1. **Include a way to set the number of stones per pit at the start of the game. (2pts)**
   1. The parameter int numberOfStoneswas added to the startNewGame(Player, int) method. The startNewGame method takes the player and the numberOfStones and creates a new board and game with that info. A data member called int startingStoneswas also added to the game class to accommodate getters and setters for changing and retrieving the number of stones the player wants. Previously, the resetBoard() method had the number of starting stones hardcoded in as 1. This implementation of passing in the number of stones as a parameter to the method makes it dynamic.
   2. Secondly, a pane called NumberOfStonesPane in the MancalaPane portion of the view package was added. Inside this class the NumberOfStonesPane was built and an event handler was added to the setButton. When the setButton is clicked, the program will try to set the number of stones for the game, if the input into the textField is not an int, an error message will show and the user will be prompted to input an integer. During this phase of game setup, the other panes in the view are disabled, because getting a valid number of stones is integral to the game functioning without error. Once a valid integer has been given for the numberOfStones, it is parsed from a string, passed into the setNumberOfStartingStones(int) method and solidified as the startingStones data member for the game. The completion of this allows for the ChooseFirstPlayerPane to enable. The player chooses the first player, and with all items in the startNewGame(Player, int) fulfilled, that method is then called in the handle() method of human and computer radio button event listeners and the game begins.
   3. The code was implemented in this way because the startNewGame(Player, int) method needs both parameters to not be valid in order start the game without error. Disabling and enabling panes subsequently as the required info is gathered creates the most user-friendly experience.
2. **Implement the standard Mancala rules that says if the last stone you place is in your store, you get to go again.  Be sure to include some kind of notification that lets the user know what just happened. (4pts)**
   1. This modification required reworking of the distributeStonesFrom(int) method. A for loop with nested if logic was created to handle the distribution of stones, instead of the previous do while loop. Then, after each deposit, several things area checked via the determineTurnStatusUpdate(int, int, int, int) method. Chiefly, if the player gets an extra turn or if the player is eligible to capture an opponent’s stones in the opposite pit. The determination for an extra turn is decided in the determineIfGetExtraTurn(int, int) method by checking to see if the current player’s last stone position is in their store. If this returns true, the player gets another turn and, the data member turnStatusUpdate has content that reflects the free turn added to it, and is displayed in a label via the invalidated() function of the StatusPane by calling the getTurnStatusUpdate(). Otherwise, the turnStatusUpdate data member remains blank. If the game ends with a player landing in their pit, this label is not updated.
   2. The code was implemented in this way to promote modularity and ease of change in the future. With this structure it would be simple to add other rules around capture or storing. It will also be able to change dynamically, even if the amount of stones or size of the board changes.
3. **Implement the standard Mancala rule that says if the last stone you place is in an empty pit on your side of the board, you will capture (move to your store) any stones in the corresponding pit on the opposite side of the board.  Be sure to include some kind of notification that lets the user know what just happened. (4pts)**
   1. This modification also required reworking of the distributeStonesFrom(int) method. A for loop with nested if logic was created to handle the distribution of stones, instead of the previous do while loop. Then, after each deposit, several things area checked via the determineTurnStatusUpdate(int, int, int, int) method. Chiefly, if the player gets an extra turn or if the player is eligible to capture an opponent’s stones in the opposite pit. If the determineIfGetExtraTurn(int, int) method returns false, the determineTurnStatusUpdate(int, int, int, int) method will check to see if the player is eligible to capture from the opponent’s side of the board. The following conditions must be met in order to capture from the opponent’s pit:
      1. The player cannot be on a store pit
      2. The player must be out of stones for the turn
      3. The current pit must only have the one stone in it that ended the turn
      4. There must be a non-empty pit directly opposite on the opponent’s side.
      5. The pit you ended in is on the current player’s side of the board.
   2. If these conditions are all met, then the capture code block is executed. The capture block takes the stones from the current pit and the opponent’s pit opposite the current pit and puts them in the players store. Lastly, the data member turnStatusUpdate has content that reflects the free turn added to it, and is displayed in a label via the invalidated() function of the StatusPane by calling the getTurnStatusUpdate(). Otherwise, the turnStatusUpdate data member remains blank.
   3. The code was implemented in this way to promote modularity and ease of change in the future. With this structure it would be simple to add other rules around capture or storing. It will also be able to change dynamically, even if the amount of stones or size of the board changes.