Kim Weible

**Changes to Pig Dice Game**

1. Add New Game Feature – 4 points
   * I decided that the best way to implement this feature was to make it a menu item under the “Game” menu but place it just above the “Exit.” This made the most sense to me as to how it should display as an option to the user, and it allowed me to give it a mnemonic Ctrl+N and accelerator. Then, I started building out the action event listener and handler for the menu option in a method called buildNewGameMenu.
   * As the name of the feature implies, I started to create a new game object, but I ran into some issues with tracking the first player, the selected strategy and clearing out the different panes. So, I decided the first thing to do was break up the current code into helper methods so I could keep track of all the parts.
   * I knew I wanted to implement the random player start, so I made a private helper (randomPlayerSelection) inside of the NewGamePane inner class of PigPane, but I didn’t want any code redundancy, so I also split out the computer player start logic (computerPlayerSelection) and human player logic (humanPlayerSelection) into private helper methods in NewGamePane that would be accessible to the random player selection, the event listeners for those radio choices and the outer PigPane methods.
   * I needed to make sure I had the first player setter and getter to store the first player selection, so I decided to make those in the Game class and set a private instance variable (firstPlayer) that holds a string. I decided on a string, because in the event that user starts the game, does not make a player selection and then tries to start a new game, this variable would be empty, and I needed to use the “empty” state to control some of the logic.
   * In the same manner as the first player, I needed to make sure I kept track of the strategy set in the menu, so I decided to take the same tactic as the first player and create a getter and setter in the Game class as well as an instance variable (currentStrategy).
   * Once these getter and setters were available, I updated the event handle code for each of the strategies and the two players to set these values when they are triggered on click.
   * Once I had these new methods, new instance variables and the code broken up, I created 3 new helper methods: resetComputerPlayerFirst, resetHumanPlayerFirst and resetPigPane that reset the individual panes using code from the private methods that build the panes objects, so as to minimize any code redundancy. This allowed me to rebuild the panes and trigger the right states if the computer started first, the human started first, or the user did not make any player selection (ie I have to force them to make a choice).
   * With these items in place, I was able to build out the event listener logic for New Game to first set the default strategy to cautious, get the firstPlayer and currentStrategy values, create a new game object off the current strategy selected with new panes and players, and then use the firstPlayer value to choose which way to reset the panes, using an empty choice to force the player to make a player selection.
2. Add Random Player Selection – 2 points
   * I decided to do this as a checkbox instead of as a radio button that could interfere with the toggle group, because it’s a standalone selection. Since I broke up the logic for the random player into a private helper method inside the NewGamePane, I was able to add the checkbox to the buildPane method, create a private inner class called RandomFirstListener to handle the action event and call the private method randomPlayerSelection to use Math.random() to choose to call either the ComputerFirstListener or the HumanFirstListener inner classes.
   * This allows the user to check the box and the game automatically chooses the random player and starts the game with the appropriate player pane enabled and disabled
3. Add Turn Dice Rolls Log to each Player – 4 points
   * I decided to use a list view object to store these rolls and place them under the current player Turn Total. A list view made sense to me, because it has a scroll feature, and allows for the list items to have a different shading for even and odd rows, so the list is easier to read. I liked the way the headers for each player pane are styled, so I added a label above the list view that looks like the header, but I gave them a simple label (~~ Rolls ~~). I thought this blended well into the current UI as a nice log.
   * I added these objects to the HumanPane and ComputerPane with the list view as a private instance variable (rollsListView) and a simple label. Each were added as a VBox named rollBox at the bottom of the buildPane().
   * In order to make the list of dice rolls, I knew the best way to proceed was to create an array list getter and setter in the AbstractPlayer class that would store the string dice pairs (die1, die2). I added a private instance variable (turnRollsList) and created a setter method (setTurnRollsList) that took a string parameter of the dice pair rolled, a getter method (getTurnRollsList) and a method to re-initialize the array list (resetTurnRollsList).
   * I added a call to the setter inside the processTurn method just after each die was rolled, so the roll would add to the array.
   * Once I had this set up, I turned my attention back to the HumanPane and ComputerPane. I added a new private method called addDicePairToList to both player pane classes and made a call to the method from inside the inner class TakeTurnListener’s handle method.
   * Both methods were very similar with the exception of the call for the Computer to setMaximumRolls. I knew I needed to clear the results from the list view for the last turn, so I used an if statement in each to check that player was the current player and the Turn Total was equal to 1, signifying the start of the turn. In the if statement, I cleared the contents of the list view. Then, I had another if statement that checked it the player was the current player and executed the original logic that was previously in the event listener handle method (setMaximumRolls() and play() for Computer and play() for Human), followed by a for loop that got the array list from the AbstractPlayer class for the player, looped through each result and added them to the list view.
   * The result was a scrollable list of dice pairs logged for each turn that the player can view, regardless of how many rolls are taken in the turn.