Jared Weyer

CPSC 224, Spring 2020

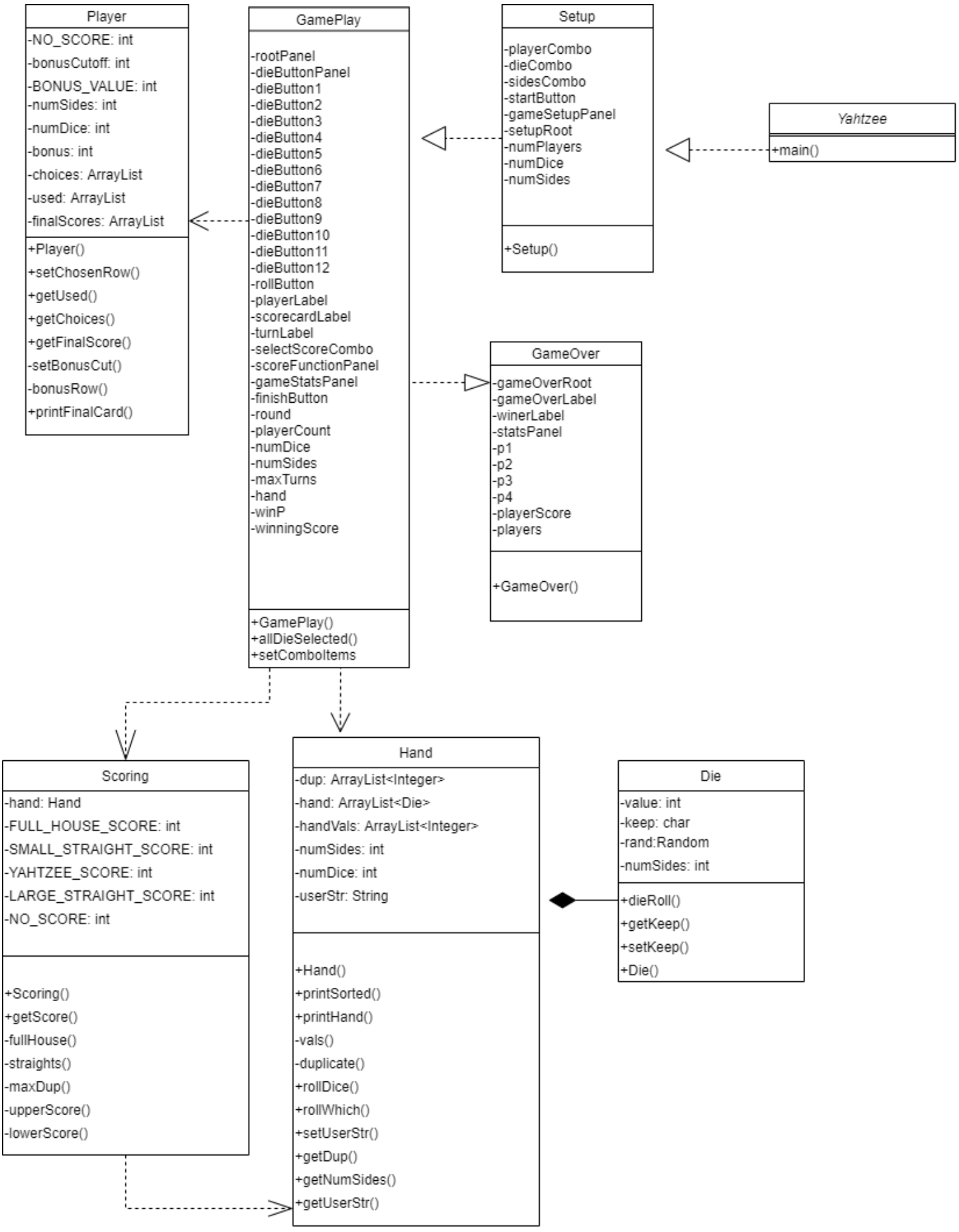
Programming assignment #4

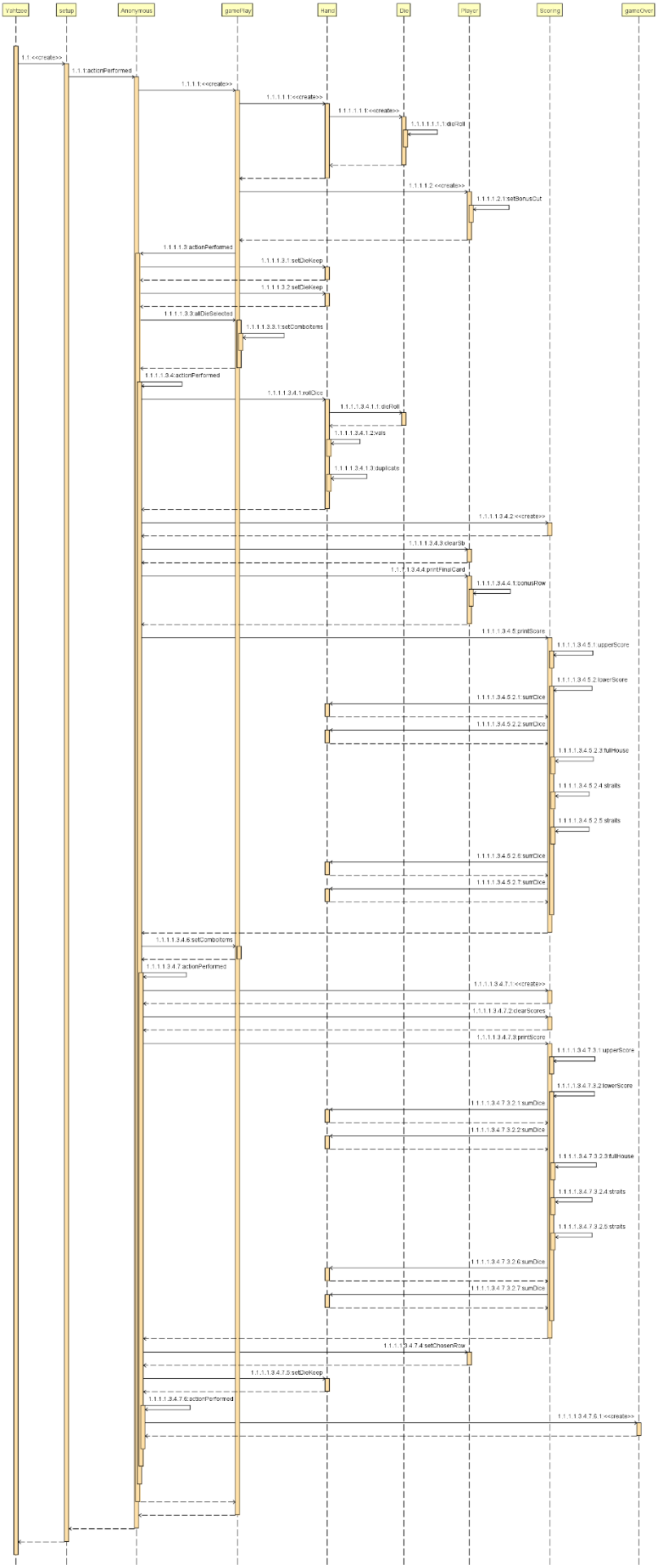
March 25, 2020

The goal of the Yahtzee program is to create a program in which users can play a virtual version of the game Yahtzee. This game should be able to be modified by the user in that the user should be able to input the following: how many players are playing in the game, how many die are in the hand, how many sides are on a die, and how many rolls a player gets. The game should allow each player to roll the specific dice in his/her hand that he wants to roll and keep the dice that he/she wants to keep. The game should output the scores for each line on the Yahtzee scorecard for a given players hand, and the rows that the player has already used should not be shown. The players should also be given the ability to see their scorecard at the beginning of their turn. The game should go on until all the rows of the scorecard are filled.

In the design of the Yahtzee program I implemented 8 classes: the main Yahtzee Class, the Hand Class, the Die Class, the Scoring Class, the Player class, the gamePlay class, the Setup class, and the gameOver class (See the UML diagram below for a description of these classes). The program design was implemented so that a user will have one hand object which contains a certain number of die objects (this will be determined by user input). The gamePlay class will allow the user to interact with a GUI which allows him to interact with the Hand object. These interactions will be limited to the user being able to control which of the die in his hand that he/she wants to roll. After the users turns are exhausted the Scoring class will be used to determine the different scores in which the final hand has earned. These scores will be determined by giving the Scoring class access to the hand object and the Hand class; further, the scores will be based off the rules listed in the Yahtzee board game. This functionality will be given to each player who, at the end of each turn, will be able to input which row they want to use on their scorecard. The player will do this by entering a string that corresponds to a certain row on the scorecard. The value of the chosen row will be taken from the Scoring class and put into the Player objects final scorecard scores. This will continue until all rows have been filled

The main challenge I faced when creating the Yahtzee program was figuring out the right number of classes to implement. In my first iteration of the program design I incorporated too many classes which were all interdependent on one another. This dependency created an overcomplicated code in which I decided to completely delete and start from scratch. In the end I found that using four classes was the right balance between overcomplication and classes that had too much functionality (god classes). In retrospect doing more design work on the program before I started to code would have saved me a lot of time. However, I leaned quite a bit about OOP programming by just jumping into the coding and learning from my mistakes. Below is the UML diagram associated with the Yahtzee program:


Player
-NO_SCORE: int
-bonusCutoff: int
-BONUS_VALUE: int
-numSides: int
-numDice: int
-bonus: int
-choices: ArrayList
-used: ArrayList
-finalScores: ArrayList
+Player()
+setChosenRow()
+getUsed()
+getChoices()
+getFinalScore()
-setBonusCut()
-bonusRow()
+printFinalCard()
GamePlay
-rootPanel
-dieButtonPanel
-dieButton1
-dieButton2
-dieButton3
-dieButton4
-dieButton5
-dieButton6
-dieButton7
-dieButton8
-dieButton9
-dieButton10
-dieButton11
-dieButton12
-rollButton
-playerLabel
-scorecardLabel
-turnLabel
-selectScoreCombo
-scoreFunctionPanel
-gameStatsPanel
-finishButton
-round
-playerCount
-numDice
-numSides
-maxTurns
-hand
-winP
-winningScore
+GamePlay()
+allDieSelected()
+setComboItems
Setup
-playerCombo
-dieCombo
-sidesCombo
-startButton
-gameSetupPanel
-setupRoot
-numPlayers
-numDice
-numSides
+Setup()
Yahtzee
+main()
GameOver
-gameOverRoot
-gameOverLabel
-winerLabel
-statsPanel
-p1
-p2
-p3
-p4
-playerScore
-players
+GameOver()
Scoring
-hand: Hand
-FULL_HOUSE_SCORE: int
-SMALL_STRAIGHT_SCORE: int
-YAHTZEE_SCORE: int
-LARGE_STRAIGHT_SCORE: int
-NO_SCORE: int
+Scoring()
+getScore()
-fullHouse()
-straights()
-maxDup()
-upperScore()
-lowerScore()
Hand
-dup: ArrayList<Integer>
-hand: ArrayList<Die>
-handVals: ArrayList<Integer>
-numSides: int
-numDice: int
-userStr: String
+Hand()
+printSorted()
+printHand()
-vals()
-duplicate()
+rollDice()
+rollWhich()
+setUserStr()
+getDup()
+getNumSides()
+getUserStr()
Die
-value: int
-keep: char
-rand:Random
-numSides: int
+dieRoll()
+getKeep()
+setKeep()
+Die()


Yahtzee
setup
Anonymous
gamePlay
Hand
Die
Player
Scoring
gameOver
1.1:<<create>>
1.1.1:actionPerformed
1.1.1.1:<<create>>
1.1.1.1.1:<<create>>
1.1.1.1.1.1:<<create>>
1.1.1.1.1.1.1:dieRoll
1.1.1.1.2:<<create>>
1.1.1.1.2.1:setBonusOut
1.1.1.1.3:actionPerformed
1.1.1.1.3.1:setDieKeep
1.1.1.1.3.2:setDieKeep
1.1.1.1.3.3:allDieSelected
1.1.1.1.3.3.1:setComboItems
1.1.1.1.3.4:actionPerformed
1.1.1.1.3.4.1:rollDice
1.1.1.1.3.4.1.1:dieRoll
1.1.1.1.3.4.1.2:vals
1.1.1.1.3.4.1.3:duplicate
1.1.1.1.3.4.2:<<create>>
1.1.1.1.3.4.3:clearSb
1.1.1.1.3.4.4:printFinalCard
1.1.1.1.3.4.4.1:bonusRow
1.1.1.1.3.4.5:printScore
1.1.1.1.3.4.5.1:upperScore
1.1.1.1.3.4.5.2:lowerScore
1.1.1.1.3.4.5.2.1:sumDice
1.1.1.1.3.4.5.2.2:sumDice
1.1.1.1.3.4.5.2.3:fullHouse
1.1.1.1.3.4.5.2.4:straits
1.1.1.1.3.4.5.2.5:straits
1.1.1.1.3.4.5.2.6:sumDice
1.1.1.1.3.4.5.2.7:sumDice
1.1.1.1.3.4.6:setComboItems
1.1.1.1.3.4.7:actionPerformed
1.1.1.1.3.4.7.1:<<create>>
1.1.1.1.3.4.7.2:clearScores
1.1.1.1.3.4.7.3:printScore
1.1.1.1.3.4.7.3.1:upperScore
1.1.1.1.3.4.7.3.2:lowerScore
1.1.1.1.3.4.7.3.2.1:sumDice
1.1.1.1.3.4.7.3.2.2:sumDice
1.1.1.1.3.4.7.3.2.3:fullHouse
1.1.1.1.3.4.7.3.2.4:straits
1.1.1.1.3.4.7.3.2.5:straits
1.1.1.1.3.4.7.3.2.6:sumDice
1.1.1.1.3.4.7.3.2.7:sumDice
1.1.1.1.3.4.7.4:setChosenRow
1.1.1.1.3.4.7.5:setDieKeep
1.1.1.1.3.4.7.6:actionPerformed
1.1.1.1.3.4.7.6.1:<<create>>