APCS Problem Set 13: `MineSweeper`

Michael Ferraro
<mferraro@balstaff.org>
Balboa High School
29 February 2016

1 Introduction

Now that you've studied the majority of the topics required for the AP Computer Science exam, you are going to have the chance to show off how much you've learned — by creating a game with a software development team. Team members will have defined roles and need to work well together in order to meet a deadline (the due date).

Your task is to create a console-based game similar to Mine Sweeper, which is commonly bundled with Microsoft operating systems.

1.1 Objectives

By completing this problem set, students will learn to...

- design a software system involving numerous parts;
- approach design problems in two ways: top-down and bottom-up;
- work within a role-based team to develop a software product;
- leverage a version control system to collaborate with other developers; and
- write and perform tests on software components.

1.2 Due Date

The deliverables for this problem set are due by the **start of class on 14 March 2016**.

2 Prerequisite Research

A good primer for `MineSweeper` may be found at `https://www.youtube.com/watch?v=LHY8NKj3RKs`. If you've never played `MineSweeper`, make sure you do the necessary background research by playing.

- On a Windows PC that has the game installed, go to
  `Start` → `Programs` → `Accessories` → `Games` → `Mine Sweeper`.
- An online version of the game, written in Javascript, is available here:
  `http://www.chezpoor.com/minesweeper/minesweeper.html`.

3 Description of APCS `MineSweeper`

As you read this section, either highlight important pieces of information or take notes. You won't want to forget about any details that you read here before you design and build the product. Otherwise, it might be very hard to change your application to have an unplanned feature later! This is the Computer Science analog of the old carpenter's saying "Measure twice, cut once."

Our `MineSweeper` will be console driven (i.e., without a GUI). It must still comply with the MVC [Model-View-Control] design pattern that was covered in Lesson 41 (see here). The reason is that we might decide to build a GUI later, so complying with MVC now will minimize the amount of code that we need to rewrite.

3.1 The Minefield

The minefield will be represented using characters printed to the console/screen. Each time the game's state changes (e.g., the player inspects a square, clearing it, which triggers the clearing of other squares in its vicinity), a method will be called to refresh the printed output representing the field.

When the game starts, the player is to be asked for a difficulty level, which determines the size of the minefield.

level	**field size**
beginner	8×8
intermediate	12×12
advanced	15×15

Regardless of difficulty level, ≈16% of the squares are to be mined. This value should be a **symbolic constant** that can be readily changed later if our game testers determine 16% to be inappropriate.

Every team must adopt the same square-naming strategy. The squares must be identified by alpha column and numeric row, as shown below in the sample 4×4 grid. Square `b3` contains an asterisk (`*`). Your `MineSweeper` game's board should resemble the sample grid.

```
    a   b   c   d
  +---+---+---+---+
1 |   |   |   |   |
  +---+---+---+---+
2 |   |   |   |   |
  +---+---+---+---+
3 |   | * |   |   |
  +---+---+---+---+
4 |   |   |   |   |
  +---+---+---+---+
```

The program is to track the player's progress from one game to the next. A suitable way of doing this is to always have the number of wins and losses shown in the UI. (This is like the "scoreboard" functionality in `Craps`.)

3.2 Game Behavior

When started, all squares are hidden, meaning we do not know whether there is a mine at any particular location. These are the outcomes when a square is inspected:

- A mine is found — game over, record a loss
- Otherwise,
  - there are no more unmined squares left — player wins, record a win
  - one or more mines is adjacent[1] to the current square; if so, show the number of adjacent mines.
  - no mines are near this one, so trigger a recursive call to some `inspect()` method on all adjacent squares.

3.3 Controlling the Game

At a prompt, the player will type commands, some of which take a single argument — a square's location. Below are sample commands and what each would mean.

command	interpretation
`i b3`	inspect (reveal) location `b3`
`f e1`	flag location `e1` so the player remembers not to inspect it later
`u c8`	unflag location `c8`
`q`	quit

Note that commands should not be case sensitive. And if a users types "`f  a2`", which has more than one space separating the command from the argument, it should be accepted as `f a2`. While extra work for the programmer, such features make the program more robust and less likely to fail.

4 Team Roles

Each team will have ≈5 members whose roles follow.

- **Architect** (1): Person responsible for other members' adherence to the design and tracking progress of the team members for the purpose of keeping the development efforts moving forward. Duties include:
  - documenting the team's design
  - maintaining regular communication with team members
  - conducting short daily team meetings to discuss progress, raise issues, etc.
  - communicating design changes to the team
  - keeping the git repository organized (see §6.3)
  - documenting tasks that team members are working on in a spreadsheet

[1] Note that *adjacent* squares include those on diagonals, so a square may have up to 8 adjacent squares, or as few as 3 (in the case of a corner).

  - resolving conflicts between team members and escalating issues to the teacher as needed
  - being the team's go-to person for technical issues

- **Developer** (≈3): Person responsible for implementing the classes specified by the application design and doing some very basic testing (i.e., making sure the classes compile). Duties include:
  - raising design issues to the architect
  - providing Javadoc[2] comments for all classes and their public methods
  - making sure that there's always a recent, compilable version of classes pushed to the shared git repository at `https://github.com` and notifying testers when classes are ready to be tested.
  - accepting feedback from testers regarding bugs and making the necessary changes.[3]

- **Tester** (1): Person responsible for certifying the correctness of the classes written by the developers. Responsibilities include:
  - understanding the APIs provided by the developers of each class
  - writing test classes to test the behavior of methods in *model* and *controller* classes in the app
  - communicating bugs back to the developer and following through to make sure the bugs are fixed
  - time permitting, provide suggested bug fixes to the developer
  - developing the end-user documentation that explains how to run and operate the game

5 Design Phase

Each team will spend time reviewing the game description and planning out how to model the various components using Java classes. After that, there will be a classwide discussion to decide how it would be best to model the game. The teams will then reconvene to refine their designs.

Questions to consider during the design phase include:

- What physical aspects of the game need to be represented?
- What information/state does each class need to maintain? → *fields*
- What actions need to be carried out by or on the class? → *methods*
- Which classes need to communicate with which others? → `HAS-A` *relationships, MVC*

[2]For instructions on how to create Javadoc API pages, refer to `src/edu/balboa/apcs/MineSweeper/api/README.txt`. See sample Javadoc comments here.

[3]Professional software development teams use products like Bugzilla to track bugs and progress toward fixing them. For this project, your team may opt to use a spreadsheet edited by the testers and developers for this purpose.

Part of making sure that the classes are planned out well is to conceive of every possible set of actions that may occur. For example, a player starts the game and chooses a difficulty level. What objects are created, what methods need to be called? What happens after a user is prompted for a command? There's a class that processes that command `String`, likely captured by a `Scanner` object. Depending on the command, certain other objects' methods will be called, etc. **Try to make sure your design is as complete as possible before your team starts writing code!**

Once your team is satisfied with the design, which might take more than a class period, make someone responsible for drawing a classes diagram (see §7.3). Other members should write *skeleton classes*, which are Java classes that have method headers but no written code (comments are OK, though).

6 Your Team's GitHub Repository

Rather than each team member having a private workspace and sharing files via email, USB stick, etc., your team will operate like professional developers with a shared `git` repository hosted at https://github.com. GitHub provides an always-available `git` service with a convenient web front-end (in case using `git` commands in the terminal shell is difficult or inconvenient).

6.1 Connecting to your team's repo

A `git` repository has been provisioned for your team. Only your team and teacher have access to it. You should consider bookmarking your team's repo URL, which will be one of the following:

5^{th} Period	6^{th} Period
https://github.com/BalboaAPCS1/MineSweeperA	https://github.com/BalboaAPCS2/MineSweeperP
https://github.com/BalboaAPCS1/MineSweeperB	https://github.com/BalboaAPCS2/MineSweeperQ
https://github.com/BalboaAPCS1/MineSweeperC	https://github.com/BalboaAPCS2/MineSweeperR
https://github.com/BalboaAPCS1/MineSweeperD	https://github.com/BalboaAPCS2/MineSweeperS
https://github.com/BalboaAPCS1/MineSweeperE	https://github.com/BalboaAPCS2/MineSweeperT

Follow these instructions to set up your Eclipse project:

1. From a terminal shell:
   (a) `cd ~/MOUNTED/apcs-locker/workspace3/`
   (b) `git clone` *`URL`* ← replace `URL` with one of the URLs in the table above
   (provide your github.com username and password when prompted)
2. From Eclipse:
   (a) Connect to your `workspace3` workspace.
   (b) Create a new Java project with **exactly** the same name as the repo you cloned — e.g., `MineSweeperP`.

6.2 Test adding and removing a file

Once you've gotten a locally cloned copy of your team's repo, and Eclipse can see it, let's do a test:

1. In Eclipse, right click the `docs` folder and add a new file: *my_username*`.txt`. Add a few lines of text to the file and save.
2. From the terminal shell:
   (a) `cd ~/MOUNTED/apcs-locker/workspace3/`*TEAM_NAME*`/docs/`
   (b) `git add` *my_username*`.txt`
   (c) `git status`

   should yield

   ```
   # On branch master
   # Changes to be committed:
   #   (use "git reset HEAD <file>..." to unstage)
   #
   # new file:   mferraro.txt    <== your filename will differ
   #
   ```

   (d) `git commit -m "Initial commit"`
   (e) `git push` ← could require you `pull` first in case teammates have pushed changes since you last updated your checked-out copy of the repo!
3. Visit your repo's web interface (see §6.1) and verify you see the new file in the `docs` directory.
4. Test removing a file from the repo from the terminal shell from the same directory as before:
   (a) `git rm` *my_username*`.txt`
   (b) `git commit -m "deleted test file"`
   (c) `git push`
5. Verify the web interface shows that your test file is gone from the `docs` directory.

6.3 Repo directory structure

```
<project root>
 |
 |-- docs
 |   |-- Architects              (a)
 |   |-- Design                  (b)
 |   \-- UserManual              (c)
 |
 \-- src
     \-- edu
         \-- balboa
             \-- apcs
                 \-- MineSweeper  (d)
                     |-- api      (e)
                     \-- tests    (f)
```

Details (deliverables in **bold**)

(a) information needed by architects

(b) **design document** and **classes diagram**

(c) **user manual** explaining how to run the app and play the game

(d) **application classes**

(e) **Javadoc files** for application classes

(f) **tester classes**

7 Deliverables

By the start of class on the due date, each deliverable must be present in the team's git repo in GitHub. Once deliverables are in place, the architect will tag the repo with a version number (`v1.0`).

7.1 Java Sources

All Java source files — both application classes and associated tester classes — must be in the proper directories as described in §6.3. Sources should be properly formatted and easy to read.

7.2 APIs

Javadoc API webpages must be produced for all classes in the application *except* the driver class and tester classes. Notes about using `javadoc` are in `api/README.txt` in your team's repo. A Javadoc sample has been provided here.

7.3 Design documents

Two documents need to be delivered in the `docs/Design` directory:

- `Architecture.md`: This document, to be completed by the architect, describes the application's design. Each class and its role in the app must be documented. In addition, how the classes communicate should be included.

  The file is in Markdown format. A basic document structure has already been provided though you might need additional Markdown tags. Refer to the official Markdown syntax reference or search online for Markdown tutorials.

- Classes diagram: Your team will need to produce a diagram of your application's classes. Consider using draw.io or Dia. You can find an example diagram on p152 of the Litvin textbook.

  Export your diagram to PDF format and add to the `docs/Design` directory in the git repo. Remember to use `git add, git commit -m "COMMENT", & git push` when adding the PDF file.

7.4 User manual

The user manual will reside in `docs/UserManual/UserManual.md`. A starter document has already been provided. This document, like `Architecture.md`, is in Markdown format. (See §7.3 for a link to the official Markdown reference.) Any screenshots or other images should be added to the `images` subdirectory and referenced in `UserManual.md`.

7.5 Tagging the repo

The architect will need to tag the git repo as version 1.0 (tag `v1.0`) to indicate the state of the repo to be graded. If another team member adds or modifies files after the tag, those changes will *not* be graded.

The architect should run `git pull` to ensure his/her copy of the repo is up to date. Once verifying that all deliverables are ready, the architect will tag the repo in this manner:

```
git tag v1.0
git push origin v1.0
```

When grading, your teacher will run this:

```
git clone <REPO_URL>
git checkout v1.0
```

8 Timeline

The entire duration of this project is just over two calendar weeks (9 class meetings). What follows is a rough timeline for the project, which may vary from team to team and is subject to change.

- **Day #1**
  - Students are introduced to the project. Teams are announced. Teams begin thinking about how the application should be designed.
  - Architects edit their team's `README.md` (in the repo root) to include their team's contact information.
  - For HW, each team member re-reads this project document and continues thinking about application design.
- **Day #2**
  - Architects hold check-in meetings with their teams to discuss current state of design. Teams continue designing the application.
  - A classwide design discussion is moderated by the teacher.
  - Developers are tasked with implementing specific elements of the design. Testers works with architects to continue refining the design.
- **Days #3–6**
  - Architects hold check-in meetings to go over current project status, have developers discuss challenges they are facing, etc.
  - Developers continue working on their parts of the app.
  - Testers write user manuals and tester classes.
  - Architects document the design and generate classes diagrams (or delegate the tasks to others).
- **Day #7**
  - Testing of classes and overall application is performed by testers and developers.
  - Developers generate Javadoc API webpages.

- **Day #8**
  - This is the last in-class work day!
  - Architects determines what tasks remain and delegate those remaining tasks to team members accordingly.
- **Day #9**
  - By the start of class, architects tag their repos as `v1.0` (see §7.5).
  - Teacher provides a GitHub URL for all students to clone that includes all teams' finished products.
  - Each team will play all other teams' applications and log identified bugs in a spreadsheet.

9 Assessment

Each member of a team will receive a team grade, with an optional ± component reflecting his/her own individual efforts (or lack thereof!). The project will be worth **250 points**, allocated as shown in the table below.

Points	Category
50	**Application:** Does the application run as specified, meeting all requirements?
40	**Design Doc:** Is the overall design described in detailed terms? Is each class' role explained? Are reasons for certain design decisions included? Is the document well organized and written?
40	**Tests:** Are the test cases sufficient in quantity and quality? Are all *model* classes well tested? Are all automated tests runnable via a single driver class?
30	**Sources:** Are the final source files neat, well commented, and well formatted? Are they correctly included in the repo directory structure?
30	**Team Management and Dynamics:** Was there good communication within the team? Was there a fair distribution of labor? Is there documentation of bugs and fixes? Were tasks tracked and recorded?
20	**Classes Diagram:** Are all classes included? Are the `HAS-A` relationships shown? If interfaces and/or abstract classes are used, are those and their implementing classes shown properly?
20	**APIs:** Are the APIs complete and accurate? Are they well written and sufficiently descriptive?
20	**User Manual:** Does the doc explain how the game is started and how to play? It is clearly written and well organized?