SWEN30006 Software Modelling and Design
Project 2: PacMan in the TorusVerse
- Project Specification -

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

1.1 Background

As you know, Arcade 24™ (A24), the modern game company has been developing their new game design, ***PacMan in the Multiverse*** to provide die-hard PacMan players with something new. That version of the game adds various monsters and maze items.

However, A24 is aware that the Multiverse alone will not be sufficient to engage the die-hard players. So, while development on the Multiverse continues, A24 has engaged your team to develop an editor and test app for *another variation of the game*: **PacMan in the TorusVerse.** You will not be concerned with the changes for the Multiverse**:** TorusVerse will be developed based on the simple version. Then, A24 will deal with merging the MultiVerse and the TorusVerse at some point in the future.

The TorusVerse will focus more on the maps provided for the game, and included features to allow characters to, for example, exit from the left and reappear on the right (doing this left-right and top-bottom effectively makes the map a torus). Further, the game will support multiple levels so players can progress from one level to the next. To provide these maps, TorusVerse will need a map editor and tester.

Figure 1: The GUI of the tile map editor for PacMan in the TorusVerse.

2 New Application Requirements

2.1 Game Capabilities

2.1.1 Multiple levels/maps

1. A game of PacMan consists of multiple maps; all these maps are stored in a single folder (*Game Folder*).
2. Each map file in the Game Folder has the extension ".xml" and is stored in xml format.
3. The maps are a grid of tiles; each tile corresponds to one location in the game map.
4. These maps include the locations of all elements; the existing property file locations become irrelevant.
5. All and only map files with names starting with a number (1, 2, …) are included in the game. Other files in the Game Folder are ignored.
6. Maps are played in ascending number order. No two maps can have the same number. The numbers do not need to be contiguous.
7. If PacMan successfully completes the last map, it wins the game; completing an earlier map will take PacMan to the next map.

2.1.2 Portals

1. A map can now contain pairs of portals.
2. Portals are represented by portal tiles on the map; two matching portal tiles in different locations are required.
3. Each pair is connected: if PacMan or a Monster moves onto one portal, it will be transported instantly to the other portal. (To go back it would need to move off the portal and then move back on.)

These portals can be used to create a torus effect, that is, to have creatures move off one side of the map and appear on the opposite side. They could also be used, for example, to give PacMan access to an area which is not otherwise accessible (like an isolated room) or a way to bypass pursuing monsters which would corner PacMan.

2.1.3 Autoplayer

The provided basecode supports an autoplayer setting and provides and autoplayer, however you are required to provide a smarter autoplayer with potential for further improvement.

1. The game should include an *autoplayer* setting.
2. If autoplayer is off, the game is played interactively by a human player in the usual way.
3. If autoplayer is on, the game is played automatically.
4. The autoplayer must be capable of completing the game (all levels) in the case that no monsters are present. Note that this needs to consider the presence of any portals.

5. The autoplayer must use a directed approach to find and eat each gold/pill in turn. It must not use random movements to try and stumble across these items. (Note: Finding and downloading path-finding or similar Java code may make this task easier.)
6. The autoplayer should be designed to allow for easy extension into being a smarter autoplayer in the future, e.g. one which can deal with monsters (Trolls and T-X5 at least) and ice cubes.

2.2 Application Capabilities

The system you are to develop will support editing the maps in a game folder and playing the game (either all levels or one level) to test these maps.

2.2.1 Start Up

1. Starting the editor with a folder as an argument will start in Test Mode, that is, start playing the game levels in that folder.
2. Starting the editor with an existing map as an argument will start in Edit Mode on that map.
3. Starting the editor with no argument will start in Edit Mode with no current map.
4. Autoplay mode will be set by the property file, that is, whether play is interactive or conducted by the autoplay will be determined by the property PacMan.isAuto in the property file.
5. Only the PacMan.IsAuto and seed properties in the property file are relevant to this application.

2.2.2 Edit and Test Mode

The behaviour in test mode is independent of whether the application is in Autoplay mode.

1. Whenever a map is loaded or saved, *level checking* (see below) is applied. If a level is being saved, it should be saved no matter the result of level testing.
2. Only saved and loaded maps can be tested, so level checking should occur (once) before any map is tested, including when a game folder is being tested. Failed level checking should result in the application returning to edit mode on the level which failed level checking.
3. Game checking should occur before any game folder is tested; if game checking fails, the game should not be tested and the application should return to edit mode with no current map.
4. If Test Mode is started with a game folder, all levels are played in sequence until PacMan is destroyed by a monster or wins the game by completing the last level. The application then returns to Edit Mode with no current map.
5. If Test Mode is started with a specific level/map, that level is played until PacMan is destroyed by a monster or that level is completed. The application will return to Edit Mode on that map.

2.2.3 Level Checking

1. The application should be able to check the validity of a level according to a defined set of rules.
2. The application should be designed to allow a developer to easily modify or add checks.
3. Where a check fails, that failure should be reported to the log file in text format.
4. The level checks required (for the current version) follow. The text in square brackets is an example of the format required for the log file message, with pairs of numbers in parentheses being grid coordinates with (1,1) at top left.
    a. exactly one starting point for PacMan

[Level 2mapname.xml – no start for PacMan]
[Level 5_levelname.xml – more than one start for Pacman: (3,7); (8, 1); (5, 2)]

b. exactly two tiles for each portal appearing on the map
[Level 1_mapname.xml – portal White count is not 2: (2,3); (6,7); (1,8)

c. at least two Gold and Pill in total
[Level 6levelname.xml – less than 2 Gold and Pill]

d. each Gold and Pill is accessible to PacMan from the starting point, ignoring monsters but accounting for valid portals
[Level 2mapname.xml – Gold not accessible: (4,7); (4,8)]
[Level 2mapname.xml – Gold not accessible: (4,9)]

2.2.4 Game Checking

1. The application should be able to check the validity of a game folder.
2. Where a check fails, that failure should be reported to the log file in text format.
3. The game checks required follow. The text in square brackets is an example of the format required for the log file message.
    a. at least one correctly named map file in the folder
    [Game foldername – no maps found]
    b. the sequence of map files well-defined, that is, is there only one map file named with a particular number
    [Game foldername – multiple maps at same level: 6level.xml; 6_map.xml; 6also.xml]

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 logic is in place, A24 is open to your ideas to handle the corner cases that are not covered by the provided requirements, within the limits of testability; if unsure about consistency with this spec or testability, please ask

3 Your Task

You are being provided with
1. PacMan basecode: the same codebase used as a starting point for the Multiverse project
2. Tile editor: a codebase which can be used to create tile-based maps for PacMan

You should create a single application with the behaviour described above. You can refactor the code as you see fit, but should preserve the playing behaviour where new functionality does not apply. For example, a level with no portals should play out like it would with the PacMan basecode.

4 The Base Package

4.1 Getting started

- The PacMan and Tile Editor basecode is provided as an IntelliJ project in a GitHub repository. (See URL in Project 2 on the LMS.) 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.

- You will need to build and run each project independently (PacMan and TileEditor) to get them working. (You will need to merge them into a single application to meet the project requirements.)
- When you run pacman, you will see the GUI appear and can play the simple version of PacMan using the standard keyboard actions. For Tile Editor, there are 2 sample map data provided to you (sample_data and sample_data2). When you see the GUI, please select Load and choose the whole folder to load into the Tile Editor.

4.2 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.
    - The required changes for this project relate only to the application behaviour. You should not need to modify any code pertaining to the GUI.
- In Test mode, the Editor will start playing the game levels in a folder. When checking and playing the game, you need to output the data as per the original behaviour, as well as any new log messages for checking the validity of the level and the game, following the above-mentioned format.
- Image files of characters and items are provided in the 'sprites' folder. The file names of these image files must not be changed.

4.3 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()".**
- Your app should look for the data folder (editor icons), sprites folder (pacman images), and the property file all in the folder in which the app is executed (the working directory).
- You must not change the format of the map files or names of the attributes in the property file. If you add a new attribute in the property file, it needs to have a default value as we will not set it in testing your submission 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.

5 Project Deliverables

5.1 Your version of the editor for PacMan in the TorusVerse

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.

5.2 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 patterns and principles (GoF and GRASP). Specifically, your report should address the following points:

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

- (P1) D**esign of editor**: Describe and justify your design for the editor/tester for PacMan in the TorusVerse, including the checking element.
- (P2) **Design of autoplayer:** Assume that eating an ice cube freezes the monsters for a defined duration. Describe how to use and/or modify your autoplayer design (information and interfaces, not the algorithms) to have autoplayer work in the presence of monsters and pills. Use diagrams as appropriate to support this description.

5.3 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 sub-diagrams or provide additional diagrams where appropriate.

- A domain class diagram for capturing the covering the domain concepts relating to the autoplayer and game levels/maps for PacMan in the TorusVerse.
- A static design model (i.e., a design class diagram) for documenting your design relating to the autoplayer and game levels/maps for PacMan in the TorusVerse.
- Additional static and/or dynamic design models to support P1 and P2 above as you judge useful and appropriate.

6 Submission

All project deliverables should be included in the <u>one</u> project zip file with the specified structure (see Figure 3). **<u>Do not</u>** include the .git folder in your submission. **<u>Only one member</u>** 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.

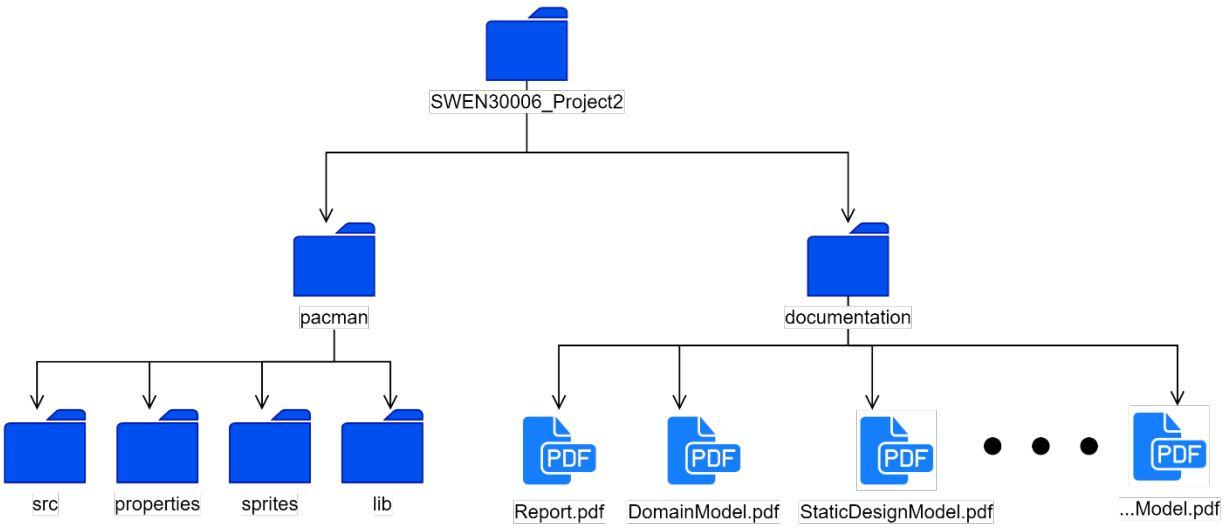

Figure 3: File structure for project submission