Homework #4: Scrabble with Stuff

In this assignment you will design and implement a variant of Scrabble^{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 February 23th)
- Milestone B: Core implementation and testing of game features (due March 9th)
- Milestone C: Full implementation with GUI (due March 23th)

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

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

Late days for the separate milestones are independent because the milestones have separate deadlines. If you use one late day for Milestone A and two late days for Milestone B, that costs you three late days, not two.

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. Many people play scrabble with house rules, but, for this assignment, we use the official rules by Hasbro as a baseline. Your implementation should roughly follow the official 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; it should include *challenges* of words by other players using the double challenge rules described in the official rules (see also Scrabble challenge on Wikipedia).

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) by March 13th, 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 Feb 26 at 11:59 p.m.)

For this milestone you must analyze and design your game. You should focus on the game-related functionality of the program, not its interface (think of playing the game by calling a sequence of methods; at the same time, it is very helpful to think about and possibly sketch out the GUI and how it interacts with the game at this early stage).

This milestone consists of the following tasks:

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

We describe each of these tasks below.

The domain model

Start by creating a domain model that provides the vocabulary for all subsequent tasks. To help identify the important concepts of your game, you may want to reread the Scrabble rules or the Scrabble Wiki page. Your domain model should be represented by a UML class diagram. It may be complemented by a glossary. For more information on domain models, see Chapter 9 of Larman's *Applying UML and Patterns.*

The system sequence diagram

Create a system sequence diagram for the interactions between the player and the system when the player plays the game. The system sequence diagram should help you determine the interactions between the high-level system and its users (here, a Scrabble player). It should also help you produce a realistic object model from your domain model. For more information on system sequence diagrams, see Chapter 10 of Larman's *Applying UML and Patterns.*

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 the 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). Constructing behavioral contracts should help you envisioning important changes of internal state of the game when a player interacts with the game. You may provide explicit examples to clarify your contract (e.g, what happens when two special tiles are activated in the same turn). For more information on contracts, see Chapter 11 of Larman's *Applying UML and Patterns*.

Two design scenarios

For each scenario below, visualize the interaction among objects in the game, creating at least one interaction diagram per scenario. You may split a scenario into multiple interaction diagrams if that improves the clarity of your visualization. Your interaction diagram should be a UML sequence diagram or UML communication diagram, optionally accompanied by prose. For more information on notations for interaction diagrams, see Chapter 15 of Larman's *Applying UML and Patterns*.

1. *Describe how a move is validated and applied to the game board when a player plays a set of tiles on the board.* This likely involves checking whether the player has the tiles to play and whether the tiles are placed in valid locations, as well as placing the tiles on the board, possibly triggering special tiles, and updating the score. This scenario likely starts by calling a method like `playMove` with parameter(s) representing a move on some abstraction of the game.

2. *Describe the actions in your game when a player challenges another player's word(s).* This scenario likely starts by calling a `challenge` method on some abstraction of the game. Challenging some previous player's word(s) might or might not trigger the removal of that player's tiles, adjust the score, or lose a turn for the challenger or challenged player.

These two scenarios are just 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 questions are worth considering and might help guide your design. You may include interaction diagrams and explanations for additional scenarios if you want.

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 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 and should reflect the interactions defined in your system sequence diagram. For more information on class diagrams, see Chapter 16 of Larman's *Applying UML and Patterns.*

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 formal scenarios above, and the informal questions that we provide later in this section.

Show your work to the course staff

To get full credit for this assignment you *must* show your work to, and get feedback from, a course staff member *before* the Milestone A deadline. We will schedule some design review sessions before the Milestone A deadline. Your best opportunity to get feedback is during these design review sessions; historically the TAs cannot substantially address everyone's design questions during office hours. We will announce the design session schedule on Piazza well before the Milestone A deadline. We expect that you finish a reasonably complete draft of all documents before the review session.

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. The problem is that, with software design, there are essentially no automated tools comparable to autograders or compilers or static checkers to provide immediate 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 this by getting early feedback from the course staff and (repeatedly) revising your design as necessary.

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.

Turning in your work

Turn in the following files for Milestone A in the `design_documents` directory:

- `domain.pdf`: The domain model for your game.
- `system_sequence.pdf`: The system sequence diagram for your game.
- `behavioral_contract.pdf`: The behavioral contract for a user placing tiles on the board.
- `interaction_move.pdf`: Your response to our scenario about validating and placing a move in your game.
- `interaction_challenge.pdf`: Your response to our scenario about challenging the previous player's move in your game.
- `object.pdf`: The object model for your game.
- `rationale.pdf`: Your justification of your design decisions.
- `README.txt/md/pdf`: Any additional information about your Scrabble implementation. This file should include, at minimum, a description of the additional special tile you plan to implement.

All documents may contain text and figures. Please label your final Git commit with a descriptive log message so we know which homework submission to grade, and whether to charge you with any late days.

Some design hints

When you are done, *Scrabble with Stuff* will be a medium-sized program with numerous interactions between game components, some of which are complex. Here are some 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 some interaction diagrams for these questions:

- How can a player interact with the game? What are the possible actions a player can perform?

- How would someone start a game with multiple players? How would players share the same screen?
- 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 special squares such as double letter score and double word score 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 general, extensible way, so special tiles can reuse code and additional tiles can be added later.
- Which component of your design is responsible for handling external events or user inputs? And which for the game logic? How should exceptions be handled?
- How does the game get represented on in a GUI and what information does the GUI need to access from the game?
- How does the turn of one player end and next player comes into play? This likely starts from some events denoting the end of a round, such as end of challenge, end of purchasing special tiles, player time out, or others. Ending a game round likely involves mutation of the game's internal state, such as active player, score, and tiles. You should think about how the game transfers controls from one play to another and what internal states need to be changed before implementing the game.

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. Chapters 9, 10, 15 and 16 of Larman's *Applying UML and Patterns* provides many details and guidance on UML notation. We do not require much formality, but we expect that relationships (such as association, inheritance, and aggregation) be described correctly in your diagrams, and that each relationship includes the cardinalities of the relationship. Attributes and methods should be specified correctly, but we do not require precise descriptions of visibility or types.

It is important that your models demonstrate an understanding of appropriate levels of abstraction. For example, your domain model should not refer to implementation artifacts and your object model should not include highly-specific details such as getter and setter methods unless they 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 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. We strongly recommend that you do not mechanically extract models from a software implementation; such mechanically-generated models are almost always at an inappropriate level of abstraction. We will accept handwritten models or photographs of models (such as whiteboard sketches) if the models are clearly legible.

Milestone B: Core implementation (due Mar 9 at 11:59 p.m.)

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

1. Improve your design based on our feedback, if applicable.
2. Implement the core features of your game.
3. Test the major components of your core implementation.
4. Document your implementation.

We describe each of these tasks below.

Revise your design based on our feedback

We will provide feedback (in a `grades/hw4a.md` file) on your work from Milestone A; our feedback might include recommendations for improving your design. We recommend that you revise your design based on our feedback, if possible.

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 of a play, and update the score.[2]

Note, however, that we are *not* asking you to implement any user interface (graphical or otherwise) in the core implementation. That is, by "set up a game", you are not adding any user interaction; you should only be calling the methods that might be invoked 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.cs.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

[2]e.g., A unit test could follow a system sequence diagram and play and test the game as follows: Game g = ...; g.addPlayer(p); g.placeMove(...); assertEquals(10, g.getPoints(p));

`src/main/resources/words.txt` file. Make sure that your solution compiles without files in the `edu.cmu.cs.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 at least 80 percent method coverage and thoroughly test the scenarios of placing and challenging a word modeled in Milestone A.

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.

If your implementation is substantially conceptually different from your Milestone A design it might help to update your design documents, but we won't evaluate your updated documents until after the Milestone C deadline.

Hints

1. If well thought through, the designs from Milestone A should significantly help you with your implementation, especially the system sequence and interaction diagrams.
2. Start early, do not wait for the Milestone B deadline to work on Milestone C.

Turning in your work

Turn in the following files for Milestone B:

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

Please label your final Git commit with a descriptive log message so we know which homework submission to grade, and whether to charge you with any late days.

Milestone C: Full implementation with GUI (due Mar 23 at 11:59 p.m.)

For this milestone you will add an additional special tile, complete the implementation of your game, and assess your design process throughout the project.

An additional special tile

After the Milestone B late deadline we will reveal another special tile that should be added to the game. This tile will be announced on Piazza by March 13th.

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 a web-based GUI 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.cs.cs214.hw4.gui` package, and your core implementation should never import anything from this package. You do not need to submit tests for 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, include them in your `README.txt` or `README.md` file.

Design reflection

Update your design documents and rationale 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 and discuss any design changes needed to incorporate our additional special tile. You should explicitly describe improvements you made to your design documents and your design; for Milestone C we will re-evaluate your design changes (if you describe them and we can find them) and partially restore credit lost in Milestone A.

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 and a discussion of the effort required for the additional special tile.
- `rationale.pdf`: An updated description of your design.
- `*.pdf`: Updated design documents from Milestone A.
- `README.txt/README.md`: A description of how we can play your game and a summary of your design changes we should re-evaluate.
- `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 Milestone A you can regain some of those points in Milestone C 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:

- A domain model that describes the vocabulary of the problem.
- 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 played, how words are challenged. Your object model should describe the core components of your game, following 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 and insightful 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.