SWEN30006 Software Modelling and Design
Project 1: PacMan in the Multiverse
- Project Specification -

School of Computing and Information Systems
University of Melbourne
Semester 1, 2023

Background

The classic arcade game PacMan has been around since the 1980's. It is a maze action video game where a player controls the PacMan character through an enclosed maze to eat all of the pills (small white dots) placed in the maze while avoiding ghosts. PacMan was a widespread critical and commercial success, but die-hard players need something new.

Arcade 24™ (A24), the modern game company has stepped in to fill this need with their new game design, **PacMan in the Multiverse**. Instead of having ghosts like a classic PacMan game, A24 wants various monsters. In addition to pills, new items (ice cube and gold pieces) will be placed into the maze. These new items are intended to provide additional behaviours of the gameplay. A previous team of A24 has developed a simple version of the PacMan in the Multiverse; their code works with limited features. The code includes some documentation (code comments and partial design class diagram), but unfortunately it was built in a rush and was poorly designed. A24 has recruited you and your team to revise the existing design and codebase and extend the simple version of PacMan in the Multiverse to include the features that will make the game more exciting.
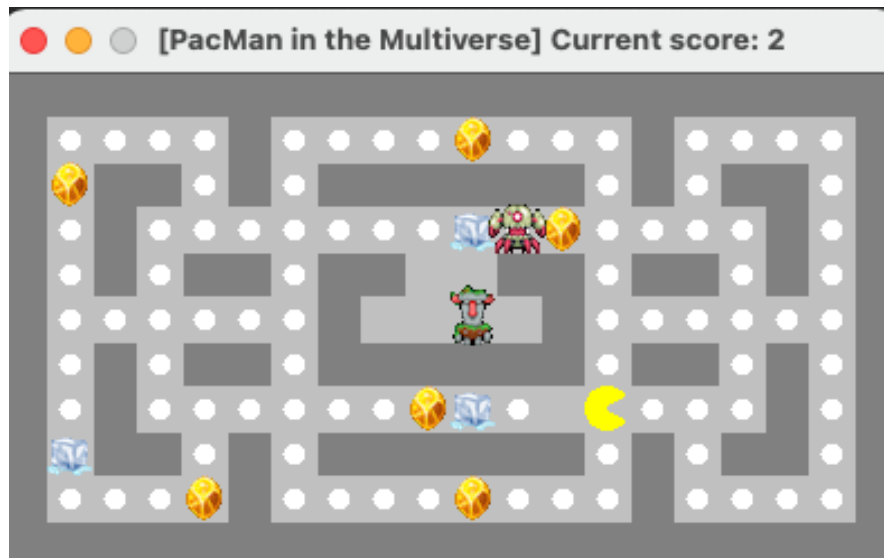

Figure 1: The GUI of the simple version of PacMan in the Multiverse.

1. Use Case: Playing PacMan in the Multiverse

Section 1.1 provides the use case text describing the simple version of PacMan in the Multiverse that is already built and provided in the codebase. Section 1.2 provides the A24's proposed extensions. The extensions have not yet been built and will be detailed further in the specification.

1.1 Playing PacMan in the Multiverse

Main Success Scenario:

1. The player opens the game which initiates and places PacMan, pills, ice cubes, gold pieces, and monsters (one Troll and one T-X5) into the map. The initial locations of PacMan and monsters are configurable.
2. The game moves the monsters automatically within the maze. A monster will move one cell at a time. A monster determines a next location to which it will move based on a given direction. The direction can be specified based on compass directions (e.g., East) or degrees clockwise ($0° - 360°$ where $0°$ is East and $270°$ is North). For example, in Figure 2, the current direction of the monster (a red arrow) is towards East (or $0°$). To move to the bottom cell (i.e., going down; a blue arrow), the direction should be set to South or $90°$.

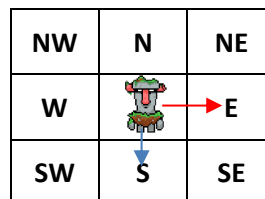| NW	N	NE
W	🧌→	E
SW	S	SE

Figure 2: An example of a monster direction.

Each type of monsters has the following walking approach.

Monster	Name	Walking Approach
	Troll	<u>Random Walker:</u> Troll will randomly turn left or right (i.e., -90° or +90° from the current direction). If it hits the maze wall, it will turn back to the original direction and go forward or turn the other side. Otherwise, go backward.
	T-X5	<u>Aggressive Follower:</u> T-X5 walks towards PacMan if it can. It determines the compass direction from itself to PacMan. Then, it will move towards that direction. If the next location is a maze wall or has been visited, it will randomly walk like Troll. When the game starts, T-X5 will wait (no moving) for 5 seconds before start moving.

3. The player controls PacMan with keyboard actions to walk in the maze. The Left and right keys move PacMan horizontally. The Up and down keys move PacMan vertically.

Repeat steps 2-3

Alternate Scenarios

2a., 3a. If there is a collision between PacMan and one of the monsters (i.e., being on the same location), PacMan explodes and the game is over.

3b. When PacMan eats a pill ⬜ (being on the pill's location), the pill disappears and the current score on the title bar increases by 1 point.

3c. When PacMan eats a gold piece 🟡(being on the gold piece's location), the gold piece disappears and the current score on the title bar increases by 5 points.

3d. When PacMan eats an ice cube 🧊(being on the ice cube's location), the ice cube disappears but the current score doesn't increase.

3e. When PacMan eats all the pills and gold pieces, the player wins the game.

*a. At any stage, the player exits the game.

1.2 Proposed Extensions

A24 proposes the following new features for the extended version of the game.

Feature 1: New Types of Monsters

Extends Playing PacMan in the Multiverse

2b. Three new kinds of monsters are created and placed into the map. Their initial locations are specified in the configuration file. A monster will move one cell at a time. Each type of monsters has the following walking approach.

Monster	Name	Walking Approach
	Orion	Gold Surveillance: Instead of chasing PacMan, Orion wants to protect gold pieces. Orion will walk through every gold location by selecting one gold piece at random, walking to it, and so on. When all gold pieces have been visited, Orion will start over again by randomly selecting any gold locations. Even though the gold pieces are eaten by PacMan, Orion still will visit those locations. However, Orion will favour locations which still have gold pieces, that is, it will select from locations with gold pieces before other locations where the gold pieces have been eaten by PacMan. Once a gold location has been selected, Orion will visit it, even if the gold piece in that location is eaten while Orion is on the way there.
	Alien	Shortest Distance Finder: Unlike T-X5, Alien will calculate the distance between PacMan and each of the neighbour locations (8 cells around the Alien) that it can move (i.e., not the maze wall). Then, it will move to the neighbour location that has the shortest distance to PacMan. If more than one neighbour locations have the shortest distance, it will randomly select one.

	Wizard	Wall-Through Walker: Wizard has a magic to walk through a 1-cell-thick wall. Wizard randomly select one of its neighbour locations (8 cells around the Wizard).
		• If the selected location is not a maze wall, Wizard will move to that location.
		• If the selected location is a maze wall, it will check if the adjacent location in the same direction as the selected location is a wall or not:
		  o If the adjacent location is not a wall, the Wizard walk through the wall to that adjacent location.
		  o If the adjacent location is a wall, Wizard will randomly select another neighbour location around itself.

Feature 2: Additional Capabilities of Items
Extends Playing PacMan in the Multiverse
3c. When PacMan eats a gold piece, all the monsters get furious and move fasters. The monsters determine the moving direction once based on their walking approach and move towards that direction for 2 cells if they can. Otherwise, determining the new direction again using their own walking approach until it can move by 2 cells. After 3 seconds, all the monsters will be back to move normally using their own walking approach.
3d. When PacMan eats an ice cube,
    3d.1. Regardless of being normal or furious, all the monsters are frozen (i.e., stop moving) for 3 seconds. Then, they will be back to move normally using their own walking approach.
    3d.2. While the monsters are frozen, PacMan can eat a gold piece without making the monsters furious.

NOTE: A24 acknowledges that the descriptions of the new features above may not cover all the possible cases of the game logic. As long as the described logics is in place, A24 is open to your ideas to handle the corner cases that are not covered by the provided use cases.

2. Your Task

2.1 Refactor the existing codebase.
A24 found that the simple version of PacMan in the Multiverse has a poor design as it is not currently very extensible. Part of your task is to refactor the existing code to meet software engineering design principles. A well-designed model will make the design and implementation of the proposed extensions much simpler. Thankfully, the existing code is quite well documented with a design class diagram and code comments to assist you with understanding the code to start refactoring.

2.2 Add new features with configurable parameters.
A24 wants PacMan in the Multiverse to be extended to include the new features described in Section 1.2. Nevertheless, A24 needs to ensure that your refactoring still preserves the original game's behaviour. Thus, a **configurable** version is needed. If the game version is configured to "*simple*", the game should behave as described in Section 1.1. If the game version is configured to "*multiverse*", the game should include additional behaviours described in Section 1.2. In addition, as described in Section 1, the locations of the game characters (PacMan and monsters) can be configurable. Examples of properties files in the **properties** folder include all necessary configurable parameters are provided in the base package.

3. The Base Package (Simple Version)

3.1 Getting started
- The package is provided as an IntelliJ project in a GitHub repository: https://classroom.github.com/a/l7Jqvftw To access the base package and set up, you can follow the instructions in Workshop 1. Note that the project requires at least Java 17.
- Then build and run the project. You will see the GUI appear and can play the simple version of PacMan in the Multiverse using the standard keyboard actions.

3.2 System design
- The classes in the provided 'src' package primarily handle the game behaviour, while the GUI operates by using the JGameGrid library. You can refer to the classes and their functions in this framework in Java Doc.

- o The required changes for this project relate only to the application behaviour. You should not need to modify any code pertaining to the GUI.
  - o Implementation Hints: The JGameGrid library has provided many useful functions that you can reuse to implement the proposed extensions. The codebase also should provide examples for implementing the extensions.
- The system reads in a properties file that will specify the version, locations of PacMan and monsters, random seed, and auto-play configuration (for testing purposes). You can edit the properties in these files for your own testing, but make sure to preserve the default behaviour.
- The **moveInAuto** function and its related functions in the **PacActor** class are for testing your system and the **GameCallback** class is used to record the game behaviours in **Log.txt**. These functions are used by the SWEN30006 staffs. Please ensure that any adjustments you make do not affect the behaviours and outputs from these functions.
- Image files of characters and items are provided in the 'sprites' folder. The file names of these image files must not be changed.
- A partial class diagram of the simple version has been provided to help understand the current design (see **Appendix 1**).

3.3 Testing Your Solution
- We will be testing your application programmatically, so we need to be able to build and run your program without using an integrated development environment (IDE).[1] You must to ensure that the entry point must remain as **"src.Driver.main()".**
- You must not change the names of properties in the provided property file or require the presence of additional properties. If you add properties, they need to have defaults values as we will not set these in testing your submission.
- **Properties Files**: 5 examples of properties files are provided.
  - o **test1.propperties** is an example of configuration for the original behaviour described in Section 1.1
  - o **test2.properties** and **test3.properties** are provided to automatically move PacMan in an auto test mode.
  - o **test4.properties** and **test5.properties** are examples of configuration that we will use to test your extended version.
- You need to ensure that any modification still preserves the original behaviour, i.e., Log.txt generated by your extended version is the same as Log.txt generated by the provided code base when using *test2.properties* and *test3.properties* files that we provided.

4. Project Deliverables
4.1 Extended version of PacMan in the Multiverse: Submit your source code (with any/all libraries used included)
- Ensure that your code is well documented and includes your team name and team members in all changed or new source code files.

4.2 Report: A design analysis report detailing the changes made to the project and the design analysis including identifying design alternatives and justifying your design decisions with respect to design principles. Specifically, your report should address the following points:
- (P1) **An analysis of the current design**: Based on GRASP principles, explain the concerns of the current design and how it may hinder the extension of the new features.

---

[1] You do not need to be concerned about running without IDE if your system can be built and run using your IDE.

- (P2) **Proposed new design of the simple version**: Describe and justify how the simple version should be refactored to achieve a better design and/or facilitate future extensions.
- (P3) **Proposed design of extended version**: Describe and justify your design for the new features that are added into your new game design.

4.3 Software Models: To facilitate the discussion of your report, the following software models should be provided in the report and used along with the discussions in P1-P3. You can use subdiagrams or provide additional diagrams where appropriate.

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

- A static design model (i.e., a design class diagram) for documenting your *new* design for PacMan in the Multiverse including the extensions.

- A dynamic design model (i.e., a state machine diagram) for documenting the dynamic behaviour of monsters in terms of their different statuses regarding effects of the gold pieces and ice cubes as per 3c and 3d described in Section 1.2.

5. Submission

All project deliverables should be included in the 1 project zip file with the specified structure (see Figure 3). **Do not** include the .git folder in your submission. **Only 1 member** in the team submits the file. **See more details about submission and evaluation on the project submission page.**

Note: It is your team's responsibility to ensure that you have thoroughly tested their software before submission.
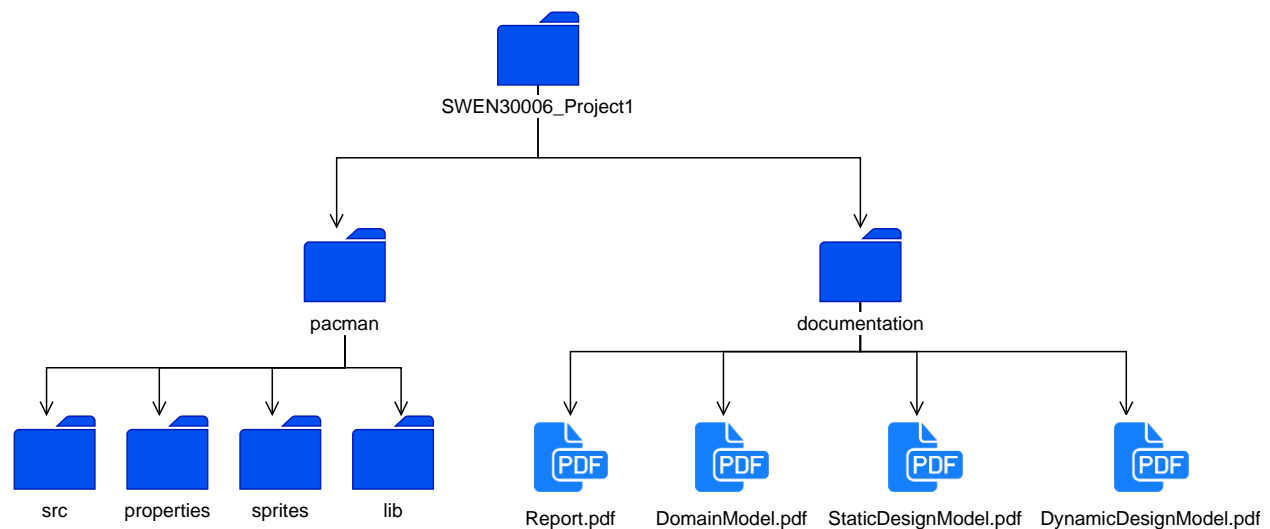

Figure 3: File structure for project submission

Appendix 1: Partial Class Diagram of the simple version