CSED332 Assignment 1

Due Saturday, September 21th

Objectives

- Use Maven
- Learn Java programming language

IntelliJ IDEA and Maven

- Download the attached file `homework1.zip`, which contains two subdirectores `problem1` and `problem2`. Each of them can be imported as a separate project into IntelliJ IDEA.
- In this assignment, your code need to be compiled using only Maven in a command line. You can use IntelliJ IDEA, but your code will be graded using Maven.
- To compile your code using Maven, you can go to where your `pom.xml` is for each problem, and execute the command `mvn compile`. You can run `mvn test` to run test cases.
- You may need to install Maven to run it in a command line. For download instructions and tutorials, we refer to `https://maven.apache.org`.

Problem 1

- The goal is to create a simple account management system for banking as follows. Check more detailed information in the skeleton code.
  - A *bank* manages two kinds of *accounts*: *high-interest* and *low-interest* accounts, where a high-interest account has a minimum balance.
  - Each account has an account number and an owner. You can deposit or withdraw money from an account, and transfer money from one account to another account.
- The `src/main` directory contains the skeleton code. You should implement all methods with *TODO* in the following classes:
  - `Bank`
  - `HighInterestAccount`
  - `LowInterestAccount`
- The `src/test` directory contains some test methods in `BankTest.java`. You can run the test cases by running `mvn test`.
- Your code will be graded by Maven, using extra test cases (different from `BankTest.java`) written by teaching staff. Make sure the command `mvn test` works.
- Do not modify the existing interfaces, the class names, and the signatures of the public methods. You can add more methods or classes if you want.

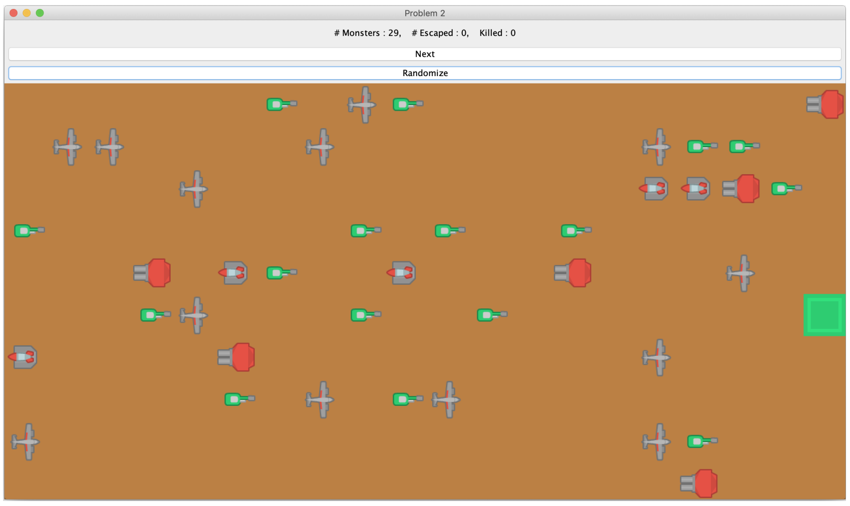

Figure 1: Game GUI

Problem 2

- The goal is to implement a mini tower-defense game. As shown in the following figures, a *game board* contains a number of *monsters* and *towers*.
  - A game board consists of $n \times m$ tiles, and the position of each tile is given by a pair (x, y), where $0 \leq x < n$ and $0 \leq y < m$.
  - Monsters moves towards the goal tile (marked as a green square). There are two types of monsters: *ground monsters* and *flying monsters*.
  - Towers attack (or kill) adjacent monsters. There are two types of towers: *air towers* only attack flying monsters, and *ground towers* only attack ground monsters.
  - Two flying monsters cannot be on the same tile. Similarly, two ground objects (ground towers, air towers, or ground monsters) cannot be on the same tile.
- A game consists of a number of rounds. For each round, monsters and towers perform actions as follows, and the game proceeds to the next round.
  1. All monsters on the goal tile "escape" from the game board.
  2. Each tower attacks (and removes) adjacent monsters of its type. E.g., an air tower at $(2, 3)$ attacks air monsters in adjacent tiles $(1, 3)$, $(3, 3)$, $(2, 2)$, and $(2, 4)$.
  3. Other remaining monsters (neither escaped nor attacked) can move to an adjacent tile, *according to some strategy defined by students (i.e., you).*
- In this assignment, you implement an object-oriented "model" of the game described above, given by the following classes.
  - `GroundTower` and `AirTower` implements the interface `Tower`, with the method `attack` that returns the set of monsters to be attacked by this tower in the current round.
  - `GroundMob` and `AirMob` implements the interface `Monster`, with the method `move` that returns an adjacent position to be moved for the next round.
  - `GameBoard` maintains a set of towers and monsters, and contains several methods for the game, such as `step` that runs a single round of the game.

- A simple monster AI will be defined and implemented by students as a part of this assignment (the `move` method of `GroundMob` and `AirMob`).
  - The game should not fall into error states (e.g., two ground objects are on the same tile) by monster movements. See the method `isValid` of `GameBoard`.
  - Each monster should try to reach the goal tile as much as possible, while avoiding being attacked by towers, without falling into error states.
  - We will run your strategy with respect to several game boards for grading. Students developing better strategies will receive additional bonus points.
- We provide a simple GUI in Figure 1 that runs the game on your model. It will not be used for grading, but you can inspect your implementation using this GUI.
  - The state of `GameBoard` is displayed in the GUI panel.
  - The Next button runs the method `step` of `GameBoard`.
  - If the game falls into an error state, Next will be disabled.
  - The Randomize button generates a new random game.
- The `src/main` directory contains the skeleton code. You should implement all classes and methods with *TODO* in the above classes.
- The `src/test` directory contains some test methods in `GameTest.java`. Your code will be graded by Maven, using extra test cases written by teaching staff.
- The command `mvn package` will create a jar file in the `target` directory, which can be executed using the command: `java -jar problem2-1.0-SNAPSHOT-shaded.jar`
- Do not modify the existing interface, the class names, and the signatures of the public methods, as they are used for the GUI and for grading.
- You can add more methods or classes if you want. In particular, you may want to add (abstract) superclasses for towers and monsters to avoid duplicate code.

Turning in

1. Create a private project with name `homework1` in `https://csed332.postech.ac.kr`, and clone the project on your machine.
2. Commit your changes in your `homework1` project that includes two directories `problem1` and `problem2`, and push them to the remote repository.
3. Tag your project with "submitted" and submit your homework. We will use the tagged version of your project for grading.

   (see `https://docs.gitlab.com/ee/university/training/topics/tags.html`)

Java Reference

- Java Language Specification: `https://docs.oracle.com/javase/specs/`
- Beginning Java 9 Fundamentals 2nd by Kishori Sharan, Apress, 2017 (available online at the POSTECH digital library `http://library.postech.ac.kr`)