Course: Olio-ohjelmoinnin perusteet TKO_2005, year 2015

Topic: Yahtzee in Java Swing

Instructor: Moodle is unclear on that

Author: Timo Valeri Junolainen (503537), tivaju@utu.fi or sigin@norsula.com

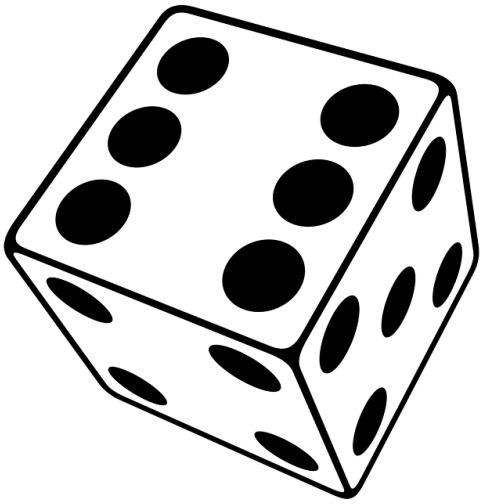

Table of Contents

Task and task analysis....3
Development....4
MVC....6
Instructions....10
Rules....11
Development and testing....12

Task and task analysis

From range of offered topics (harjoitustyon_kuvaus.pdf), I decided to select **C Peli,** and for game I decided to go with Yahtzee.

Must admit, choice of topic is dictated due to simplicity. Outlined task can be achieved mostly with `java.*` and `javax.*` classes, with exception of Saving and Loading game, which for convenience is implemented by using INI file format, and external library `ini4j`.

For GUI I decided to go with Swing library, while JavaFX would work too, and increase portability to mobile devices. JavaFX has worse, less convenient tables, and is not requirement, so Swing is the choice.

Development

Development is implemented with Object Oriented principles, as course name and common sence dictate.

Lazy trick were used upon finishing game – instead of garbage collection and giving end user possibility to play another game – game quits.

Also it might be good choice to reset `save.ini` upon loading, but thats more of taste choice and I decided to leave it like it is. Besides – it helped to further debug work.

Few words about debugging – I left **”Debug”** button on play field active, which brings game to very final stage.

Main objects are forms(JForm) or modal forms (Jdialog), and since yahtzee game is very distinctly structured, game logic is governed by active/passive elements of Form.

In principle yahtzee game can include more than two players, but my software implements solely two players. Each game consists of 13 turns, and each player makes exactly same action during own turn.

Players turns arent dependable on one another, so I decided to leave out ”who is first”, it makes absolutely no difference from game point of view.

Game flow can be shortly described this way:

Turn 1

Player 1, roll1
Player 1, hold choice, roll2
Player 1, hold choice,roll3
Player 1, choice for scoretable

Player 2, roll1
Player 2, hold choice,roll2
Player 2, hold choice,roll3
Player 2, choice for scoretable

Turn 2

Player 1, roll1
Player 1, hold choice,roll2
Player 1, hold choice,roll3
Player 1, choice for scoretable

Player 2, roll1
Player 2, hold choice,roll2
Player 2, hold choice,roll3

Player 2, choice for scoretable

…..

…..

Turn 13

Player 1, roll1
Player 1, hold choice,roll2
Player 1, hold choice,roll3
Player 1, choice for scoretable

Player 2, roll1
Player 2, hold choice,roll2
Player 2, hold choice,roll3
Player 2, choice for scoretable

Points counting and determination of winner

After turn 13 it is determined, that game should end. Amount of turns cant be more or less than 13, it is exactly 13, and there is exactly 4 action for each player in each turn.

Game can be saved at any moment possible, analogue of ”Panic button”. Should mention here, that game do not save which dices are on hold. Just at which point game was saved, and what combination of dices is on the board. There is no techincal limitation to implement this, but since it is kinda luxury, I decided to leave it like it is.

MVC

Controller:

Implemented in Java Swing. Has few forms, modal or otherwise:

`LoginFrame` – where everything starts. User is given choice to start new game, load saved game or check list of previous game results. New game calls for `AskNames`, load game calls for `BattleField` and previous results call for modal `Hall`.

`AskNames` – Very simple form-object, which asks for Player1 and Player2 names, initializes empty player records and initializes and calls for `BattleField`. ATTN: Loading game in LoginFrame loads all Player1,2 and `BattleField` states, and calls `BattleField`. Since upon loading we dont need to ask player names, `AskNames` is skipped.

`BattleField` – This is where magic happens. Formobject holds two similar records for both players, as well as general, shared information: dices and `turnState`. Contains important methods from game point of view, like rolling and holding dice, saving game, as well contains important object `PlayerRecord,` which is instantiated for both players.

`ClaimVictory` – called upon finishing game, updates hall.txt and shows winner with final scores. Modal, and calls `System.exit(0)`, which is lazy but not forbidden.

`Hall` – another very simple formobject, called from `LoginFrame`, modal, and simply shows contains of `hall.txt` (results of previous games)

Model:

Almost all logic happens in BattleField, which extend Jform.

Contains turn state – turnState integer, important object PlayerRecord, import of ini4j.

turnState is approach to say at exactly which part of the game we are. Ie
11 – Player1 roll
12 – Player1 roll
13 – Player1 roll (Roll it button active, choice inactive)
14 – Player1 choice (Roll it is inactive, but choice of combination is active)

21 – 24, same but for Player2

I dont count how much moves were made, instead at each state=24, `PlayerRecord.hasFinished` is called, to determine if player2 is finished filled scoreboard. Notice – there is no need to check Player1 for final condition, since game is deterministic. If player2 filled scoreboard, this is condition at which game ends.

`PlayerRecord` should be mentioned separately, since it holds many important from gameplay point of view methods and attributes.

`ones, twoes, threes, fours, fives, sixes, threekind, fourkind, fullhouse, large, small, yahtzee, chance` integers hold amount of points scored for combination in question. If integer is -1, means player hasnt been used this combination yet.

Method `score()` returns amount of points in `PlayerRecord`.

```
oneScore()
twoScore()
…
chanceScore()
```

methods return amount of points, counted from `DiceA,DiceB,DiceC,DiceD,DiceE` on board with regard to asked combination. *Score methods are the heart of gamelogic.

More about classes, methods and attributes:

`Yahtzee`

metacontainer for starting application, nothing special about it.

`LoginFrame`

Contains two important methods, `newGameBtnActionPerformed` – this one is simple, does call for `AskNames and loadGameBtnActionPerformed` – this one is more sophisticated, does similar actions to what happens in `AskNames`, but instead of initializing records and objects with default values, loads values from ini file and initializes `BattleField` with those.

`AskNames`

Another extremly simple class, does nothing besides `namesDoneBtnActionPerformed` method, which is called after players enter their names, initializes `BattleField` object, and calls methods `InitPlayerOne()` and `InitPlayerTwo()`, which basically puts -1 in all fields of player records.

Also initializes turnState to 11, which means that in the beginning of game it is first player move

and no rolls been done yet.

`BattleField`

Already mentioned before, this class extends `Jframe`, and is heart and brains of whole game.

Has important property, `turnState` – indicates at which point of game we are exactly now. 11 player 1 first roll, 12 second roll, 13 third roll, 14 choice of combination, 21,22,23,24,11,12,13.. cycles indefinitely, until endgame requirements are met.

`DiceThrow()` - extremly simple method, utilizes java.util.Random for randomized dice throwing.

`PlayerRecord` – two of those will be instantiated, each represents information relevant to each player. Contains player `name`, and record corresponding how much points player have for combination. -1 means player didnt yet chose combination – initial state.

`PlayerRecord.bonus()` – method which returns 50 is player has 63 or more points in upper section, otherwise returns 0.

`PlayerRecord.hasFinished()` – returns true if there is no empty categories in player record (no -1es, all moves has been made), otherwise returns false. Called only for second player, after `turnState` is 24.

`PlayerRecord.score()` – returns sum of points for player records, along with possible bonus points.

`PlayerRecord.onesScore(), twoesScores()...*Score()` – checks dice situation, and counts corresponding combination score.

`UpdateChoice()` – updates list with possible combinations to pick, called from `UpdateScoreboard()`

`MakeRoll()` – Utilizes `DiceThrow()` for all dices, excludes holded dices from changing value.

`InitPlayerOne(name),InitPlayerTwo` – initializes `PlayerRecord` objects, one for each player. Sets name and scores to -1.

`UpdateScoreboard()` – another importand from game flow point of view method. Updates visual elements, ie – disables score choice if we didnt make all rolls(11,12,13,21,22,23), and disables Roll it! Button, if we are making choice(14,24)

Updates scores on screen for both players, puts 'x' if score is -1.

View:

Java+Swing, nothing more to be said here, except that dices are part of the model, labels work not only as visual element, but also as a variable to keep dice value, therefore made public, not private (So those can be accessed outside of the class)

`ClaimVictory` form and `Hall` form probably good to be mentioned here as well, those have no impact on game logic whatsoever, and purely decorative. `ClaimVictory` is called when endgame requirements are met, and `Hall` is called when we want to check history of previous games.

Instructions

Upon starting game user have three choices, to start new game, to load previously saved game or to check hall of previous games. Hall is pretty self explanatory, and has nothing to it – just shows new for with previously finished games and scores.
New Game – asks for both players names and initiases battlefield.
Load Game – initialises battlefield with previously saved values, names and continues from the point which was saved.

For simplicity lets imagine we started new game and entered both players names.

Debug button on battlefield fills both player records, except second player has chance open and three rolls, this is for debug reasons, left it enabled, in final build should be disabled, invisible and probably removed.

Save&Quit – saves game position and scores to ini file, and quits the game.

Upon starting game we see whos turn currently on. If Roll it button is active, choice is inactive – this means we still have possibility to hold some dices, to unhold some dices and to reroll unholded dices.
After three rolls, Roll it becomes inactive, but choice becomes active. Player chooses which combination he/she would like to full fill, and turn transfers to another player.
After second player makes last rolls and choice winner is determined, and game quites. Debug button transfers us exactly here.

Rules

Scoring instructions vary from yahtzee to yahtzee, for simplicity I used scoring rules from Wikipedia:

Uppersection:

Ones – 1*dices(ones)
…
Sixes – 6*dices(sixes)

Lowersection:

Threekind – All dices
Fourkind – All dices
Fullhouse – 25 points
Small – 30 points
Large – 40 points
Yahtzee – 50 points
Chance – All dices

Notice – Bonus points(50) awarded if player has 63 or more points in uppersection.

Development and testing

All done by Timo Junolainen, noone else was involved. Work started at 15.3 and ended 18.3

References:

http://www.utu.fi/fi/yksikot/sci/yksikot/it/Opiskelu/Sivut/Dokumentointiohje.aspx
https://moodle2.utu.fi/pluginfile.php/349395/mod_resource/content/0/harjoitustyon_kuvaus.pdf
http://en.wikipedia.org/wiki/Yahtzee