CMPT 213 Assignment 3

Due: Nov 1, 11:59:59pm.

Overview

In this assignment, you are to create a Tokimon finder game. You are a Tokimon trainer who is trying to collect all Tokimons within a 10 cell by 10 cell game grid. You have no knowledge of what is in each unvisited cell and at each turn you will make a move to explore a new location on the grid. The grid can be occupied by (i) a Tokimon, (ii) a Fokimon or (iii) nothing. Fokimons are evil bird-like creatures who purpose is to harm Tokimon trainers. If you end up on a cell occupied by a Fokimon, you lose the game!

Game play Requirements:

- Your game should start by accepting anywhere from 0 to 3 arguments to `main()`. The arguments to the program will provide options for gameplay, the options are:
  - `--numToki=X` (where X is a positive integer >= 5): this determines the number of Tokimons in the 10x10 grid
  - `--numFoki=X` (where X is a positive integer >= 5): this determines the number of Fokimons in the 10x10 grid
  - `--cheat` puts program into cheat mode. See below.
  - If the number of Tokimons is not specified, use a default value of 10. If the number of Fokimons is not specified, use a default of 5. For example, assuming the `main()` method is located in `TokimonFinder` (see Technical requirements):

    ```
    > java TokimonFinder --numToki=20 --numFoki=6
    ```

    will specify 20 Tokimons and 6 Fokimons on the game grid. You should provide some basic error checking for the options.
- The program will then randomly choose positions on the grid to put the Tokimons and Fokimons.
- The game grid has the columns numbered 1, 2, … and has the rows lettered A, B, …
- The program will begin by asking user for an initial position:
  - For example, B5, and then enter.
- The player will also begin with 3 spells. Using a spell can either:
  1. Jump the player to another grid location
  2. Randomly reveal the location of one of the Tokimons or
  3. Randomly kill off one of the Fokimons

  It is up to you how you would like to do this.
- At each turn, the player is prompted for the next move, the player can choose to
  1. Move up, down, left, or right from their current position (key W, A, S, or D) or
  2. Use a spell

  Choosing option 2 will result in further prompting as described in the previous point.
- If a move results in the player landing on a cell occupied by a Fokimon, the game will be over, and the player loses.
- If a move results in the player landing on a cell occupied by a Tokimon, the player should be notified and congratulated.
- After each turn, the player is shown the number of Tokimons they have collected, the number of Tokimons remaining, and the number of spells remaining.
- At each turn, the player is shown a map of what is known about the game-grid so far:
  - ~ indicates unknown (unvisited)
  - $ indicates a found Tokimon
  - ‘’ (space) indicates a visited but empty location.
  - @ player’s current position
- A player wins when all Tokimons on the board have been collected.

- When the game ends, the player is shown the complete board including X's to indicate the positions of the Fokimons.
- CHEAT MODE: if the option `--cheat` was included in the `main()` call, the program should show the user the full board (including the positions of all Tokimons and Fokimons) before starting the game.

Technical Requirements:

- Main class should be in a file called `TokimonFinder.java`
- Must exhibit good OOD principles:
  - You must have two packages: One package for the UI related class(es); another package for the model related classes (actual game logic). Imagine that you wanted to have not only a text game, but also a web version. You should be able to reuse the entire model (game logic) in a completely different UI.
  - Each class is responsible for one thing.
  - Reasonably detailed break-out of classes to handle responsibilities.
  - Each class demonstrates correct encapsulation.
  - Consider use of immutable classes where applicable.
  - Respect the command/query separation guideline when appropriate.
- The game is to use a text interface for display, and the keyboard for input.
- Implementation must follow the online style guide. Specifically, important are:
  - Good class, method, field, and variable names.
  - Correct use of named constants.
  - JavaDoc - Good class-level comments (comment on the purpose of each class).
  - Clear logic.
- OOD Hint:
  - When you have some complex state, it is often best to encapsulate it into an object. Consider having an object for storing the state of each cell of your game-grid. Store a group of these to makeup the game grid.

Design:

Complete the following steps to create an object-oriented design for this application.

1. Use case
   - Create a use case for the game. Hint: it will be titled "Play game".
   - Provide a reasonable list of steps for playing the game from the user's perspective. For example, how to recover from incorrect user input without crashing (use a Use Case variation).
   - This must be typed on a computer and submitted as a text or PDF file named USECASE.TXT (or .PDF). Place this file in the docs/ folder of your project (you will need to create this folder).
2. CRC Cards
   - Create CRC cards to come up with an initial object-oriented design.
   - Do not submit the actual cards, but once you have settled on a design, you must type up the information stored on the CRC cards, or take a picture of the cards.
   - Each card must show the class name, responsibilities, and collaborators. Submit this as a .txt, .pdf, or .jpg/.png file named CRC.TXT (or .PDF,...) in the docs/ folder. If you take a picture, ensure that the text is clear, and that the image is less than one MB.
3. UML Class Diagram
   - Create an electronic UML class diagram for your OO design.
   - The diagram should not be a complete specification of the system, but rather contain enough information to express the important details of your design.
   - Your diagram must include the major classes, all class relationships, and some key methods or fields that explain how the classes will support their responsibilities.

- You must use a computer tool to create the diagram. We will discuss some free software for doing this 😊. You may *not* generate the diagram directly from your Java code.
- Submit your UML diagram as a PDF or image file named CLASSDIAGRAM.PDF (or .PNG, or .JPG, …) in the docs/ folder.

Implementation:

- Implement the game in Java.
- You must implement according to your OOD from the Design Phase.
- Source Control: Although not a requirement, you may find it helpful (especially if you'd like to use this for your portfolio).
    - If you'd like to develop using GIT, you can use SFU's GitLab: https://csil-git1.cs.surrey.sfu.ca/
    - (**Do not use GitHub unless you keep your projects closed source**; do not share your code publicly during the course, otherwise it counts as academic dishonesty).

Marking Scheme:

Phase 1: Design - Total [15] Marks

- [3] Use case - Reasonable description of the steps for playing the game from the user's point of view.
- [3] CRC - Reasonable break-out of classes and high-lever responsibilities. List major class collaborators.
- [9] UML Class Diagram - Clear OOD, Correct and meaningful class relationships. Some methods and/or fields listed

Phase 2: Implementation - Total [35] Marks

- [4] Correctly handle command line arguments
- [3] Game-grid display
- [3] User interaction handles errors
- [15] Correct game-play handling player moves
- [5] Correct handling of the remaining workflow & win/loss.
- [3] Game polish: Clear messages, nice layout.
- [2] Creativity

Correctly follow coding style guide. - Total [0] Marks

- [-8] Up to 8 point max deductions

Submission

Submit a zip file of your project (according to the directions outlined in the assignments link of the course website) to the coursys server. https://courses.cs.sfu.ca/. Your project must generate a JAR file as part of the build process.

Please note: all submissions are automatically checked for similarities of all other submissions on the server.

In addition, you must submit your files from the design phase:

1. USECASE.TXT (or .PDF)
2. CRC.TXT (or .PDF, or JPG)
3. CLASSDIAGRAM.PDF (or PNG, or .JPG)

Sample game grids:

These are just samples, you do not have to follow this convention. In fact, **please try to come up with more interesting UI**.

1. During gameplay, when a Tokimon is discovered, after a few cells have been visited.

```
Game Grid:
      1  2  3  4  5  6  7  8  9 10
   A  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   B  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   C  ~  ~     ~  ~  ~  ~  ~  ~  ~
   D  ~  @$    ~  ~  ~  ~  ~  ~  ~
   E  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   F  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   G  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   H  ~  ~  ~  ~           ~  ~  ~
   I  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
   J  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
```

2. When you lose:

```
Game Grid:
      1  2  3  4  5  6  7  8  9 10
   A
   B                  $
   C  $     X  X
   D
   E  $     @X             $
   F
   G                  $  $  X
   H     $  $                $  $
   I     X
   J              X  $
```

3. When you win:

```
Game Grid:
      1  2  3  4  5  6  7  8  9 10
   A
   B                  $     X
   C  $
   D
   E        X              @$
   F         X
   G                  $  $  X
   H           $  $                 $
   I     $
   J     $        X  $
```