Software I Project

Neil Wilcoxson

April 26, 2018

Link to repository:
`https://github.com/neilwilcoxson/yahtzy`

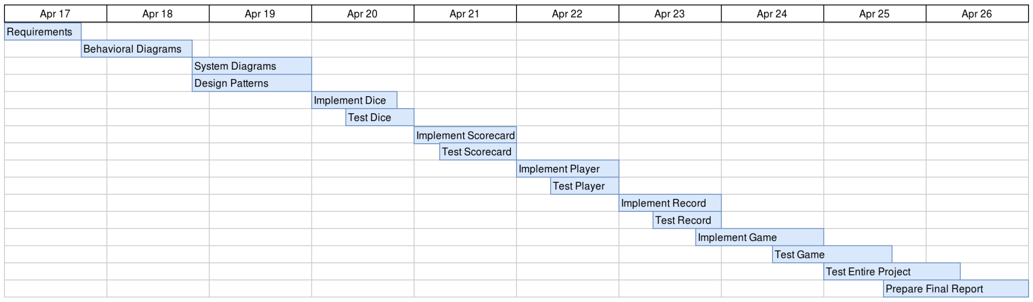
Abstract

Insert Abstract Here

Background

Yahtzy is an extended digital of Hasbro's popular dice game, Yahtzee. While playing with physical dice and paper scorecards can be fun, this digital version seeks to make gameplay faster and easier. Furthermore, the autocalculation and hint features help new players learn how to play the game.

Schedule

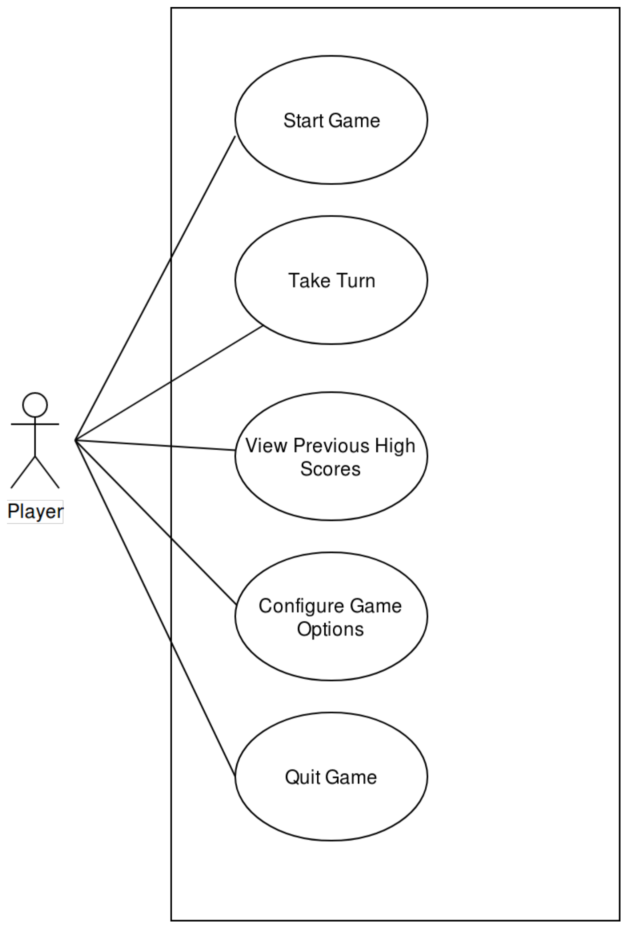


Requirements

1. Player can start a new game
2. Multiple players can play together
3. Player can select game options (normal or tripple mode)
4. Player can roll dice
5. Player can select which dice to hold
6. Player can select where to score their rolls
7. Player can view previous high scores
8. Player can view their total score
9. Player can exit the game during gameplay

Analysis

Use Cases



UC1: Starting a game
Scope: Yahtzy Application
Level: user goal
Primary Actor: Player
Stakeholders and Interests:
-Player(s): Want to have fun playing a game
Precondition(s): Application is ready to run on the machine
Postcondition(s): Game is ready for the player(s) to play

Main Success Scenario:

1. Player launches the application
2. Application asks how many players and their names
3. Application is ready for the first player to take their first turn

UC2: Taking a turn
Scope: Yahtzy Application
Level: user goal
Primary Actor: Player
Stakeholders and Interests:
-Player(s): Want to have fun playing a game
Precondition(s): Application is running and game has been started
Postcondition(s): Players score is recorded and Application is ready for next player

Main Success Scenario:

1. Game displays name of player who needs to take their turn
2. Player clicks roll dice
3. Player selects which dice to keep
4. Start again from step 2 up to 2 more times
5. Player selects where to score their roll
6. Control is passed to the next player

Alternative paths:
3.* Player wants to keep all dice they rolled and score immediately
3.1 Jump to step 5

UC3: View previous high scores
Scope: Yahtzy Application
Level: user goal
Primary Actor: Player
Stakeholders and Interests:
-Player(s): Want to have fun playing a game
Precondition(s): Application is running

Postcondition(s): List of high scores is displayed

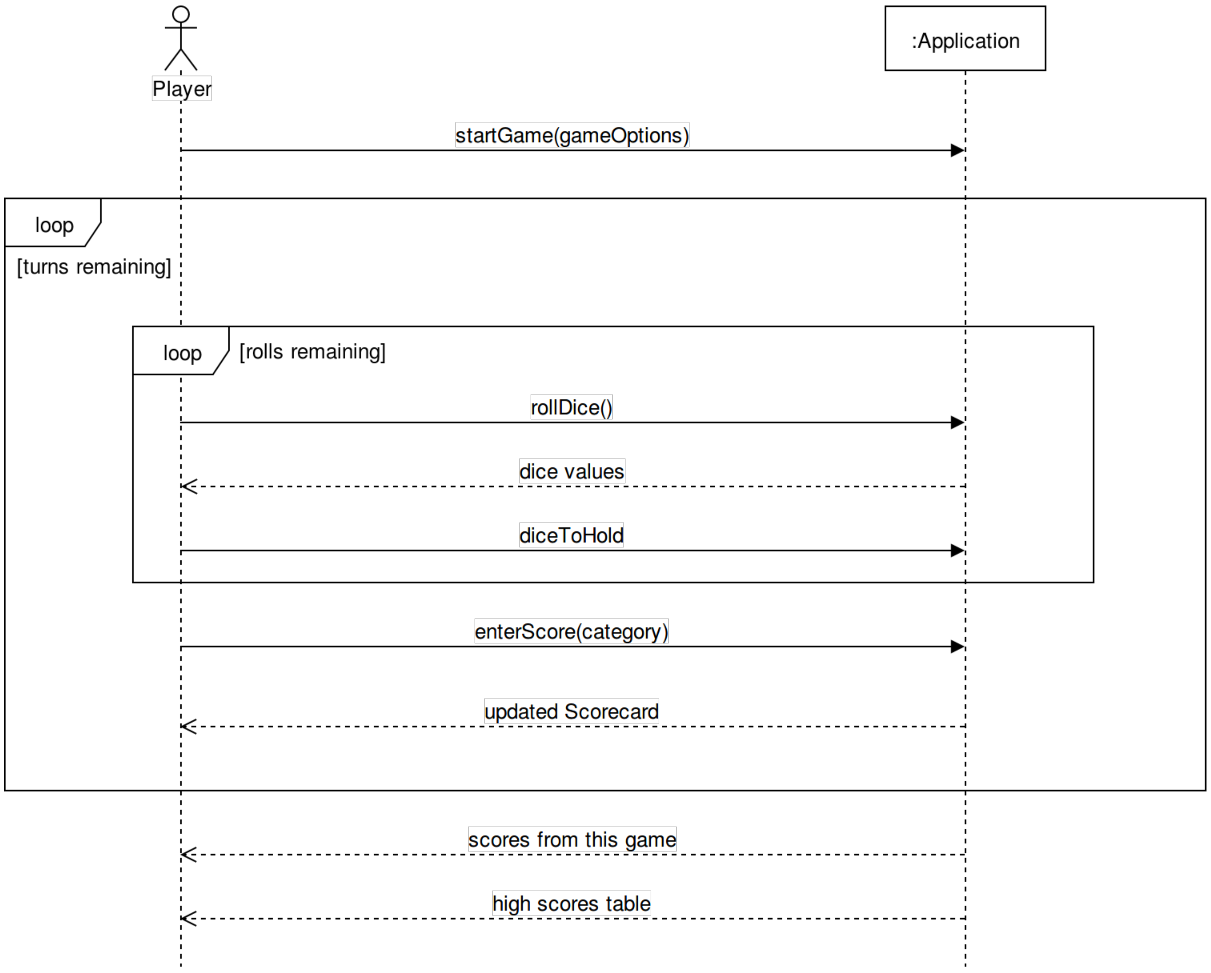
Main Success Scenario:

1. Player clicks the high scores button
2. Application displays a list of previous high scores (if any)

Business Rules

1. Scores should be calculated by the application rather than by the player.
2. When a score category is selected for which no combination of the dice apply, the recorded score should be zero.
3. Players are not eligible for the bonus if they have already recorded a zero in the Yahtzy space.

System Sequence



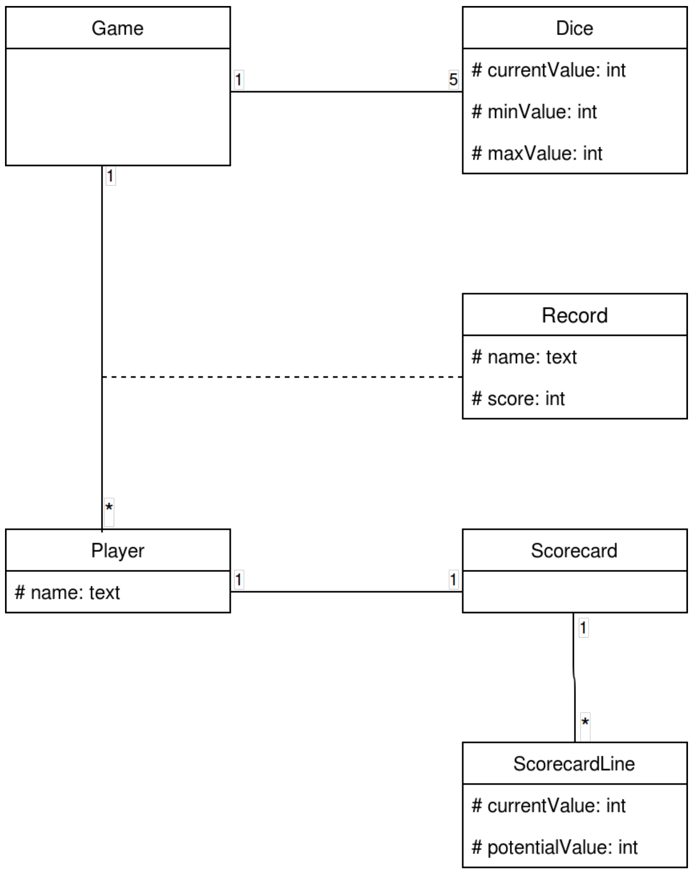
Operation Contracts

As stated in Larman's book, not all artifacts will always be needed. In the case of a game for which nearly everyone is familiar with the rules, these are not necessary in this case.

Design

Concepts

The main conepts were mostly based on the physical objects involved in playing the game (scorecard, dice, player), along with well understood objects like the game and record.



GRASP

Information Expert

Creator

Controller

Low Coupling

High Cohesion

Polymorphism

Pure Fabrication

Indirection

Protected Variations

Design Patterns

Although many more design patterns were used in the making of this application, here are five major ones.

1. **Polymorphism**
   All of the scorecard categories inherit from a common ScorecardLine class. This allows them each to implement their own method for scoring the dice, which is taken advantage of by dynamic binding.
2. **Facade**
   The Game class is a Facade as it arranges all of the game logic for the GUI. Thus the GUI does not have to interact with each element of game logic directly.
3. **Bridge**
   Certain methods of each class are implemented with compatibility in mind. For example, the Game class implements a getDiceValues method, because the GUI does not need to know about the enitre Dice object, but rather just the values.
4. **Chain of Responsibility**
   Almost every class implements Chain of Responsibility. When the game wants to score the current dice roll in a particular category, it does not have to contact the ScorecardLine directly. Instead the game contacts the scorecard. The scorecard knows what to do, but the game doesn't need to know how it is implemented.
5. **Observer**
   In order to respond to the GUI, the Observer pattern is used. The buttons present in the GUI are observered and the appropraite repsonse is initiated when the buttons are clicked.

Implementation

This application was implemented using Java SE using Swing as the front end. The source code is available at the repository for the project: `https://github.com/neilwilcoxson/yahtzy`

The logical line count for the entire application at the time of this report was xxxx lines.

Evaluation and Testing

The logical components of the game were tested using JUnit 5 test cases. However, the graphical components of the game often had no means for testing via JUnit. These elements were tested manually. SpotBugs was also used to help identify bugs. Furthermore prerelease versions of the game were sent to a small group of beta testers who were asked to make suggestions for improvements to the game. This suggestions were taken into account before the final release.

Conclusion

Software Engineering is always subject to the opinions and the people doing it.