Homework #4: Scrabble with Stuff

In this assignment you will design and implement a variant of Scrabble$^{\text{TM}}$.[1] This variant of Scrabble, called *Scrabble with Stuff*, includes special tiles and features of your choice and should be playable by multiple players. We have included a description of *Scrabble with Stuff* below.

This assignment focuses on the design and implementation of a medium-sized program. The goals are for you to:

- Demonstrate a comprehensive design and development process including object-oriented analysis, object-oriented design, and implementation.
- Demonstrate the use of design goals to influence your design choices, assigning responsibilities carefully, using design patterns where appropriate, discussing trade-offs among alternative designs, and choosing an appropriate solution. The core logic of your solution must be testable and completely independent from your solution's eventual graphical user interface (GUI).
- Communicate design ideas clearly, including design documents that demonstrate fluency with the basic notation of UML class diagrams and interaction diagrams, the correct use of design vocabulary, and an appropriate level of formality in the specification of system behavior.
- Demonstrate basic fluency in GUI implementations, including an understanding of event handling and the Observer Pattern.

This assignment is much larger and more challenging than the previous assignments in this course. You will design and implement your game over the period of a month, with three main milestones:

- Milestone A: Object-oriented analysis and design (due October 11th)
- Milestone B: Core implementation and testing of game features (due October 18th)
- Milestone C: Full implementation with GUI (due October 25th)

Please start your work for future milestones early whenever possible. For example, you should ideally start your implementation (Milestone B) before the Milestone A deadline.

[1] Scrabble$^{\text{TM}}$ is a popular Words with Friends$^{\text{TM}}$-like board game (see the Wikipedia article for details).

The Game: Scrabble with Stuff v1.0

Like traditional Scrabble, players take turns placing letters (to form words) on a two-dimensional grid board, earning points for the words they place on the board. However, each player has a private view of her board and cannot view other players' tiles. To familiarize yourself with the rules and vocabulary of the game, read the Scrabble Wikipedia page or the official rule book. Your implementation should roughly follow the original rules, including features such as double letter score, triple letter score, double word score, and triple word score squares on the game board. Your game does not need to include blank letter tiles.

In *Scrabble with Stuff*, in addition to letter tiles of the classic game, a player may purchase *special tiles* with the points she has earned so far. These special tiles are added to the player's inventory of tiles, and may be played in a subsequent turn along with a word. Pre-existing Scrabble board squares (such as double letter score, etc.) may not be purchased like special tiles.

After placing a special tile, the tile is visible to the player who played it but is invisible to all other players in the game. A special tile is activated when a regular tile is placed on top of it. When activated, a special tile has an immediate or latent effect on the game, depending on its type.

You must implement at least five types of special tiles for this assignment:

- **Negative-points:** The word that activated this tile is scored negatively for the player who activated the tile; i.e., the player loses (rather than gains) the points for the played word.
- **Reverse-player-order:** The turn ends as usual, but after this tile is activated play continues in the reverse of the previous order.
- **Boom:** All tiles in a 3-tile radius on the board are removed from the board. Only surviving tiles are scored for this round.
- **Your own tile:** Implement a special tile of your choice. The exact effects of this special tile are up to you. Be creative!
- **Unknown tile:** We will reveal an additional required special tile (via a Piazza post) on October 25th, after the Milestone B late deadline. Your design does not need to explicitly include this tile, but you should try to anticipate this tile and require as few changes as possible when implementing it later.

Milestone A: Object-oriented analysis and design (due Oct 11th at 3:00 p.m.)

For this milestone you must analyze and design your game. You should focus on the game-related functionality of the program, but it might help to sketch a user's GUI to better understand how a user will interact with the game.

This milestone consists of the following tasks:

1. Describe the system's behavioral contract for when a player places a valid word on the board.
2. Respond to two design scenarios (given below) and provide at least one interaction diagram for each scenario.
3. Create an object model for your game.
4. Justify your design choices, describing the rationale for your design.
5. Get feedback from the course staff.

We describe each of these tasks below.

Behavioral contracts

Describe the behavioral contract for when a player places tiles during a game. This interaction might include giving the player additional tiles, updating their score, activating special tiles, etc. The contract should explicitly describe the preconditions and postconditions for the interaction, and your behavioral contract should be consistent with your domain model and interaction diagrams (below). You may provide explicit examples to clarify your contract (e.g, what happens when two special tiles are activated in the same turn).

Two design scenarios

Please formally address the two scenarios below. For each scenario you must visualize the interaction among objects in the game, creating at least one interaction diagram for each scenario. Your interaction diagram should be a UML sequence diagram or UML communication diagram, possibly accompanied by an additional textual description. You may include interaction diagrams and explanations for additional scenarios wherever helpful.

1. Describe how a move is validated when a player plays a set of tiles on the board.

   Validation likely involves checking whether tiles are placed in valid locations and whether they form actual words. This scenario likely starts by calling a method like `isValid` with a parameter representing a move on some abstraction of the game or one of its parts. This scenario ends by returning whether the move is valid.

2. Describe the actions in your game when a player places a set valid set of tiles on the board.

   This scenario likely starts by calling a method like `move` with a parameter representing a move on some abstraction of the game or one of its parts. Performing the move might or might not trigger the evaluation of special tiles, compute points, adjust the score, and trigger the next round in the game.

These two scenarios are just two examples of many scenarios worth considering as you are designing your game. Later in this section we provide a set of additional informal questions that can help you design your game. We expect a formal answer only for the two scenarios above, whereas you do not need to respond to the informal questions below. Nonetheless, the informal questions are worth considering and might help guide your design.

The object model

Create an object model of your game, documented as a UML class diagram. Your object model should demonstrate that you understand the intellectual distinctions between a domain model and an object model; the object model should describe the classes and interfaces of your design, as well as their key associations (with cardinalities), attributes, and methods. The objects and methods in your object model should correspond to the objects in your interaction diagrams above.

In Milestone C of this assignment you will implement a GUI for this game—and your game's methods will be called by the GUI—but your design here should be independent of any specific GUI implementation. Your object model should just model the *Scrabble with Stuff* game; do not include any GUI elements in your design.

Justify your design decisions

Write a short ($\leq$ 2 pages) description of your object model justifying your design choices. To support your justification, refer to design goals, principles and patterns where appropriate. To help formulate your justification, we recommend that you consider design decisions related to the two formal scenarios above and the informal questions that we provide later in this section.

Show your work to the course staff

It is our experience that software design is a deceptively simple task, but that good software design is very difficult and requires many iterations of design and revision. When you complete a programming assignment, the compiler (or auto-grader) provides instant feedback and you can run your program to easily evaluate the quality of your work. With software design, however, you don't have automated tools to provide feedback.

Instead, you should get feedback from more experienced software designers – such as the course staff – before you turn in your homework. In previous semesters, students who did not solicit feedback generated poor software designs, which scored poorly and made completing Milestones B and C needlessly difficult. You should avoid that outcome by getting early feedback from the course staff and (repeatedly) revising your design as necessary.

Turning in your work

Turn in the following files for Milestone A:

- `behavioral_contract.pdf`: The behavioral contract for a user placing tiles on the board.
- `interaction_validate.pdf`: Your response to our scenario about validating a move in your game.
- `interaction_move.pdf`: Your response to our scenario about executing a move in your game.

- `object.pdf`: The object model for your game.
- `rationale.pdf`: Your justification of your design decisions.
- `README.txt/README.md`: Any additional information about your Scrabble implementation. This file should include, at minimum, a description of the additional special tile you plan to implement.

We encourage you to discuss your design decisions with your peers. However, you must fully understand your design decisions and generate your own UML diagrams and rationale.

Some design hints

When you are done, *Scrabble with Stuff* will be a medium-sized program with numerous interactions between game components, some interactions of which are complex. Above we provide two concrete scenarios you should consider when designing your game, and you must submit a response describing how your design addresses those scenarios.

Here we provide additional helpful questions you should consider when creating your design. You do not need to formally respond to these questions, but we recommend that you consider them when planning your object model and interaction among game components. It might be helpful to create interaction diagrams for each question.

- How can a player interact with the game? What are possible actions a player can perform? How would someone start a game with multiple players?
- How is a player's action of placing multiple tiles represented?
- How are points calculated for placing a word at a specific location on the board? How are double letter score / double word score etc. squares evaluated? How can words be attached to existing words? How might special tiles affect the calculation of the score?
- How can a player acquire special tiles? When and how are special tiles placed on the board? How are special tiles triggered? What happens when special tiles are triggered? How can special tiles (current or future-planned) affect the game?

  Consider how special tiles observe and modify game state and which components have responsibility for updating game state. It is also helpful to consider the effect of words that trigger multiple special tiles, and whether special tiles can trigger or affect each other.

  You should design special tiles in a generic extensible way, such that special tiles may reuse code and that additional tiles can be added later.

- How are turns managed? Who is responsible for knowing who the current player is?

Again, you do not need to respond to these questions, but we believe that any good design will clearly address these issues.

A note on notation

To ease communication and avoid ambiguity, we expect all models to use UML notation for class, sequence, and/or communication diagrams. We do not require much formality, but we expect that relationships (such as association, inheritance, and aggregation) are described correctly in your diagrams, and that each relationship includes the cardinalities of the relationship. Attributes and methods should be specified correctly where appropriate, but we do not require precise description of visibility or types.

It is important that your models demonstrate an understanding of appropriate levels of abstraction. For example, your object model should not include highly-specific details such as getter and setter methods if those details do not aid the general understanding of your design.

UML contains notation for many advanced concepts, such as loops and conditions in interaction diagrams. You may use UML notation for these advanced concepts but we do not require you to do so. You may describe such concepts with your own notation or textual comments, as long as you clearly communicate your intent.

To maximize clarity, we strongly recommend that you draw UML diagrams with software tools. We do not require or recommend specific tools; you may share tool-related tips on Piazza.

Milestone B: Core implementation (due Oct 18th at 3:00 p.m.)

For this milestone you must complete four tasks related to your game implementation:

1. Implement the core features of your game.
2. Test the major components of your core implementation
3. Document your implementation.

We describe each of these tasks below.

Implementing the core features of your game

Implement the core features of your *Scrabble with Stuff* game. By "core features" we mean that your program should include all the functionality needed to play a *Scrabble with Stuff* game. Hence, the core implementation is only playable by calling a sequence of methods, not via any any user interface. Your implementation should be able to set up a game and allow you to test the correctness of basic features; for example, your implementation should include a method (or sequence of methods) to place a word, evaluate the legality points of the word and any special tiles, update the score, etc.[2]

Note, however, that we are not asking you to implement any user interface (graphical or otherwise) in the core implementation. That is, by "setting up a game," you are not adding any user interaction, you are merely calling the methods that might be called by a GUI and testing that those methods are correct.

We recommend that you finish implementing the core features of your game before you start to develop your GUI. Your core features must be implemented and tested independently of any user interface. You may commit (possibly incomplete) GUI code in the `edu.cmu.qatar.cs214.hw4.gui` package, but we will not consider this when grading Milestone B. Your solution may (but is not required to) use the dictionary we placed in the `src/main/resources/words.txt` file within the Eclipse project. Make sure that your solution compiles without files in the `edu.cmu.qatar.cs214.hw4.gui` package.

Testing the major components of your implementation

Test your implementation with JUnit tests. We do not have any coverage requirements, but you should be confident that the major features of your implementation work. As in homework 3, follow best practices for unit testing. We do recommend that you achieve around 80 percent method coverage and thoroughly test the scenarios of validating and placing a word modeled in Milestone A.

[2] e.g., Game g = ...; g.addPlayer(p); g.placeMove(word); assertEquals(10, g.getPoints(p));

Automate your build and tests with gradle so that your tests are automatically run on Travis CI.

Documenting your implementation

As usual, your implementation should be well documented using Javadoc comments and regular comments where appropriate.

Update your design documents if your implementation differs substantially from your initial Milestone A design. Do this only if your changes are conceptually significant. For example, you do not need to add private helper methods to your design documents, but you should update your design document if you change a component's design pattern or introduce new classes or important methods to your design.

Turning in your work

Turn in the following files for Milestone B:

- `object.pdf`: (optional) An updated object model.
- `src/main/java/edu/cmu/qatar/cs214/hw4/core` (and sub-packages): Your implementation of your game's core features.
- `src/test/java/edu/cmu/qatar/cs214/hw4/core`: JUnit tests for your core implementation.
- `build.gradle`: A gradle build script so that `gradle test` compiles and executes all tests.

Milestone C: Full implementation with GUI (due Oct 25th at 3:00 p.m.)

For this milestone you will complete the implementation of your game and assess your design process throughout the project.

Implementing a GUI

The full implementation of your Scrabble game must include a graphical user interface that allows a player to play your game. Your game must be playable, but we impose no requirements on the visual design of your interface. A fully-functional, playable game will earn more credit than a cool-looking-but-broken implementation.

We recommend that you write a Swing-based GUI. You may, however, use a different GUI framework (including web-based GUIs or an Android application) if you specifically ask for and obtain our permission in a private message on Piazza.

You may revise the design and core implementation of your game (from Milestone B) as necessary, but the core implementation must remain independent of your GUI implementation. All GUI-related code should be contained within the `edu.cmu.qatar.cs214.hw4.gui` package, and your core implementation should never import anything from this package. You do not need to test the GUI-related components of your implementation.

Extend your gradle script so that `gradle run` automatically starts your game. If instructions are needed on how to play your game, describe them in your `README.txt` or `README.md` file.

Design reflection

Update your design documents (`object.pdf` and `rationale.pdf`) to reflect the design, as implemented, of your core game. Discuss how you changed your design from your initial design in Milestone A to your final implementation in Milestones B and C. Explain why you made these changes. Discuss any changes to your design that were necessary to incorporate our additional special tile.

Turning in your work

Turn in all non-`.class` files related to this project in your `homework/4` directory and its subdirectories.

These files should include:

- `discussion.pdf`: A discussion of the changes you made in your design throughout the milestones.
- `object.pdf`: An updated object model.
- `rationale.pdf`: An updated description of your design.
- `README.txt/README.md`: A description of how we can play your game.
- `build.gradle`: A gradle build file supporting the targets `test` and `run`.

With your permission, we might feature your game in class if your game, GUI, or general implementation is exceedingly awesome.

Have fun!

Evaluation

Overall, this homework is worth 320 points; Milestones A and B are each worth 120 points, and Milestone C is worth 80 points. If you lose design-related points in earlier milestones, you can regain some of those points in the last milestone by improving your solution. For example, if your design has serious problems, we will provide feedback and outline what improvements are necessary to regain some of the lost points. The feedback you receive for Milestones A and B will contain additional details and instructions. Exceptional submissions may receive up to 10 points extra credit.

For full credit, we expect:

- Design documents that describe the core structures and behaviors of your game. Specifically, your interaction diagrams should clearly answer how a move is validated and executed and your object model should describe the core components of your game. The object model should follow the design goals discussed in lecture. If you trade one design goal for another, document this trade-off in your rationale and/or discussion documents. The components in your object model should correspond with the components in your interaction diagrams.
- Models that generally follow UML notation, as discussed in the 'A note on notation' section above. We will focus more on your overall design than on the specifics of UML notation.
- Plausible design discussions. It is possible to achieve full credit with a slightly flawed model as long as your design discussions convince us that you carefully considered your decisions. You should refer to the design goals, design principles, and design patterns introduced in lecture.
- JUnit tests that follow good practices, automated with gradle and Travis CI.
- A core implementation that is independent of any GUI implementation.
- A functional game implementation playable by multiple players.
- Readable code that follows standard naming conventions and good style.

When in doubt use your best judgment. Make reasonable assumptions and document them.