Homework #3: Single Player Terminal Yahtzee

CPSC 224 - Software Development

Note: There is no new GitHub Classroom link for this assignment. Use the same repo you used for HW2, just continue your work from your last HW2 commit.

THE PROBLEM:

Extend your solution to Homework #2 to play a complete one-player game of Yahtzee. Your solution should be functionally equivalent to the Bruce's Python code available on GitHub: https://github.com/worobec/Yahtzee. You can download and run that python program on ada.gonzaga.edu if you don't have a Linux computer (or Python3 on Windows) to see it with.

Specific requirements include:

- Ability to enter a code to select the Yahtzee scorecard line to use at the end of each turn
- Ability to enter "S" to see the current scorecard at the beginning of each turn
- The upper scorecard should calculate a bonus per the rules of Yahtzee
- Once a score has been placed on a scorecard line, that line should not be included when displaying *possible* scores in subsequent turns
- If the player selects a 0 score for a line (like having 1,1,3,4,5 for Yahtzee is a 0 score), that line should not be included when displaying possible scores in subsequent turns (so you cannot simply look for a score of 0 to determine is a line is still available)
- As per HW #2, the upper scorecard should adjust based on the number of sides on the die as selected by the user
- The lower scorecard does not need to change as the number of dice and sides change, although you are free to do so (make sure any enhancements you make are documented in your design summary3.pdf document)
- In HW #4 your code will need to validate input and check for other exceptions—any investment in error-checking you include in this assignment will save you time later

Unit testing continues to expand:

- For this assignment all of your classes need at least one test on them, and I would suggest at least two for each method for good practice.
    - If it's a trivial class (like a driver Yahtzee class with only a static main), then just an instantiation class will suffice. If it's a non-trivial class, then the test needs to be non-trivial too.

Project operations and development processes:

- To kick off this assignment, make sure to tag the commit with your work from HW2, and then branch from there to start HW3
    - On command line:
        - git tag -a "HW2" -m "my message about it being ready for release"
        - git push --tags
    - In intellij:
        - Menu: VCS->Git->tag
        - Then: VCS->Git->push (checkbox for "push tags")
- For this assignment, I shall be looking at your git repo use. You should be starting to follow the more 'proper' git workflow behavior, as we saw in class.
    - The workflow is illustrated here: https://guides.github.com/introduction/flow/
    - You can do this assignment in a single branch or several smaller branches, but there shall be at least once branch off main/master
    - Once your work is done in the branch, you should use a pull request to merge it back into the main branch
    - If you still need practice on using this workflow, please create a repo on GitHub to practice on. They're free and you can try out the various commands, tools, web interfaces, and everything else.

    - Note: IntelliJ or VS code (with the 'git graph' extension) will render the git repo's network of commits and branches under the git log area:
        - See image attached at end of this document for an example
- After your code is ready, merged to main, and you're done with the project, tag the commit with the tag "HW3"
    - On command line:
        - git tag -a "HW3" -m "my message about it being ready for release"
        - git push --tags
    - In intellij:
        - Menu: VCS->Git->tag
        - Then: VCS->Git->push (checkbox for "push tags")

SUBMISSION:

When formatting and turning in your assignment be sure to follow Dr. Sprint's Coding Standards which include the use of javadoc throughout your code.

Your code should be submitted via GitHub Classroom. Your assignment repository should also contain a summary .pdf document (in the Summaries/ directory) describing the major design and development considerations encountered in implementing your assignment. This document should be written in a narrative style and should include:

- A summary of the goal or purpose of the program in your own words.
- An overview of the general design you chose for your program.
- A UML Class Diagram that includes all classes that are part of your solution (but not the test classes, those are not part of the solution). Diagrams must be produced by a software tool and not hand drawn - dia, draw.io, or Visio
- UML Sequence Diagrams - Show the actors/objects in your code and how they're interacting over time
    - This doesn't need to be *every* object, but core gameplay behaviors should be covered
- A description of any major design and/or programming issues, why these were the major issues, how you addressed them, and why you addressed them the way you did.
- A retrospective of what you would have done differently if you had more time.

GRADING:

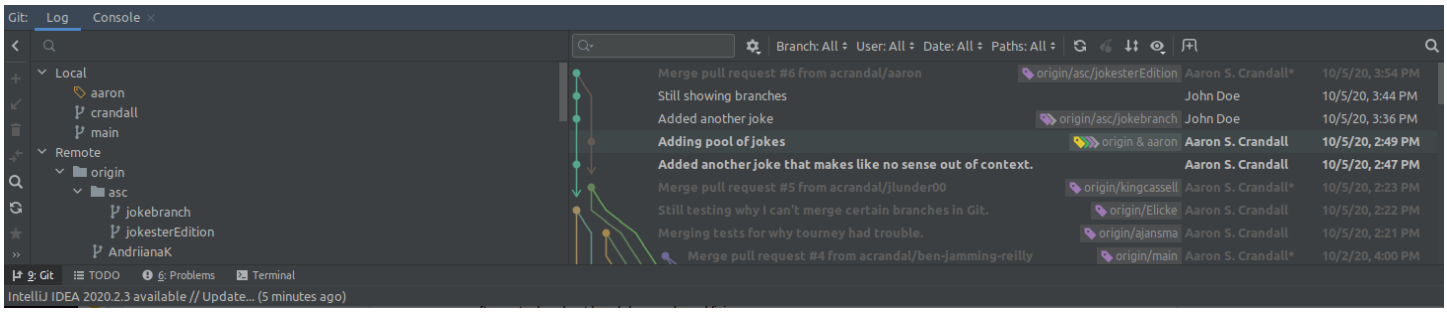
- Summary document - 10pts
- Style - Dr. Sprint style - 10pts
- Unit tests cover all classes at least in one place - 10pts
- Code compiles - 30pts
- Game plays to end - 30pts
- Multiple object solution with reasonable design - 10pts
- Git branch in repo - 10pts

IMPORTANT:

This is still an individual assignment. Everything you turn in should be the result of keystrokes done by you. You should not consult nor use any existing classes related to Yahtzee or dice. You should not share your code nor look at the code of your classmates. It is ok to have generalized discussions about java and to have high-level design discussions about the classes, attributes, and methods necessary for the solution.

Appendix A - Git commit network rendering

IntelliJ showing network graph on the right. Newest commits are at the top.



VSCode's Git Graph extension which shows up under the version control section.