SWEN30006 Report - Project 1 PacMan in the Multiverse

fri-11:00 team 2

Part 1: The concerns of the current design

The current design is low cohesion and high coupling, which hinders the expansion of new features as the ‘Game’ and the ‘Monster’ class are bloated. All behaviors of different monster actors are written in the ‘Monster’class, with simple if/else statements determining the walking approach method of monsters. Hence, implementing new rules for new features will require many changes, leading to a low cohesion design.

In particular, the ‘Game’ class has too many responsibilities, which fails to meet the Information Expert principle in GRASP. Too much concentration on responsibilities will lead to a lack of collaboration in the overall design, and future modification is required when adding new features. For example, the ‘Game’ class not only contains game data and logic but also includes the implementation of the game interface(gameGrid.draw()), which may fit better in the ‘PacManGameGrid’ class.

Moreover, the current design could be better in extensiveness. For example, pills, gold, and ice are stored as separate variables in the ‘Game’ class though having similar properties. Such design complicates adding features in the future – adding a new entity will need to modify ‘Game’s attributes and all related methods.

The current code base also has minor issues, such as bloated methods and redundant codes, making the code base less maintainable and testable.

Part 2: Proposed new design of the simple version

Based on the assumption that the game will be further expanded in the future, we tried to refactor the game into different modules.

Firstly, according to the Polymorphism and Information expert principle, we abstracted the ‘Monster’ class and have specific monster types as subclasses, which can override behaviours in the ‘Monster’ class if a particular implementation is needed. The new design aligns more with high cohesion and low coupling principles. When new monster actors need to be added, we can easily add inheriting classes without impacting the entire system structure. In addition, subclasses can inherit methods, reducing the workload of further extensions. Here, we added two monster subclasses, ‘Troll’ and ‘TX5’, for the simple version to separate the ‘walk approach’ of the two monsters. If the settings of the monsters change in the future, only their respective subclasses need to be modified, which increases cohesion.

At the same time, similar to abstracting the ‘Monster’ class, we also abstracted a ‘StaticEntity’ class and stored as a single list in the Game class to facilitate future expansions. The ‘Gold’, ‘Ice’, and ‘Pill’ can inherit ‘StaticEntity’ to increase cohesion, reduce coupling, and make adding entities in the future without hassle.

On the other hand, we also created an enumeration class, ‘CellType’, to set map elements so that map elements can be determined without needing more details for further extensions, which fits the Pure Fabrication and Information expert principle of GRASP.

Furthermore, based on the indirection principle, we added a ‘GameActor’ class as a controller, which acts as an intermediate class between the ‘Actor’ class in the external library and the ‘PacActor’ class and ‘Monster’ classes in our internal code base. This design allows a specific class to manage methods in the external library needed to create the game, which reduces the coupling between the library and the game, increases cohesiveness, and benefits maintainability – only one class needs to be modified if changes occur in the external library.

It’s also noteworthy that separated attributes for monsters in the Game class are aggregated into a single Monster ArrayList so that potential future adjustments to monsters in the future (e.g., having two Trolls instead of one) can be made quickly.

We also moved parts not belonging to the controller out of the game class. For example, we moved portions of the method draw() from the ‘Game’ class to the PacManGameGrid, as PacManGameGrid already has enough information to draw parts of the grid, and the grid drawing process is more related to the PacManGameGrid class than the Game class.

Part 3: Proposed design of extended version

For the extended version multiverse, three new monsters are added, and a new game logic regarding Pacman eating gold pieces and ice making monsters furious and freezable functions, respectively, are added. Based on the same reason for abstracting the ‘Monster’class for the simple version, we added three new subclasses, ‘Orion’, Alien’, and ‘Wizard’, which inherit the ‘Monster’ class with unique functions written to their respective subclasses. One thing that needs to mention is that to implement the function of finding the shortest path for the monster 'Orion' as fast as possible, we add a new ‘PathNode’ class to help with a BFS path-finding algorithm.

In addition, although the functions of ‘furious’ and ‘freezable’ are not complicated at present, considering that the game may expand in the future, we still created two interfaces, ‘Furious’ and ‘Freezable’, to separate them from the ‘Monster’ class.

Finally, to switch between different versions of the game and increase code reusability, as well as prepare for possible future expansions, based on pure fabrication, indirection, and the Protected Variations principle of GRASP, we created a 'PacVersion' interface, with 'SimpleGame' and 'MultiverseGame' implementing the interface. As a result, different versions can optionally implement methods 'eatPill', 'eatGold', and 'eatIce' based on their own version rules, which decrease coupling and increased cohesion.

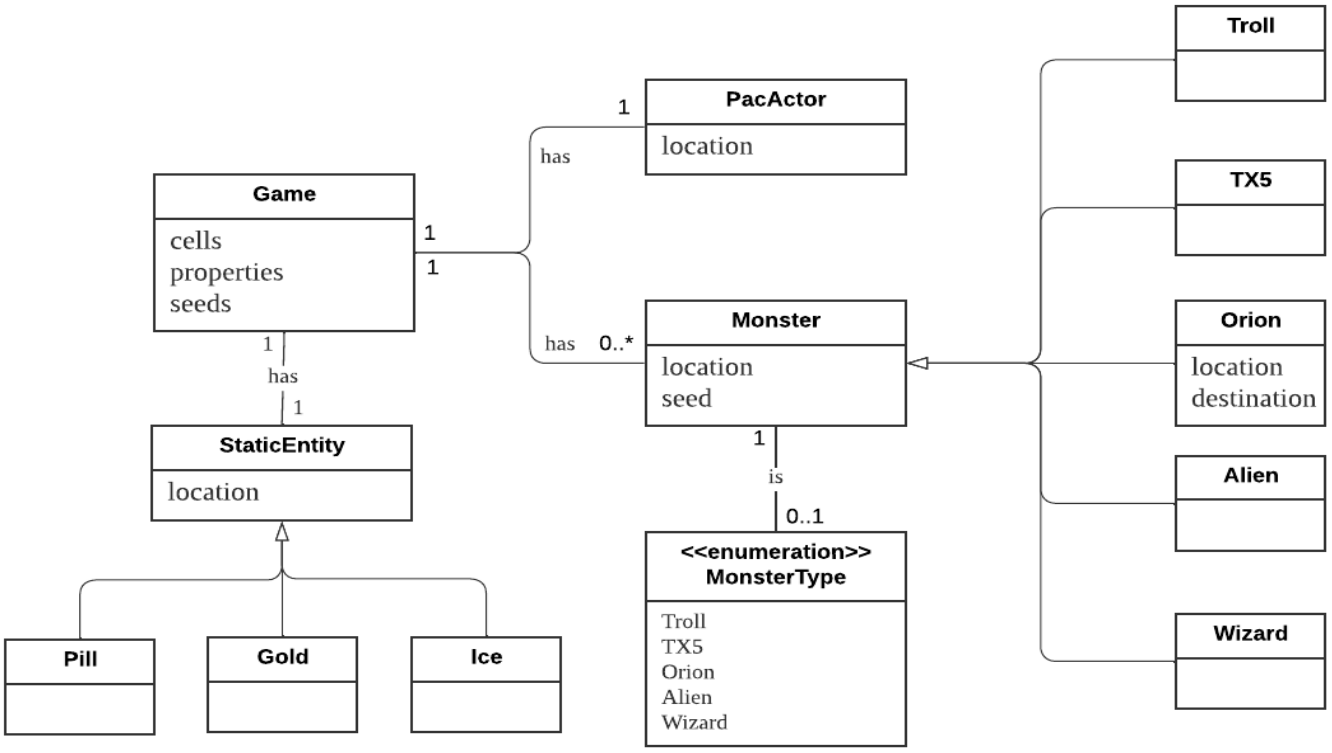

Figure1. domain class diagram for capturing the noteworthy concepts and their associations of PacMan in the Multiverse, including the extensions

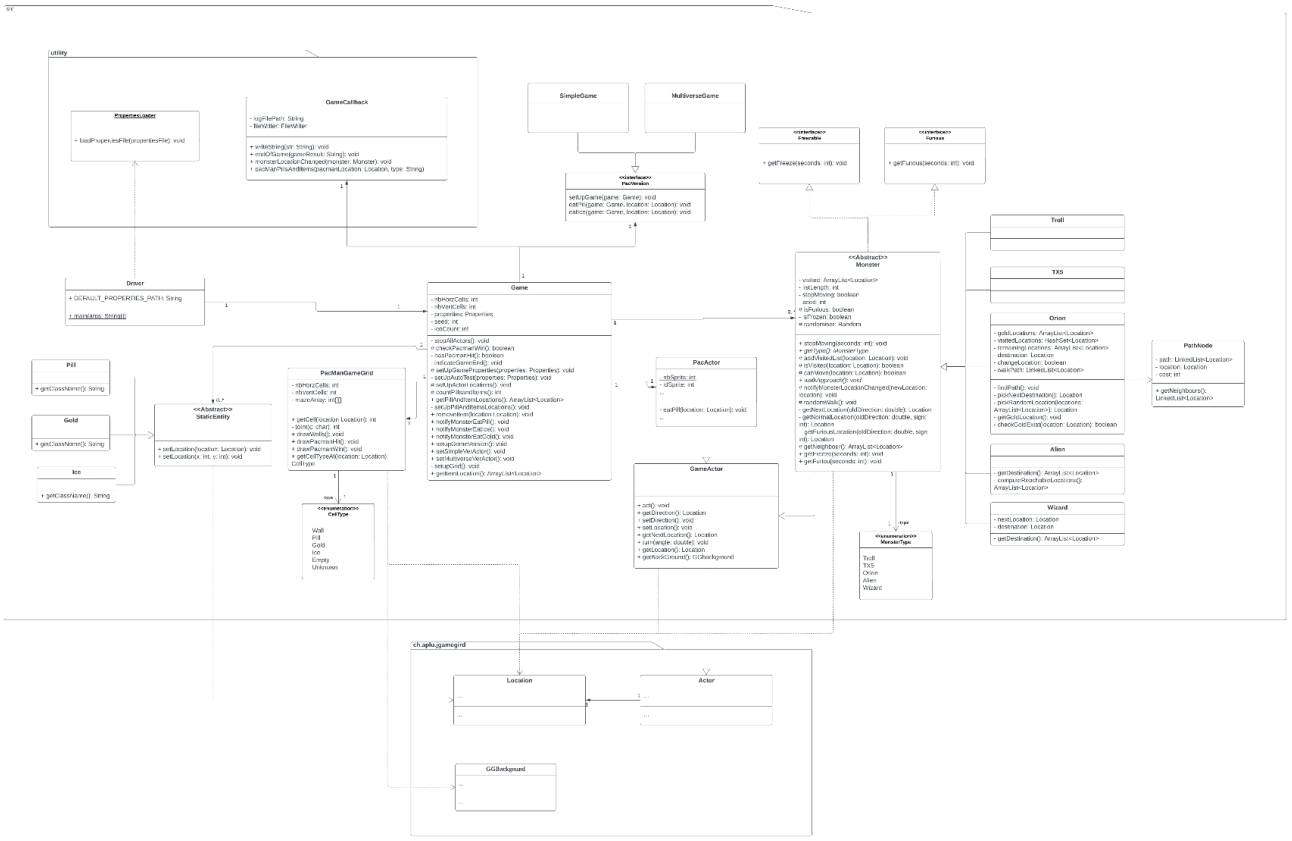

Figure2. design class diagram for documenting your new design for PacMan in the Multiverse, including the extensions (some details of the old design are hidden).

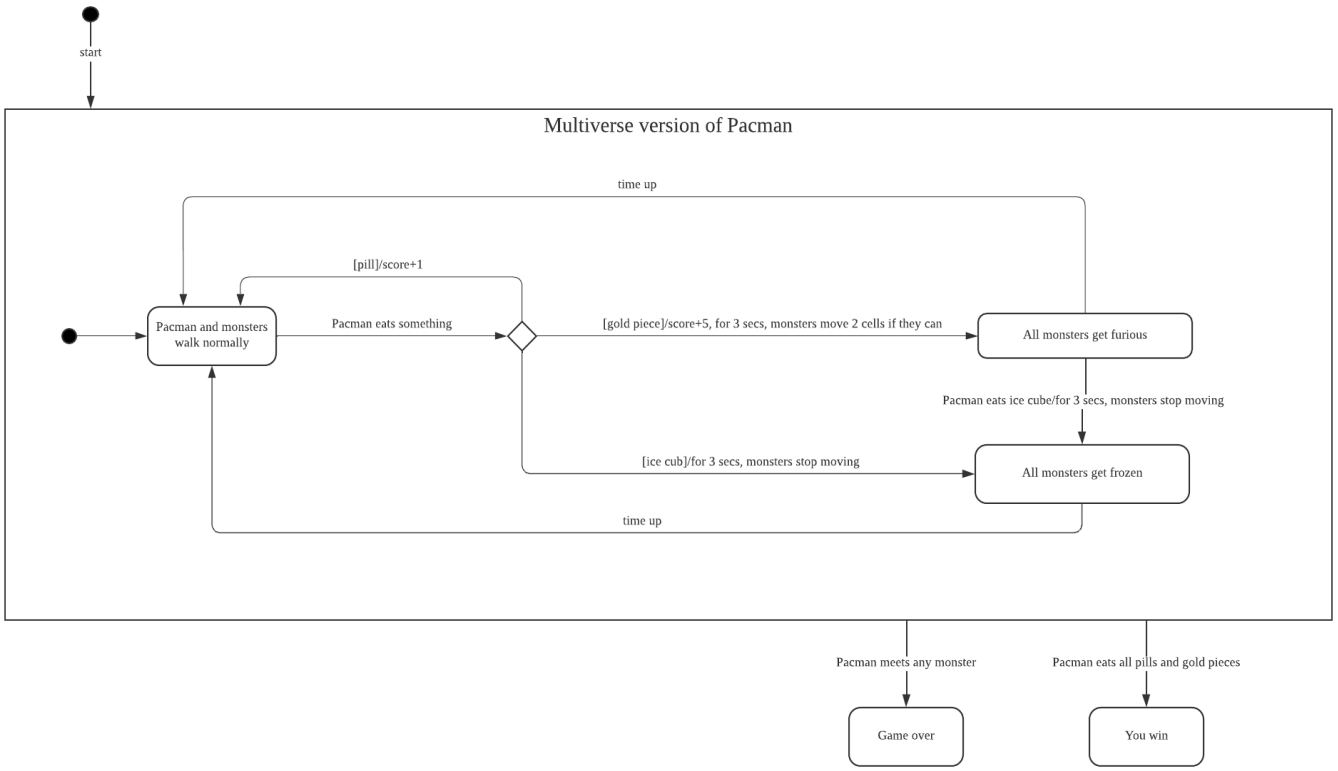

Figure3. state machine diagram) for documenting the system's behaviour of additional capabilities (the effects on the monsters) of ice cubes and gold pieces as per 3c and 3d described in Section 1.2