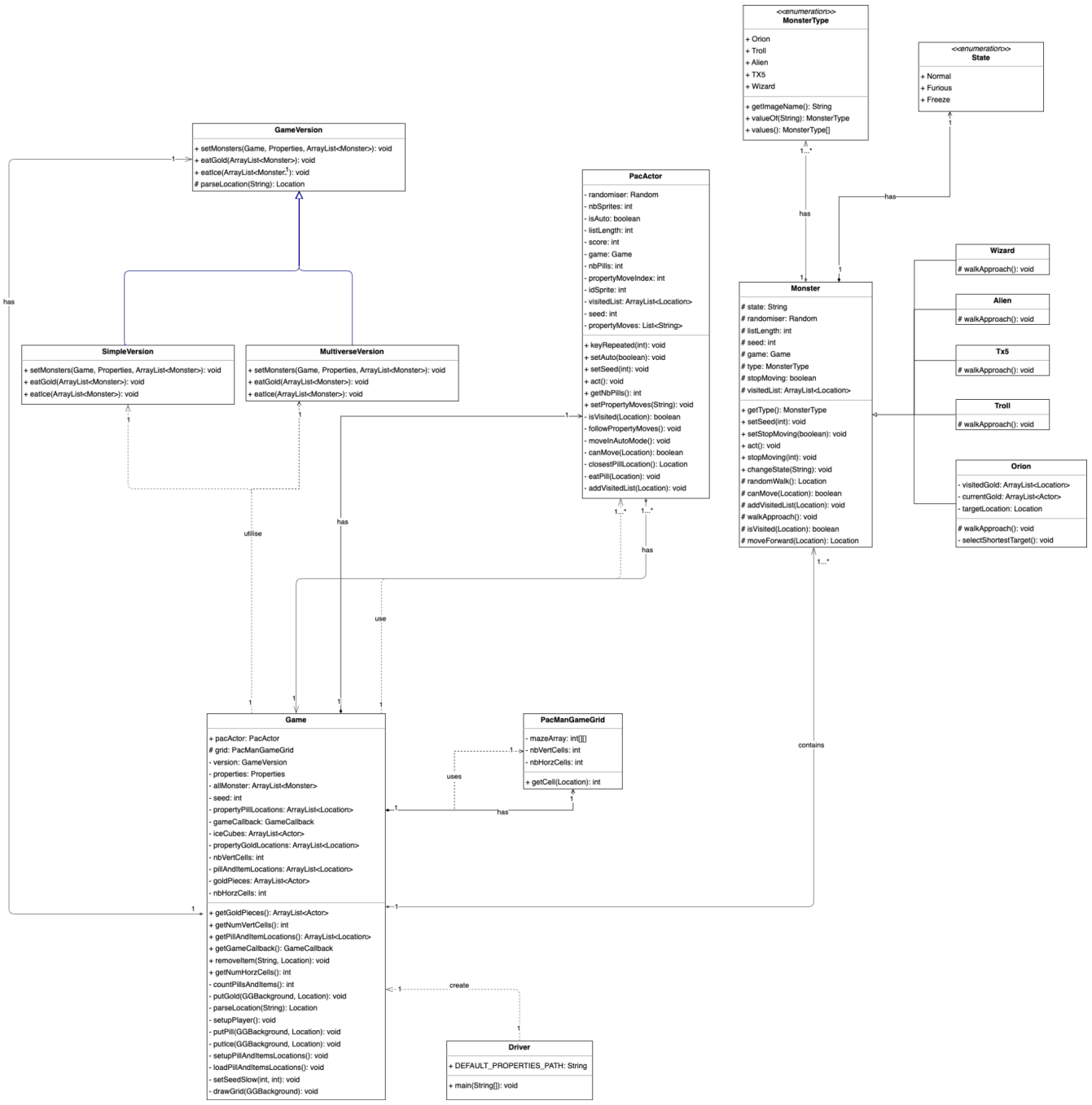

Figure 2: static design model(design class diagram)

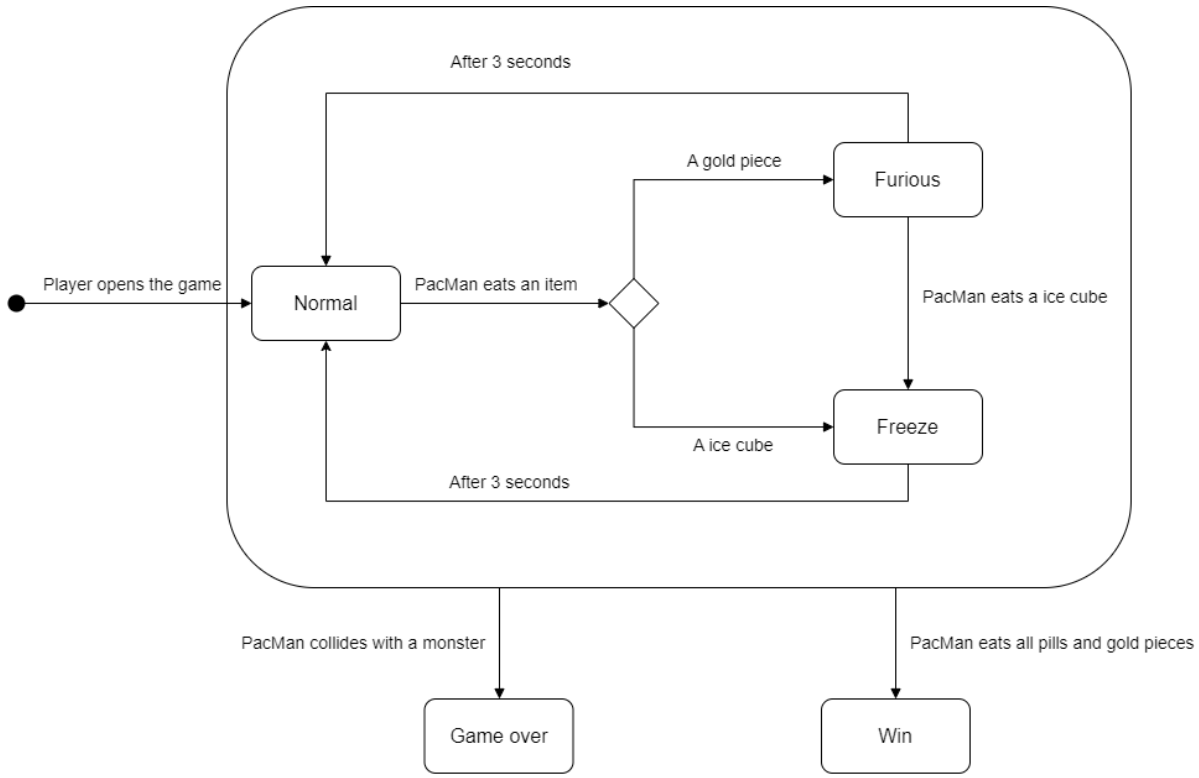

Figure 3: dynamic design model(state machine diagram)