Design and Development

ArrayBag

We started off by creating our custom data structure, called ArrayBag. This was similar to the one we made in class, but we have decided it should only include methods which we needed for our scenario. If we were to use this data structure for other projects, it would have made sense to include more of the usual methods a data structure would have. Since this data structure was made explicitly for this scenario, we felt it was unnecessary. Our ArrayBag class contains one constructor method, a get method for the number of elements it contains, an add method, a method which removes the element at a specified index, a method that returns the element at a specified index, and a method that sets the value of the element at a specified index.

Card

The next step was to create the Card class. In this scenario, a card is made unique by a combination of two attributes: a rank name, and a suit name. There is a constructor method which takes these two attributes as parameters. We have decided it would be nice if we didn't have to implement yet another variable for rank value, and instead, it would be automatically calculated from the rank name by a method. We used switch case in a method for calculation, and a get method which calls this calculator method. It is worth noting that this decision assumes that an non-existent rank name won't be given as a parameter when calling this method, but since we did the Deck class too, it was obvious we would not do such a thing. There are three additional get methods for the rank name, suit name, and card name, which is the combination of the two.

Deck

For most of the Deck class, all we do is introduce an ArrayBag type variable called deck and fill it up with Card type variables. We felt it would have been unnecessary using for loops for calculating the combinations of ranks and suits and filling up the deck with these. Maybe it would have made the code a little bit shorter but more complicated overall, thus we simply defined all of the 52 cards manually. Similar to the ArrayBag class, this class also has a getSize, a removeAt and an elementAt method, which simply call the ArrayBag methods of the same name. Lastly, there is a method for shuffling the deck. In this method, there is a temporary deck being introduced and we take a random number between 0 and the current deck size, and put the element with that index from the original deck to the temporary deck. This is being done until there is nothing left from the original deck. After this, we make the original deck equal with the temporary deck. This decision involves occupying additional addresses in the memory, which might not be the most efficient solution for this task, but it certainly is a convenient one.

Elevens (main class)

The Elevens class contains a Scanner object for handling input, a few outputs to prompt for an input, an ArrayBag of Cards for the player's hand, Deck (ArrayBag of 52 Cards) for the deck and an ArrayBag of String for the replay. There is also a String which keeps track of the chosen game mode. More on that later.

All that our **main** method does is call the newGame method. The reason we start the actual code with the newGame method instead of the main method itself is because newGame method can be called.

The **newGame** method simply waits for the user input and directs the player to the next corresponding method accordingly, or throws them back to the newGame method in case of an invalid choice. If the user chooses the player vs computer or the demonstration mode, the choice gets stored in the gameMode string, then both choices direct the player to the same Play method. This is done this way because the algorithm is largely the same for both cases and the few differences of each will be executed later depending on an if condition taking a look at the gameMode variable.

The **Play** method creates a deck, shuffles it, then draws from it into the player's hand. Then startTurn gets called.

printDeckAndHand is a method used to output the cards in the player's hands in each turn. These output Strings are also stored in the String ArrayBag of replay.

The **hint** method checks for pairs or trios in the player's hand. It can have three results: an array of three elements (Jack, Queen, King), an array of two elements (number pairs) or an array of zero elements (Stalemate).

startTurn calls this printDeckAndHand method, and then checks for a stalemate. This is done by introducing an array variable and assigning the length of the array returning from the hint method to it. If this value is 0, the gameOver method gets called with a specific String reason as a parameter. Otherwise we examine the gameMode string and either direct the player to the turnMenu to choose, or call the selectDemo if it is the computer playing.

The **turnMenu** method is very similar to the newGame method, but with its own choices: select (player vs computer), printHint (the player asked for a hint), newGame (the player wants to start over) or turnmenu (invalid choice).

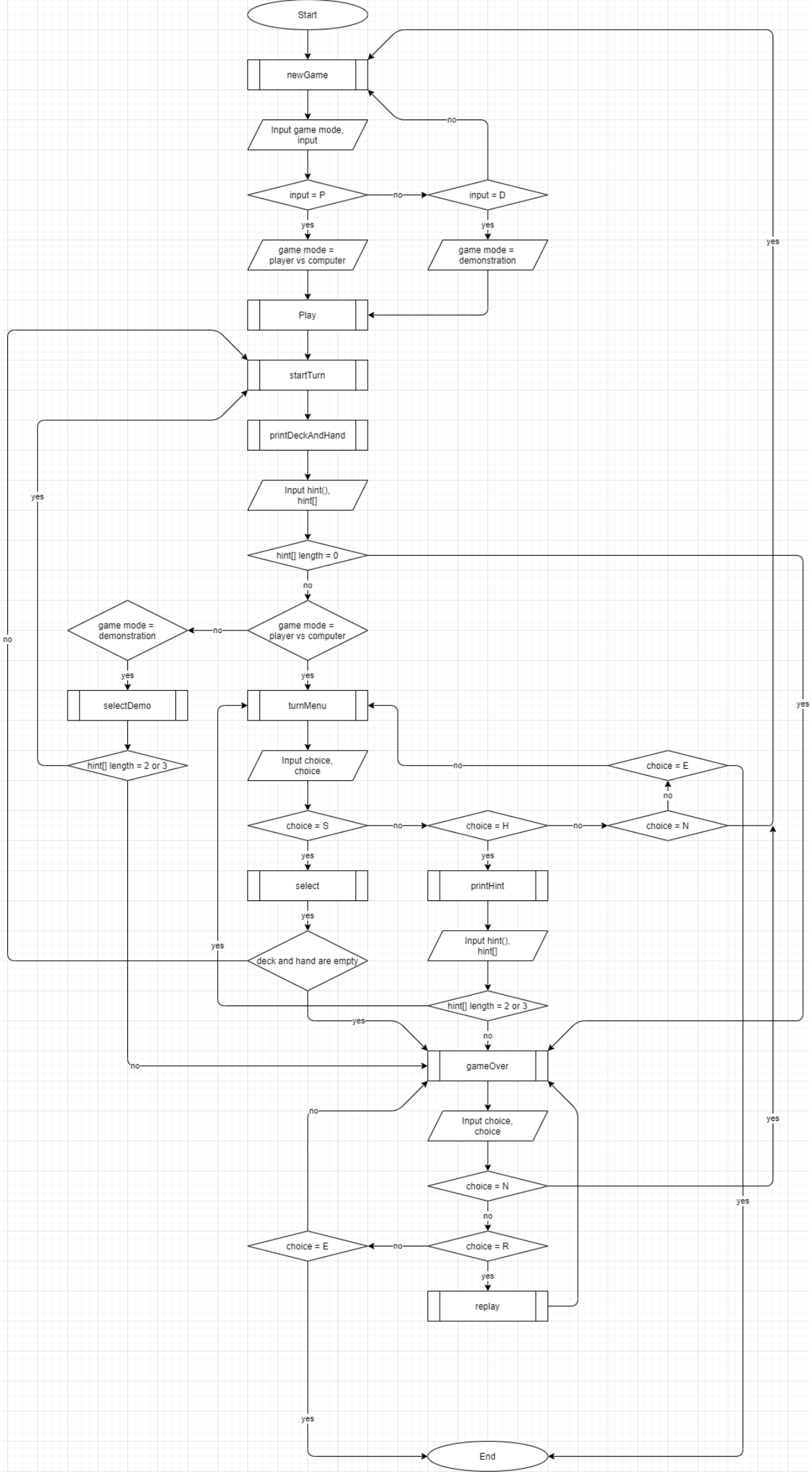
The **gameOver** method, like newGame and turnMenu, lists a number of options for the player: newGame, or gameOver (invalid choice).

The **select** method is the core player vs computer method. It prompts the player to select two cards, checks the selected cards and either prompts for a third card accepts it or refuses it if it is not a pair or trio. If the selected cards are accepted, the choice variables get sorted in a descending order, then the Cards in the player's hand with those indices get removed and substituted by new cards from the deck, if there is any left. A check for game over also happens here, in case both the deck and the hand is empty. This time, however, the string parameter for the gameOver method is that the player has won. Each time there are cards being removed, the corresponding removal message is added to the replay ArrayBag.

The **selectDemo** method is the computer equivalent of the select method. Here the computer plays automatically, checks for hints with calling the hint method, executes those hints, fills up the replay ArrayBag, prints the corresponding messages and basically uses the same code the select method does, except for the prompting for user input part.

Even though the hint method is used both by the computer for calculating cards to remove, and by the player upon request, it only gets printed out in the latter case. This is done by the **printHint** method, which is really not more than what it sounds like.

Upon choosing it from the gameOver menu, the **replay** method goes through the replay ArrayBag with a for loop and prints out each turn's happenings, and waits for user input between them. After the for loop ends, the gameOver method gets called yet again.


Start
newGame
Input game mode, input
input = P
input = D
game mode = player vs computer
game mode = demonstration
Play
startTurn
printDeckAndHand
Input hint(), hint[]
hint[] length = 0
game mode = demonstration
game mode = player vs computer
selectDemo
turnMenu
hint[] length = 2 or 3
Input choice, choice
choice = E
choice = S
choice = H
choice = N
select
printHint
deck and hand are empty
Input hint(), hint[]
hint[] length = 2 or 3
gameOver
Input choice, choice
choice = N
choice = E
choice = R
replay
End
yes
no