SWEN30006 Software Modelling and Design
Project 1: Ore Mining Simulator
- Project Specification -

School of Computing and Information Systems
University of Melbourne
Semester 1, 2024

Background

OreStrike Corporation have been a beacon of innovation in the realm of mining technology. Their engineers and scientists have toiled tirelessly to push the boundaries of what was possible in the mining industry. They have a vision to revolutionize the industry and make mining safer, more efficient, and more sustainable through autonomous mining.

OreStrike hired a software company to develop a mining simulator called "***OreSim***" to safely test and explore autonomous mining strategies. Unfortunately, the software company was dissolved due to the COVID-19 pandemic, leaving behind an early stage prototype of OreSim with basic functionality. The software lacks proper design and was hastily constructed. The software documentation is limited, comprising only code comments and a partial design presented in the form of a class diagram. OreStrike has recruited you and your team to make OreStrike's vision come true. You are tasked with extending the simple version of the OreSim to include necessary features as well as improving the existing design and codebase to better support future extension.

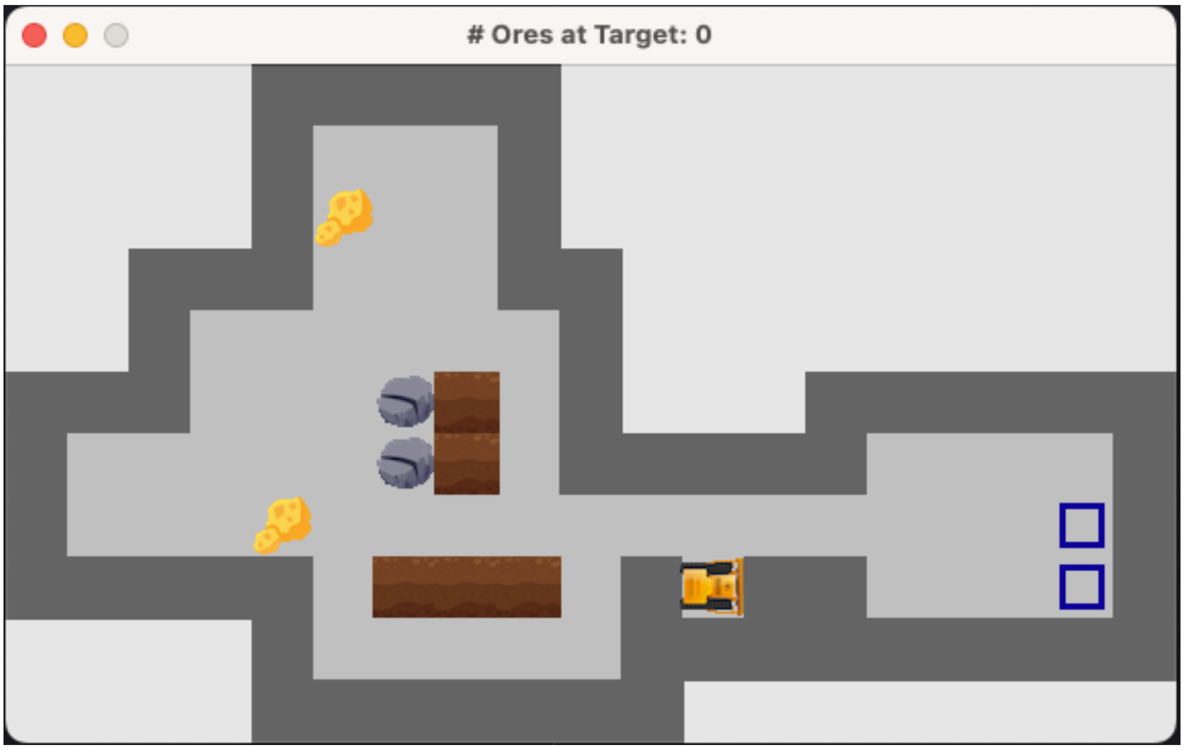


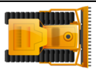
Figure 1: The GUI of the simple version of OreSim (Map 0)

1. Use Cases:

1.1 Ore Collection

This section provides the use case text describing a simple version of OreSim that is already built and provided in the codebase. The goal of the simulation is to control machines to collect all the ore pieces to put into designated targets. Table 1 provides a brief description of simulation elements.

Table 1. Description of landscape elements and a machine used in the simulation.

Graphic	Name	Description
	Ore	A valuable mineral that is found in various locations throughout the mining map.
	Target	A collection area for a piece of ore.
	Ore Cart	A cart for ore that has been collected, i.e., ore is in the target location.
	Rock	A hard and solid material that occurs in various locations throughout the mining map.
	Clay	A type of sedimentary material that can be found in various locations throughout the mining map.
	Ore Pusher	A machine that is designed to transport ore by pushing it, capable of moving only one piece of ore at a time. It cannot bypass rocks, clay, and other machines in its path.

Main Success Scenario (Manual Operation):

1. A miner opens the simulator which initiates and places mining machines, ores, obstacles, and targets into the simulation map. See Figure 1 for an example of the map.
2. The miner controls the ore pusher around the mining map to find an ore piece using the arrow keys on the keyboard (←↑↓→).
3. The ore pusher moves one cell (a distance unit of the map) at a time in the corresponding direction.
4. Upon reaching an ore on the map, the ore pusher pushes the ore toward the same direction as the pusher moves. The ore can only be pushed forward and cannot be pulled back.
5. The ore pusher pushes an ore until the ore reaches the target location, the simulator displays an ore cart, indicating successful collection.

   Repeat step 2 – 5.

6. When all targets are filled with ores, the simulator concludes the operation.

Alternate Scenarios

2a., (Auto Operation) If the simulator is configured the movement mode from "**manual**" to "**auto**", the ore pusher moves based on the pre-defined sequence of movement without using arrow keys. The simulator follows the pre-defined sequence of movement provided by the miner to operate the ore pusher. The pre-defined sequence is in the format of "machine type-direction" (one cell per move). For example, given a sequence of "P-U, P-L, P-D, P-R", where **P** denotes to an ore pusher and **U, L, D, R** denote to the four directions of movements, i.e., upward, left, downward, and right, respectively.

3a., 4a. (Obstacle Reaction) For both manual and auto operations, if there is an obstacle (a rock or clay) blocking the movement direction of the machine, the machine will not move.

*. At any stage, the miner can exit the simulation.

1.2 Additional Features

This section describes the additional features that need to be developed. OreStrike requests the following features to complete basic functionality of OreSim and make the system ready for future extensions.

Feature 1: Additional Mining Machines

1.i., According to the map configuration, additional mining machines (i.e., an excavator or a bulldozer) can be placed into the map. One machine per type will be used in this version. See Figure 2 for an example.

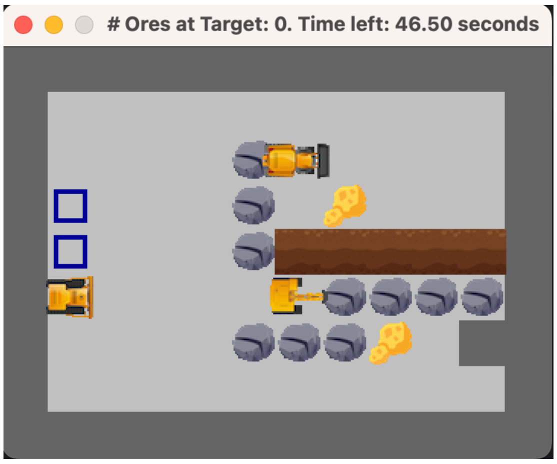
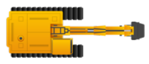
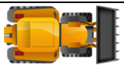


Figure 2: The GUI of the simple version of OreSim (Map 1) with additional types of machines

Table 2. Description of new machines used in the simulation.

Graphic	**Name**	**Description**
	Excavator	A machine that is designed to remove *rock* so that an ore pusher can move pass the rock's location. It cannot bypass ore, clay, and other machines in its path.
	Bulldozer	A machine that is designed to clear *clay* so that an ore pusher can move pass the clay's location. It cannot bypass ore, rock, and other machines in its path.

(Machine Operations)

2.i. When a movement mode is configured as "**manual**", these two types will not move and cannot be controlled by the miner.

2a.i Similar to an ore pusher, when a movement mode is configured as "**auto**", the simulator follows the pre-defined sequence of movement to operate machines, where **E** denotes to an excavator and **B** denotes to a bulldozer. For example, "E-U, B-D, P-U" means move an excavator up by one cell, then move a bulldozer down by one cell, and then move a pusher up by one cell. These two types of machines can move in four directions like an ore pusher.

(Obstacle Reactions)

3a.i, 4a.i The machines remove a corresponding type of obstacles by moving onto the locations of an obstacle. In particular, the simulator should operate as follow:

- If an excavator moves onto a location occupied by a rock, the rock disappears, and the path is cleared. If ore or clay block the movement direction of the excavator, the excavator will not move.
- If a bulldozer moves onto a location occupied by clay, the clay disappears, and the path is cleared. If ore or rock block the movement direction of the bulldozer, the bulldozer will not move.

Feature 2: Operational Statistics
OreStrike wants the operational data to be recorded for future analysis. The following statistics (stat) should be presentedrecorded at the end of the simulation.

1. The total number of moves of each of the machines.
2. The total number of obstacles (e.g., rocks, clay) that each of the machine removes.

The stat will be recorded in the `statistics.txt` file with the following format:

Format	Example

`Pusher-<ID> Moves: <total number>`
`Excavator-<ID> Moves: <total number>`
`Excavator-<ID> Rock removed: <total number>`
`Bulldozer-<ID> Moves: <total number>`
`Bulldozer-<ID> Clay removed: <total number>`
	`Pusher-1 Moves: 10`
	`Excavator-1 Moves: 5`
	`Excavator-1 Rock removed: 3`
	`Bulldozer-1 Moves: 2`
	`Bulldozer-1 Clay removed: 1`

In this extended version, only one machine per type is used. Therefore, the <ID> will always be 1. The stat will be recorded only when the simulation completes (all targets are filled with ores), or the simulation runs over the time limit specified in a configuration.

1.3 Future Development Plan
This section describes an extension plan of your OreSim for future development. You are **not tasked** to implement the following requirements, but ***your design of OreSim should support this extension*** that will be developed in the future***.*** You will need to discuss this in the report (See Section 4.2).

OreStrike has a tentative plan to extend OreSim from your design as follow:

1. Multiple machines with different IDs used in the simulation instead of one machine per type. The stat tracker should report the operational data for each machine used in the simulation.
2. More types of machines with different capabilities, e.g, haul trucks, crusher.
3. More types of obstacles.
4. Various ways of controlling machines. For example, for a manual mode, a remote-control stick may be used instead of using arrow keys. For an auto mode, the machines are controlled by an autonomous planner which generates optimal movement sequences of the machines; or even control to the machine directly.

2. Your Task

2.1 Refactor the existing codebase
OreStrike found that the simple version of the OreSim has a poor design. 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. *Your new design should facilitate the extension of new features described in Section 1.2 and the future plan described in Section 1.3.* 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.
OreStrike would like OreSim to include the new features described in Section 1.2. Nevertheless, OreStrike needs to ensure that your refactoring still preserves the original simulation's behaviour described in Section 1.1, an ore pusher can be controlled by manually using arrow keys or move automatically according to the pre-defined sequence. Examples of properties files in the **properties** folder include all necessary configurable parameters are provided in the base package.

<u>Expected System Behaviour:</u> OreStrike acknowledges that the descriptions of the new features may not cover all the possible cases of the simulation logic. As long as the described logic is in place, OreStrike is open to your ideas to handle the corner cases or edge cases that are not covered by the provided use cases.

3. The Base Package

3.1 Getting started

- The package is provided as an IntelliJ project in a GitHub repository: https://classroom.github.com/a/em9_mH1d
- To access the base package and set up, you can follow the instructions in Workshop 1. Note that the project requires at least **Java 19**.
- <u>Build the system</u>: We use Gradle to build, run, and test the project. This is the same structure as the Workshop 0 Part 2. The Gradle script that we have set up for you to run the project is already provided in the package. When the project is cloned in the local machine, IntelliJ should automatically set up the project based on the provided Gradle script.
- <u>Run the sysmtem</u>: The entry point for running the project is in the **Driver** class is an. To run the code, you can click on the Green Triangle in the Driver class. Then, you will see the GUI appear and can control an ore pusher in the OreSim using the standard keyboard actions.

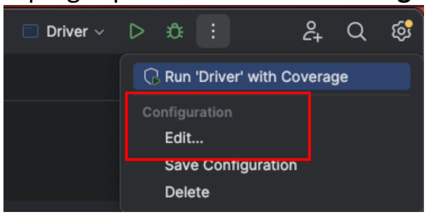
3.2 System design

- The classes in the '**app/src/main/java'** folder primarily handle the simulation 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 simulation behaviour. You should not need to modify any code pertaining to the GUI.
  - Image files of characters and items are provided in the 'sprites' folder (app/src/main/resources/spites). You should not need to modify these files.
  - Implementation Hints: The JGameGrid library has provided many useful functions that you can reuse to implement the proposed extensions. The codebase should also provide examples of using functions of the JGameGrid library.
- A partial class diagram of the simple version has been provided to help understand the current design (see **Appendix 1**).
- <u>Configuration:</u> The system reads a properties file that specifies the simulation configuration. We have provided three properties files in the '**app/src/main/resources/properties'** folder as samples for development (gameX.properties). You can change the properties file by changing value of the `DEFAULT_PROPERTIES_PATH` variable in the Driver class; and/or edit the properties in these files for your own testing but make sure to preserve the default behaviour.
- For more details about the structure of the code base, please refer to **Appendix 2** in this document.

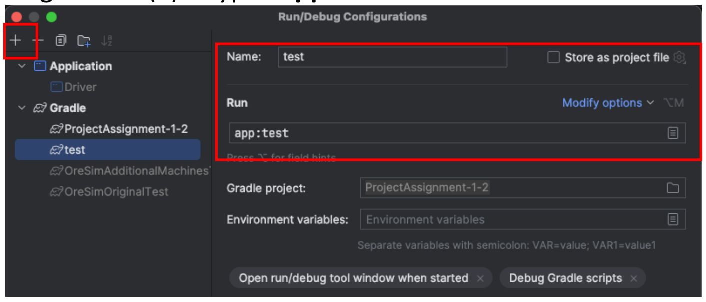
3.3 Testing your solution

- We will be testing your application programmatically, so we need to be able to build and run your program with a Gradle build script.
- We use the 'logResult' string generated in the OreSim class for testing whether your extended version preserves the original behaviour and the additional features have expected behaviour as described in the specification. You need to ensure that logResult generated by your extended version can pass the testing that we provide.
  - Samples of logResults that the system needs to generate are provided. You log needs to follow the format but does not need to match the sample log 100%.

- To ensure that we can test your extended version in our machine, you should follow the following guideline:
    - You should use the Gradle script that we have set up for you to run the project.
    - The following parts in the codebase **must remain unchanged**. Any changes made to these parts will be discarded during our testing (i.e., we will overwrite all your changes with our default values when testing is executed). It means that any modification made these parts that make your system works in your local machine, may not work in our testing machine.
        - The entry point at Driver and the Driver file needs to remain at `src/main/java`**.**
        - The maps (the values of map_0 and map_1 variables in MapGrid.java) should be not altered.
        - The file names of the image files under the app/src/main/resources/spites folder.
        - The whole test package (app/src/test) including the properties files in testing (testX.properties)
        - The build.gradle file
        - The settings.gradle file
- Two properties files are provided under the test package (app/src/test) as the key resources for you testing your extended version (testX.properties). Initially, your code will pass test cases for Original behaviour and fails test cases for AdditionalMachines and Statistics behaviours. You need to implement the requirement and pass all test cases. Your code needs to pass the automatic tests as well as the manual tests that the teaching staff will perform on your final submission. For manual tests, we will run the app with all the additional machines and run the simulation as a normal end user.
- <u>Testing on GitHub:</u> When you push your code to the repository of GitHub assignment, you can use the **GitHub action** to build and run the tests. Refer to the instruction in Workshop 0 Part 2 on how to trigger the build and tests on GitHub action. However, there may be a delay due to a large volume of build requests.
- <u>Testing locally</u>: You can also test your system on your local machine by setting up the Run configurations on IntelliJ IDE.
    - Select the option on the top right panel > select **edit configuration.**



    - Add a new configuration (+) > type **"app:test"** in the Run command.



Name: test

Run

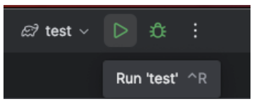
app:test

Gradle project: ProjectAssignment-1-2

Environment variables: Environment variables

Separate variables with semicolon: VAR=value; VAR1=value1

    - Then, you should be able to run the test with a new configuration by clicking the green triangle on the top right. The test outcome will be shown in the run panel.



4. Project Deliverables

4.1 Extended version of OreSim: Submit your whole project (with any/all libraries used included, with all the Gradle structure and Gradle setting files)

- Ensure that your code is well documented and includes your team's name and team members in all changed or new source code files.

4.2 Report: A design analysis report detailing your design analysis and how you improve and extend the system. The design analysis should include identifying design alternatives and justifying your design decisions with respect to design principles. The discussion should be *specific:* referring to a particular principle/pattern design and point out a specific part of source code, e.g., class or functions. See examples of reports in the submission page. 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.
- **(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 the extended version**: Describe and justify your design for the new features that are added into your new 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. You can use sub-diagrams or provide additional diagrams where appropriate.

- A domain class diagram for capturing the noteworthy concepts and their associations of OreSim including the extensions.
- A static design model (i.e., a design class diagram) for documenting your ***new*** design for OreSim including the extensions.
- A dynamic design model (i.e., a sequence diagram) for documenting the dynamic behaviour of OreSim in the scenario where OreSim operates an excavator based on this pre-defined sequence in Map 1 shown in Figure 2: "**E-U,E-R,E-L**", assuming that the excavator is in the initial location as shown in the figure.

5. Submission

All project deliverables should be included in the 1 project zip file with the same structure as the project when we are giving you (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 rubric 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.

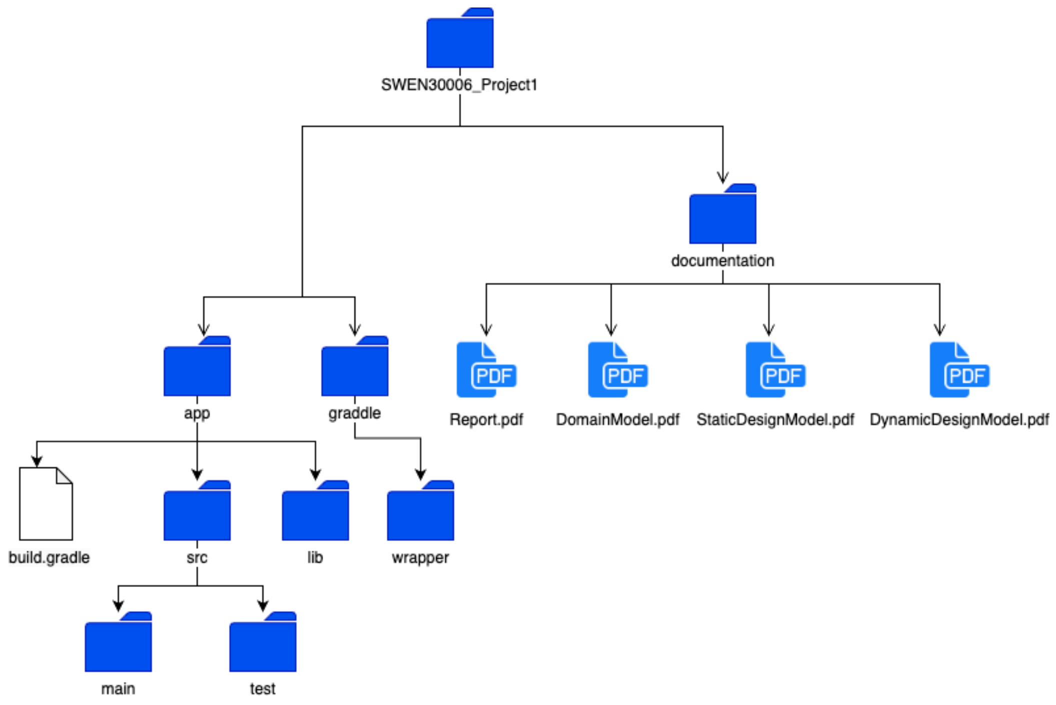
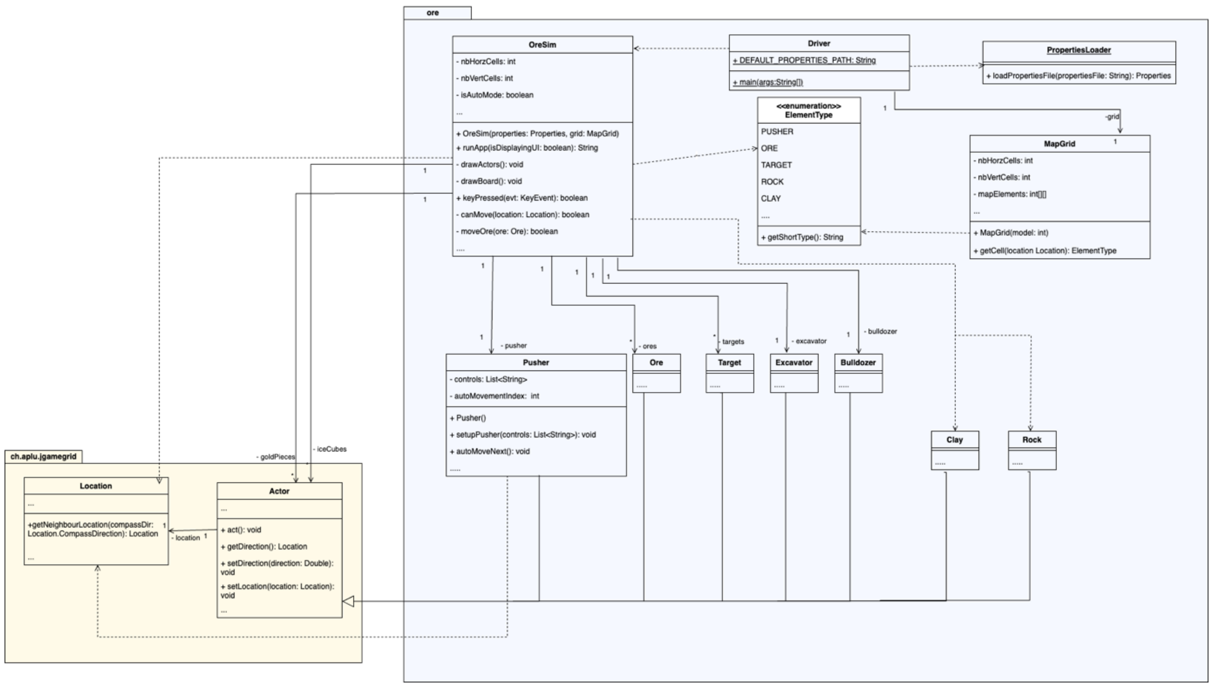


Figure 3: File structure for project submission

Appendix 1: Partial Class Diagram for the Current Version



Appendix 2: File Structure Explanations

This appendix provides some key information about file structure and the Gradle build for your understanding. You are not expected to change anything outside of your development directory and documentation directory:

File directory explanation:

- **`app/src/main/java` contains the development code that require your changes**
- `app/src/main/resources`: contains the properties files and images
- `app/src/test/java`: contains the testing code
- `app/src/test/resources`: contains the properties files and images for testing code
-

Gradle set up explanation:

- `settings.gradle`: is a Gradle setting file to set up the name of the project and the main development directory
- `app/build.gradle` is a Gradle setting file to set up the dependencies, the language version and the main class
- `app/src/test/resources`: where the properties files and images of testing code are