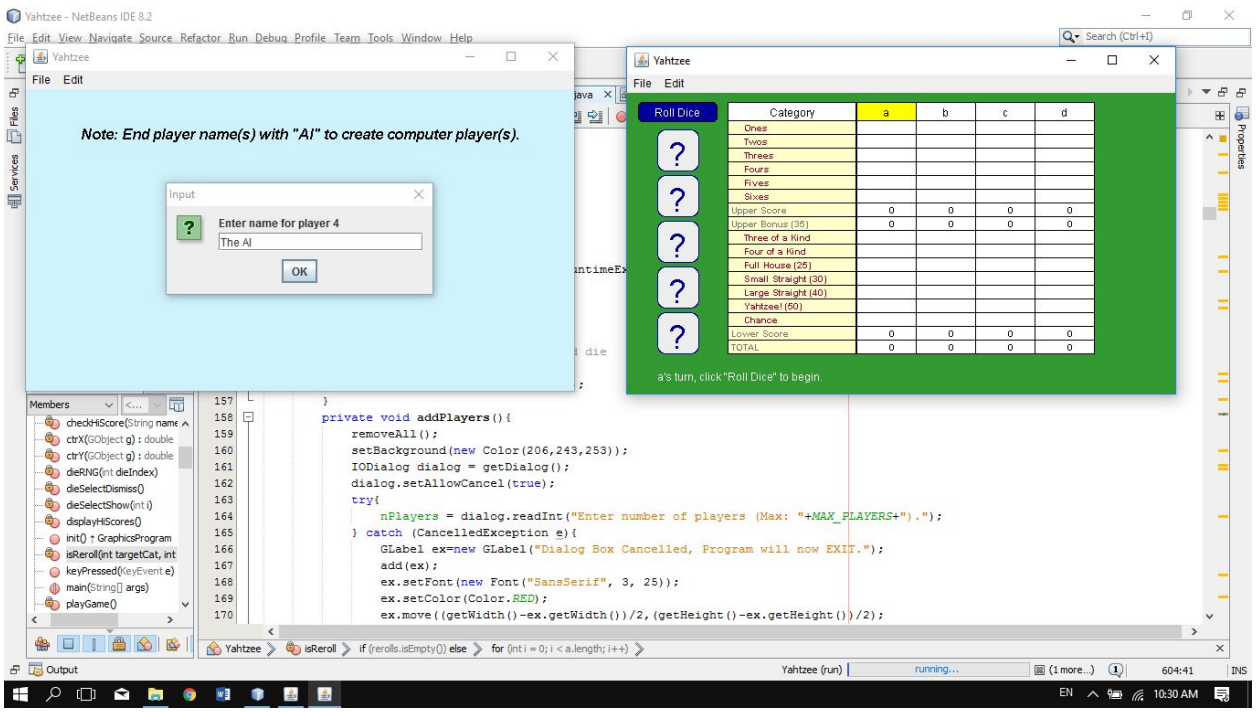
LBYCP12

Data Structures and Algorithm Analysis Laboratory



Laboratory Activity 4

Yahtzee!

By

Chan, Patrick Matthew J, LBYCP12-EQ1

INTRODUCTION

The main goal of this activity is to give students a better mastery in creating and manipulating data arrays, to use be able to create applications from a new library with only its documentation, and to integrate a List ADT Data Structure, mainly in the form of a High Scores list at the end of the game. The activity also allows the student to extend the basic assignment and further improve the code of the program.

OBJECTIVES

- To use arrays in storing and manipulating huge sets of data
- To correctly display the necessary data in the Yahtzee! Display, using the attached yahtzeelib.jar, aside from acm.jar
- To learn how to correctly use a library by only reading its documentation
- To integrate a List ADT in the creation of the program
- Program a complete computer version of the Yahtzee™! Game by HasBro
- To learn how to set out the proper logic for the computer to determine if a certain combination of dice is valid for a given category, and return its equivalent score

MATERIALS

1. Java SE Development Kit (JDK) 8
2. NetBeans integrated development environment (IDE)
3. `acm.jar` by the ACM Java Task Force
4. `yahtzeelib.jar` by the ACM Java Task Force

PROCEDURE

1. Learn how to play Yahtzee!, and how the scoring is done per category.

There are five dice and one to four
players. A round of the game consists of each player taking a turn. On each turn, a player rolls the five dice with the hope of getting them into a configuration that corresponds to one of 13 categories (see the following section on “Dice Categories”). If the first roll doesn’t get there, the player may choose to roll any or all of the dice again. If the second roll is still unsuccessful, the player may roll any or all of the dice once more. By the end of the third roll, however, the player must assign the final

dice configuration to one of the thirteen categories on the scorecard. If the dice configuration meets the criteria for that category, the player receives the appropriate score for that category; otherwise the score for that category is 0. Since there are thirteen categories and each category is used exactly once, a game consists of thirteen rounds. After the thirteenth round, all players will have received scores for all categories. The player with the total highest score is declared the winner.

Dice Categories
The thirteen categories of dice configurations and their scores are:
1. Ones. Any dice configuration is valid for this category. The score is equal to the sum of
all of the 1's showing on the dice, which is 0 if there are no 1's showing.
2–6. Twos, Threes, Fours, Fives, and Sixes. (same as above but for different values). Any
dice configuration is valid for these categories. The score is equal to the sum of the 2's, 3's, 4's, and so on, showing on the dice.
7. Three of a Kind. At least three of the dice must show the same value. The score is equal
to the sum of all of the values showing on the dice.
8. Four of a Kind. At least four of the dice must show the same value. The score is equal to the sum of all of the values showing on the dice.
9. Full House. The dice must show three of one value and two of another value. The score
is 25 points.
10. Small Straight. The dice must contain at least four consecutive values, such as the sequence 2-3-4-5. The score is 30 points.
11. Large Straight. The dice must contain five consecutive values, such as the sequence 1-2-
3-4-5. The score is 40 points.
12. Yahtzee! All of the dice must show the same value.. The score is 50 points.
13. Chance. Any dice configuration is valid for this category. The score is equal to the sum
of all of the values showing on the dice.

2. Read the documentation and learn how to use yahtzeelib.jar.
3. Take note of the constants in the included YahtzeeConstants.java file.

```
/*
* File: YahtzeeConstants.java
* ---------------------------
```

```
* This file declares several constants that are shared by the
* different modules in the Yahtzee game.
*/
public interface YahtzeeConstants {
/** The width of the application window */
public static final int APPLICATION_WIDTH = 600;
/** The height of the application window */
public static final int APPLICATION_HEIGHT = 350;
/** The number of dice in the game */
public static final int N_DICE = 5;
/** The maximum number of players */
public static final int MAX_PLAYERS = 4;
/** The total number of categories */
public static final int N_CATEGORIES = 17;
/** The number of categories in which the player can score */
public static final int N_SCORING_CATEGORIES = 13;
/** The constants that specify categories on the scoresheet */
public static final int ONES = 0;
public static final int TWOS = 1;
public static final int THREES = 2;
public static final int FOURS = 3;
public static final int FIVES = 4;
public static final int SIXES = 5;
public static final int UPPER_SCORE = 6;
public static final int UPPER_BONUS = 7;
public static final int THREE_OF_A_KIND = 8;
public static final int FOUR_OF_A_KIND = 9;
public static final int FULL_HOUSE = 10;
public static final int SMALL_STRAIGHT = 11;
public static final int LARGE_STRAIGHT = 12;
public static final int YAHTZEE = 13;
public static final int CHANCE = 14;
public static final int LOWER_SCORE = 15;
public static final int TOTAL = 16;
}
```

4. Program the game so that:

   On each turn, players must

   i. Click on the Roll Dice button to set up the initial roll of all five dice.
   ii. Select a set of dice and then click the Roll again button to reroll the selected dice.
   iii. Repeat step 2 to generate the final dice configuration after the third

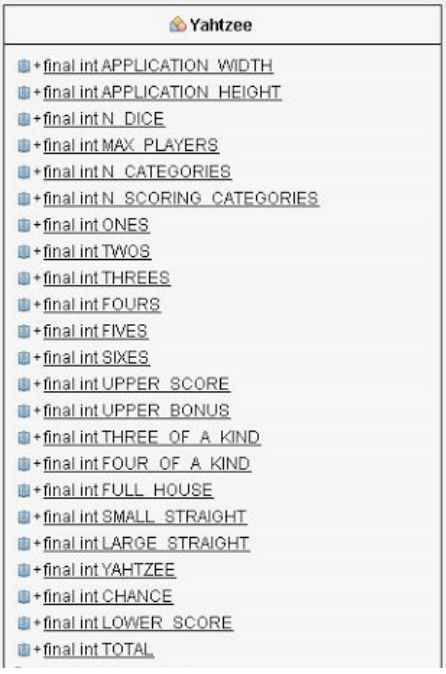
roll.

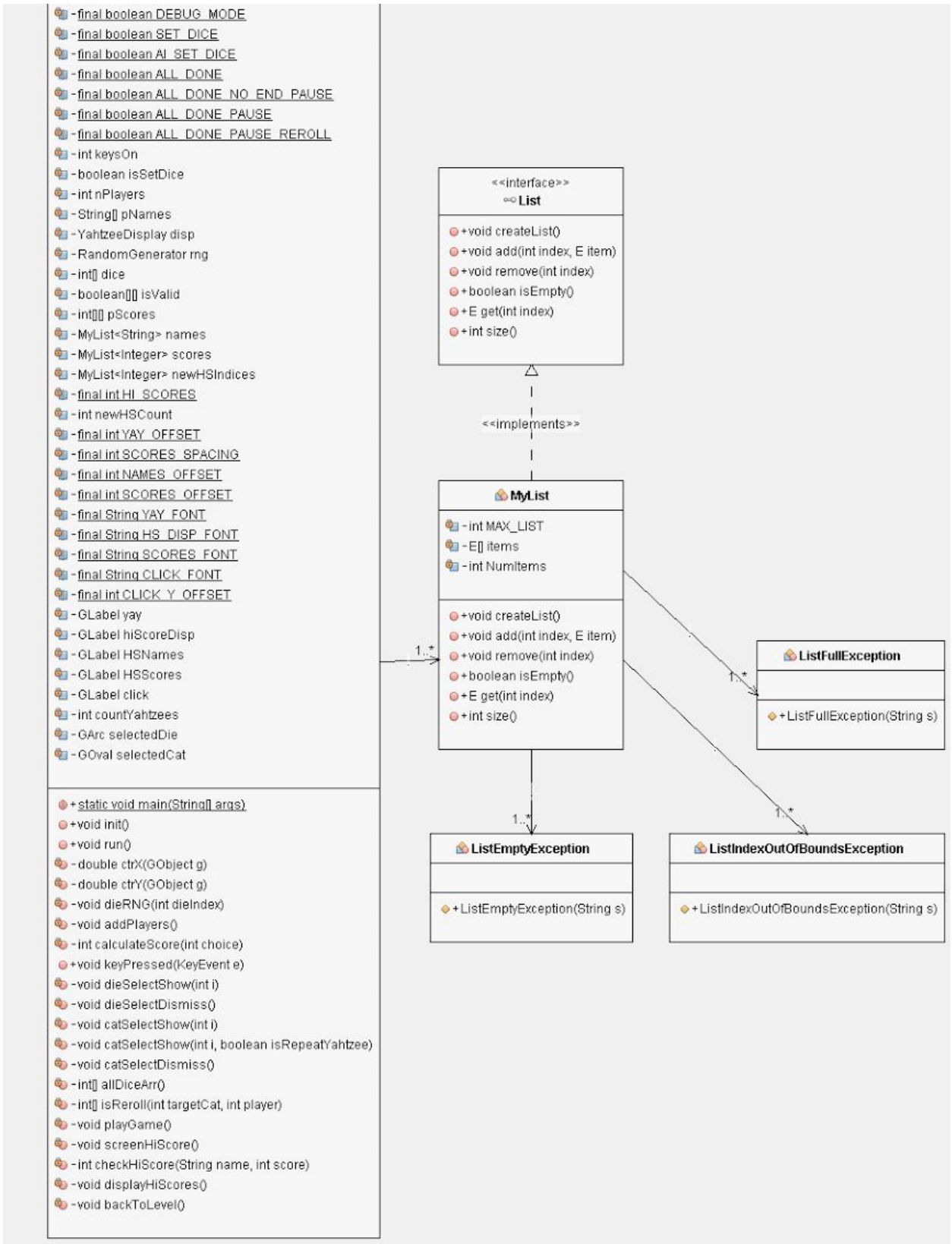
iv. Click on a category to store the score in the appropriate box.

5. Use a random number generator to set the values for the dice
6. Code a checking system that sees whether the dice roll satisfies the chosen category, and give a score of zero if it does not.
7. Program the appropriate scoring system for each category
8. Make sure that the player cannot choose the same category twice by telling the player whenever an invalid category is chosen, and ignoring this input in the program.
9. Make sure to update the scorecard whenever a player's turn ends.
10. Add a code to check whether a player is eligible for the Upper Bonus.
11. When the game ends, the game should be able to know who won and print it out, together with his/her score.
12. Add a high score list at the end of the game, with the use of a List ADT.

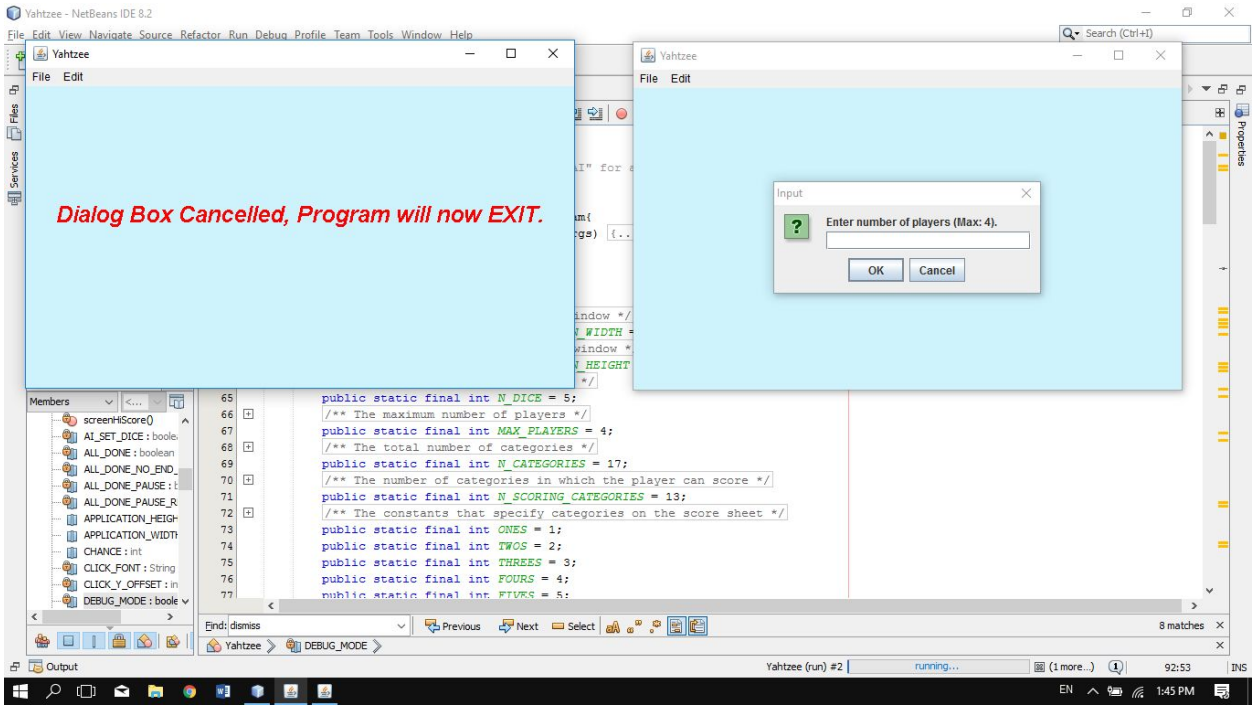
RESULTS AND DISCUSSION

The UML

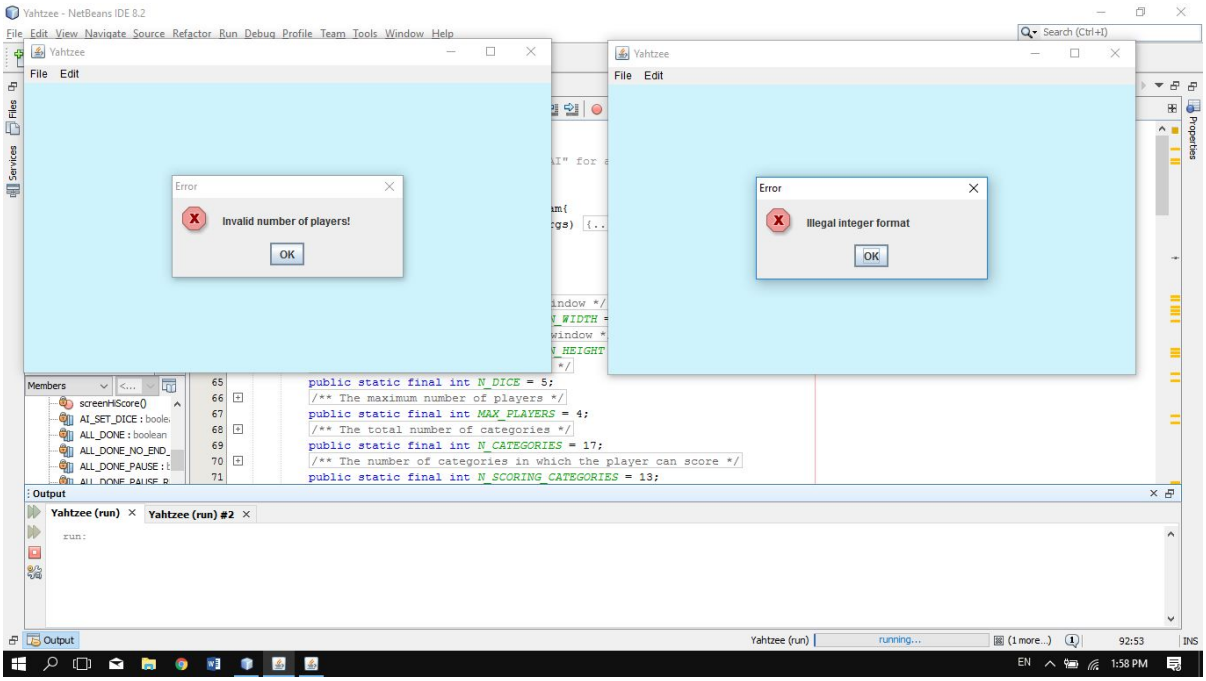



-final boolean DEBUG_MODE
-final boolean SET_DICE
-final boolean AI_SET_DICE
-final boolean ALL_DONE
-final boolean ALL_DONE_NO_END_PAUSE
-final boolean ALL_DONE_PAUSE
-final boolean ALL_DONE_PAUSE_REROLL
-int keysOn
-boolean isSetDice
-int nPlayers
-String[] pNames
-YahtzeeDisplay disp
-RandomGenerator rng
-int[] dice
-boolean[][] isValid
-int[][] pScores
-MyList<String> names
-MyList<Integer> scores
-MyList<Integer> newHSIndices
-final int HI_SCORES
-int newHSCount
-final int YAY_OFFSET
-final int SCORES_SPACING
-final int NAMES_OFFSET
-final int SCORES_OFFSET
-final String YAY_FONT
-final String HS_DISP_FONT
-final String SCORES_FONT
-final String CLICK_FONT
-final int CLICK_Y_OFFSET
-GLabel yay
-GLabel hiScoreDisp
-GLabel HSNames
-GLabel HSScores
-GLabel click
-int countYahtzees
-GArc selectedDie
-GOval selectedCat
+static void main(String[] args)
+void init()
+void run()
-double ctrX(GObject g)
-double ctrY(GObject g)
-void dieRNG(int dieIndex)
-void addPlayers()
-int calculateScore(int choice)
+void keyPressed(KeyEvent e)
-void dieSelectShow(int i)
-void dieSelectDismiss()
-void catSelectShow(int i)
-void catSelectShow(int i, boolean isRepeatYahtzee)
-void catSelectDismiss()
-int[] allDiceArr()
-int[] isReroll(int targetCat, int player)
-void playGame()
-void screenHiScore()
-int checkHiScore(String name, int score)
-void displayHiScores()
-void backToLevel()
1..*
<<interface>>
List
+void createList()
+void add(int index, E item)
+void remove(int index)
+boolean isEmpty()
+E get(int index)
+int size()
<<implements>>
MyList
-int MAX_LIST
-E[] items
-int NumItems
+void createList()
+void add(int index, E item)
+void remove(int index)
+boolean isEmpty()
+E get(int index)
+int size()
1..*
ListFullException
+ListFullException(String s)
1..*
1..*
ListEmptyException
+ListEmptyException(String s)
ListIndexOutOfBoundsException
+ListIndexOutOfBoundsException(String s)

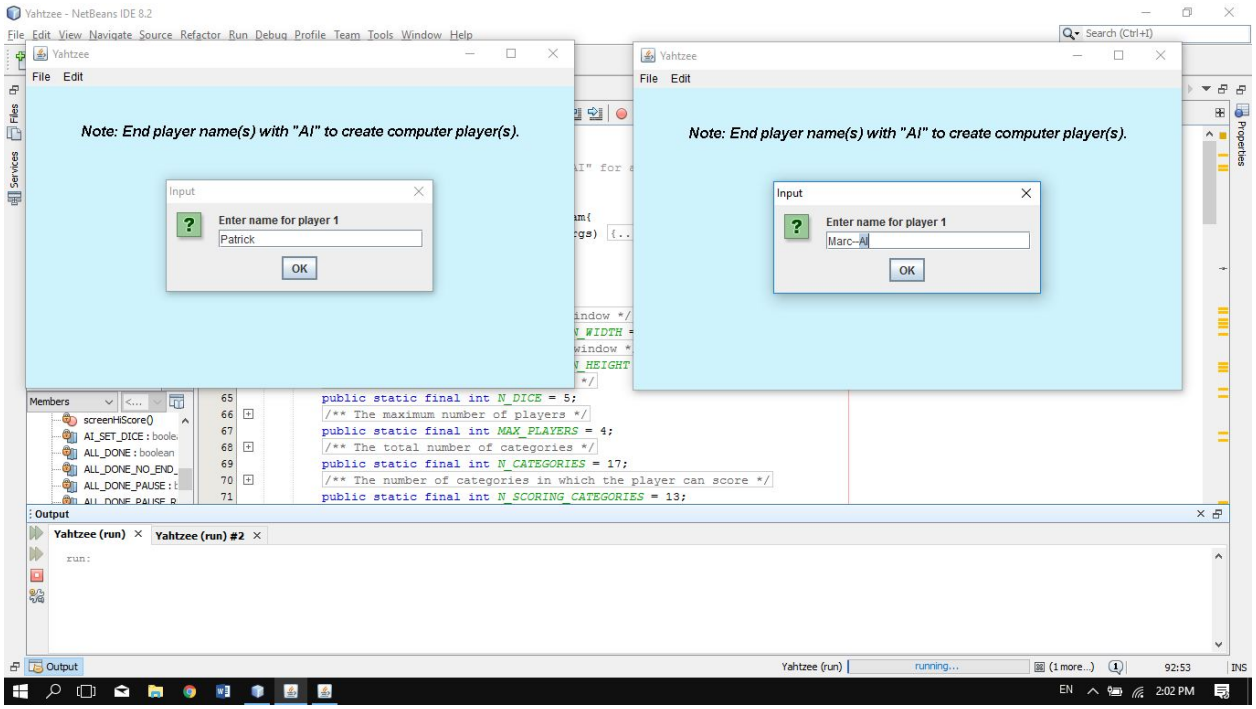
Result--Screenshots



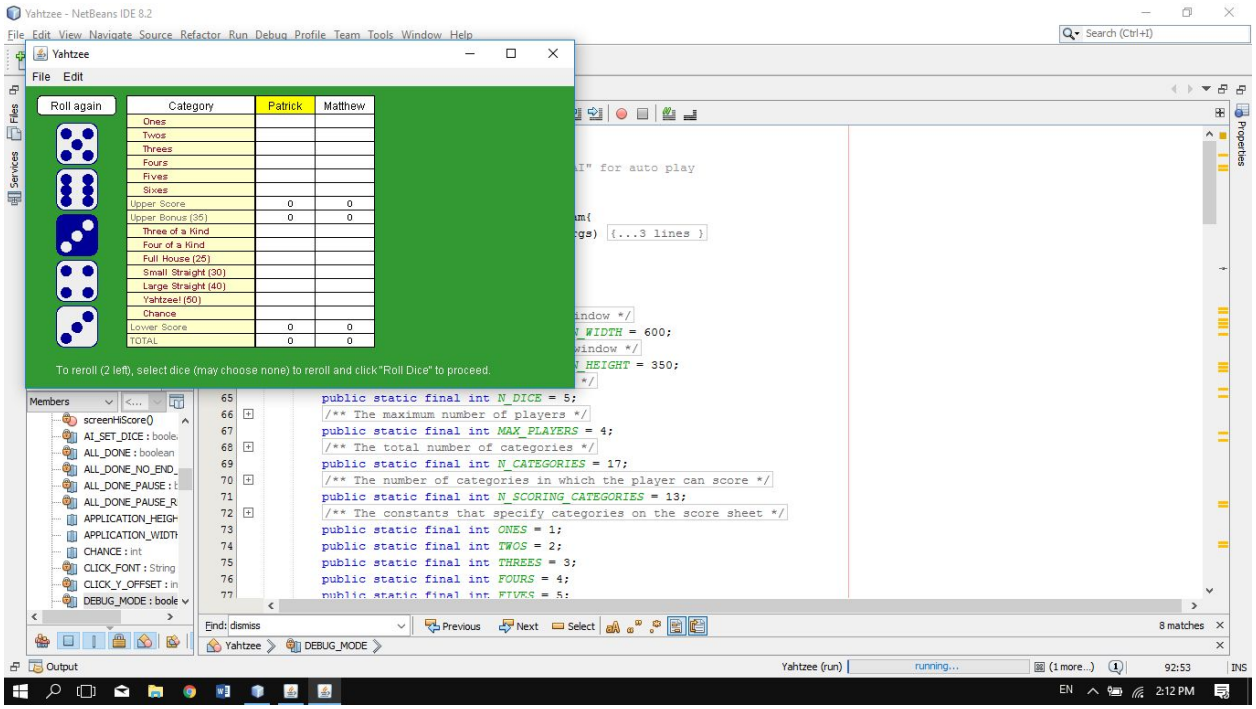
When the program starts, it shows a dialog box that asks for the number of players, which, when cancelled, signals the program to close instead.



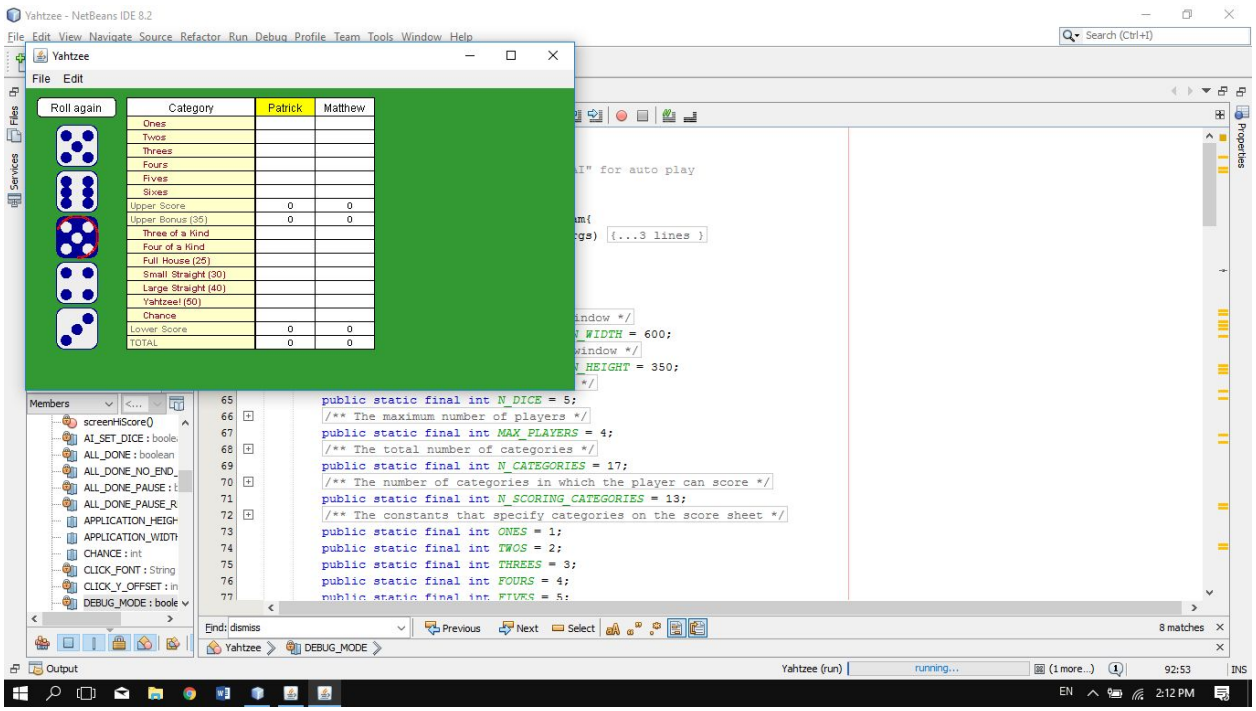
Program checks the input for its validity (must be the integers 1,2,3 or 4 only), and reprompts the user when the input is invalid.



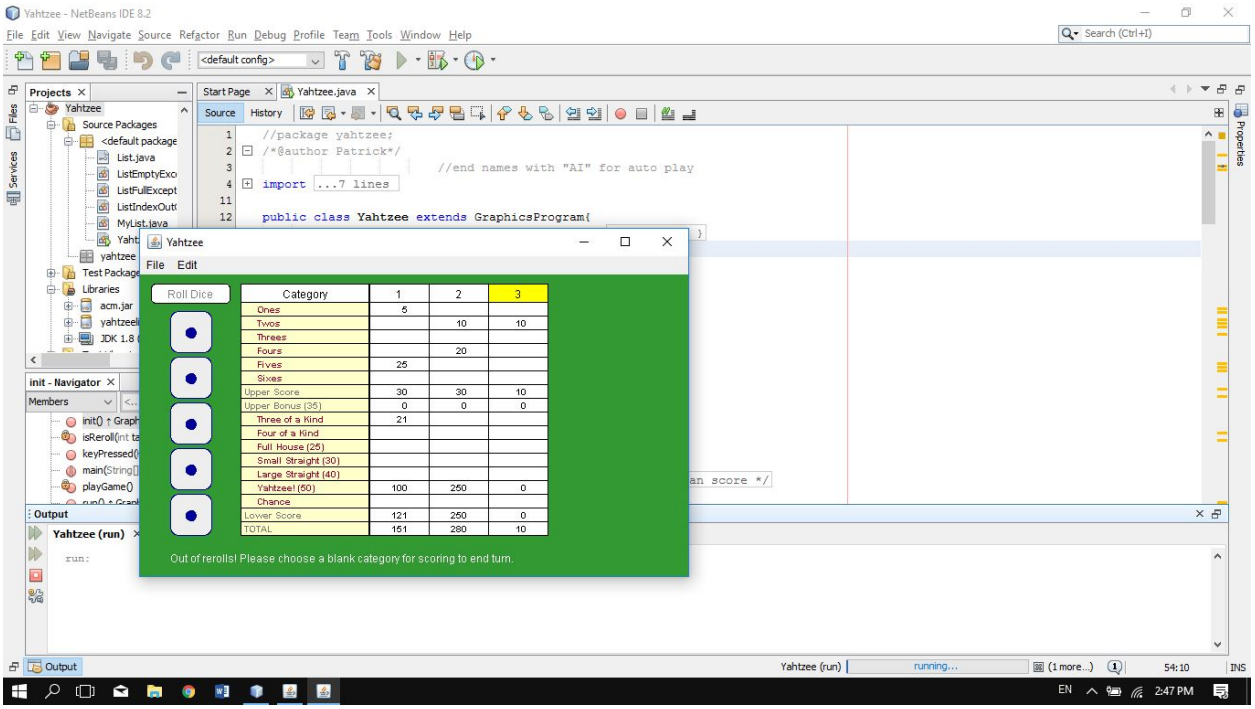
Users have the option to create AI (or computer controlled) players. The program supports having all human players, human vs AI, or all AI players.



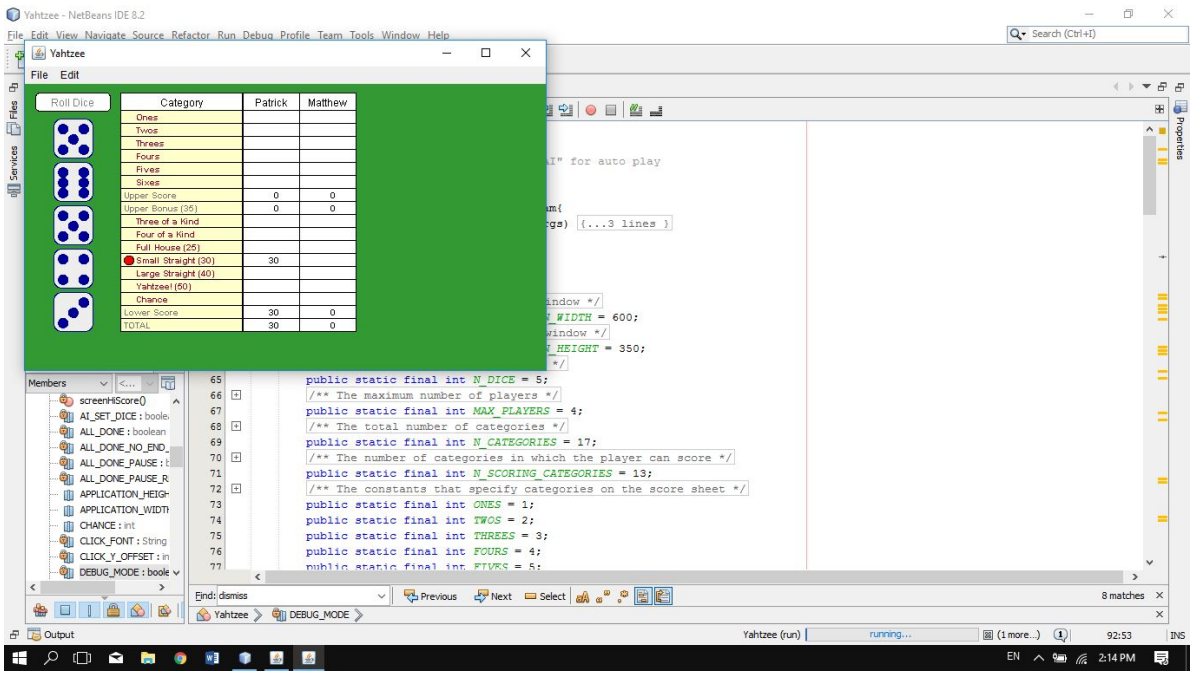
Program shows that player has 2 more rerolls, and player can choose which dice to reroll



A circling animation is used to indicate the dice about to be rerolled

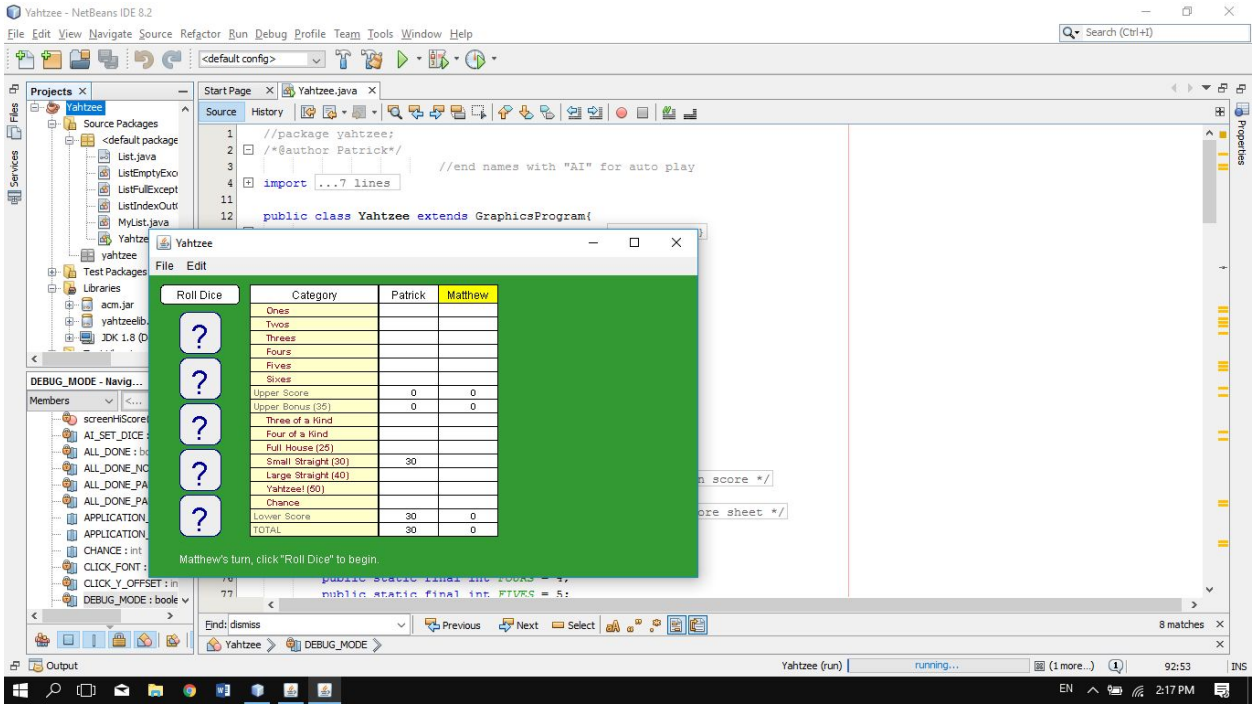


When the player is out of rerolls, the “Roll” button is disabled, and the player must choose a category to proceed.

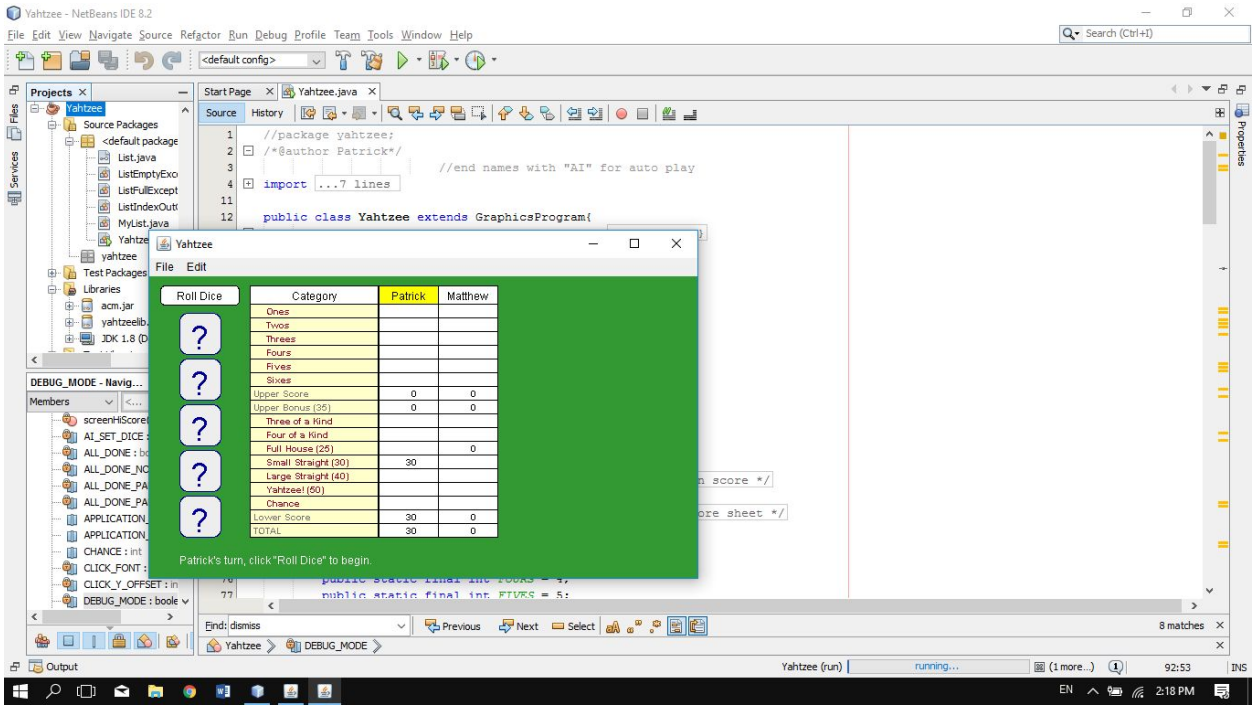


When the player clicks a category, score is credited when the dice results fit in the chosen category. After updating the scoreboard, a red dot is briefly shown on that

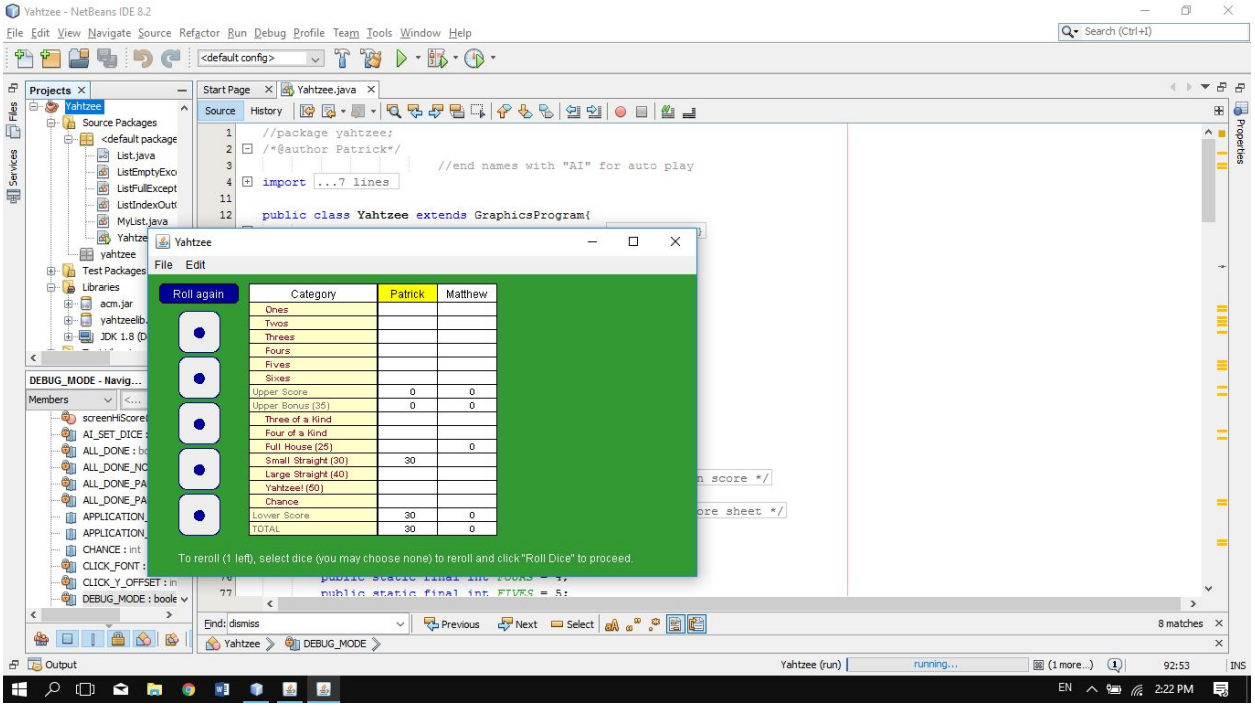
category on the scoreboard to indicate the update.



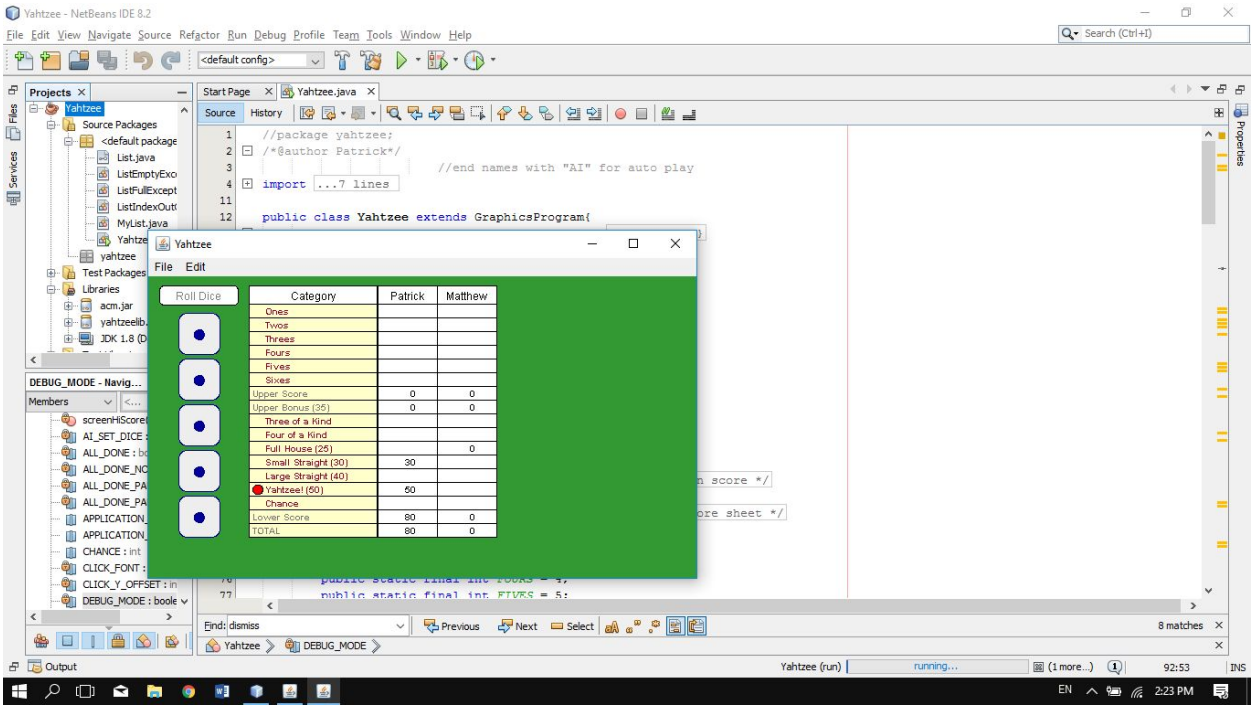
Player 1's turn ends, and Player 2 is now instructed to roll the dice.



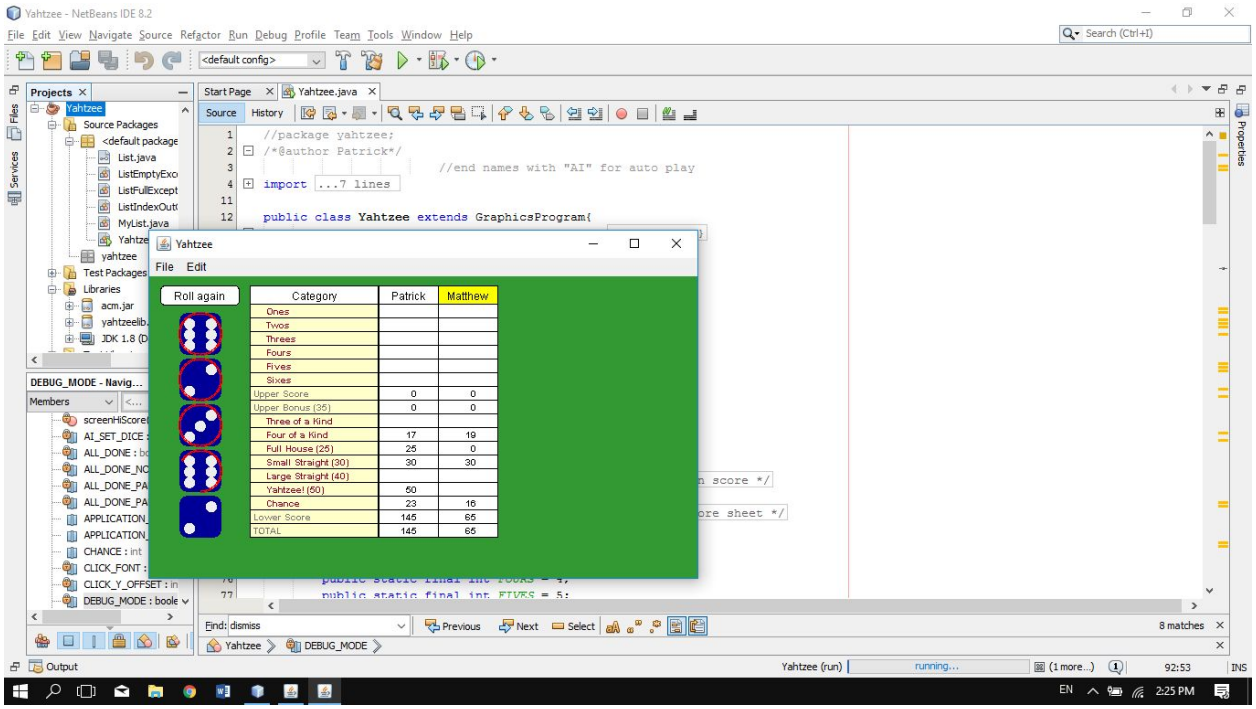
When the dice result does not fit in the category, a score of zero is given instead.



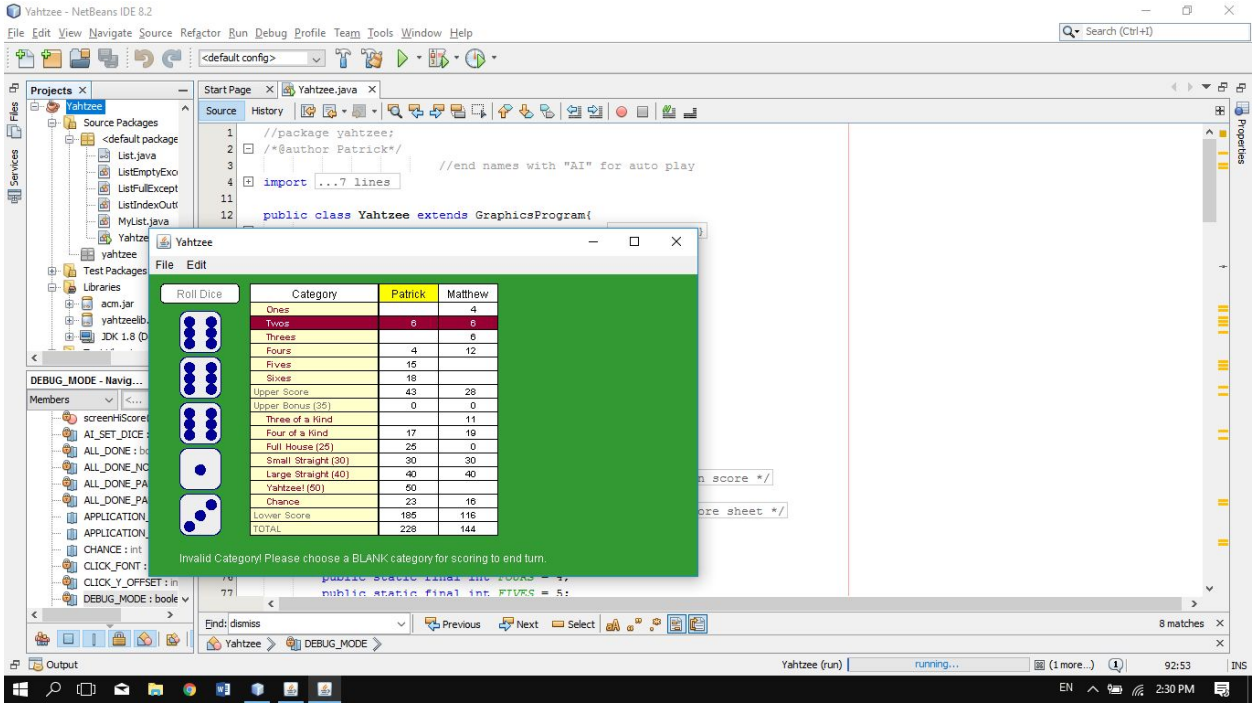
When the player doesn't want to reroll any of the dice, he can choose none, and proceed with the game by clicking "Roll Again"



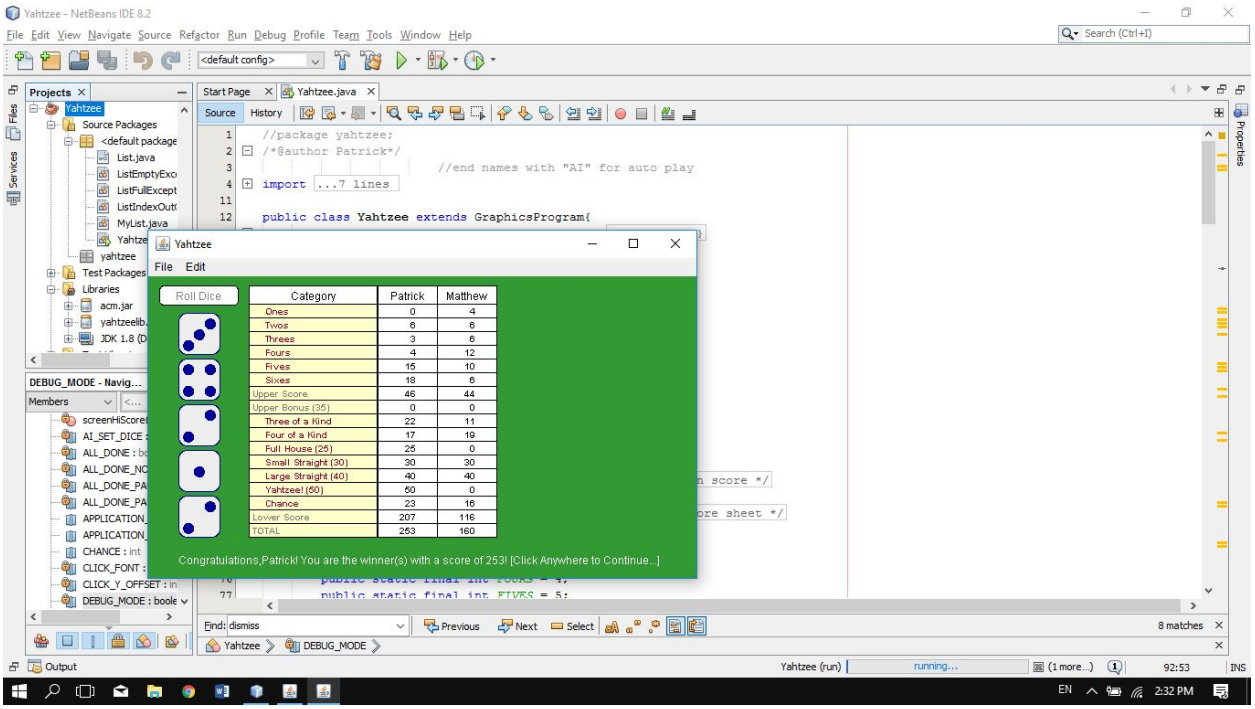
This shows that player 1 has gotten a Yahtzee, and the score is credited accordingly.



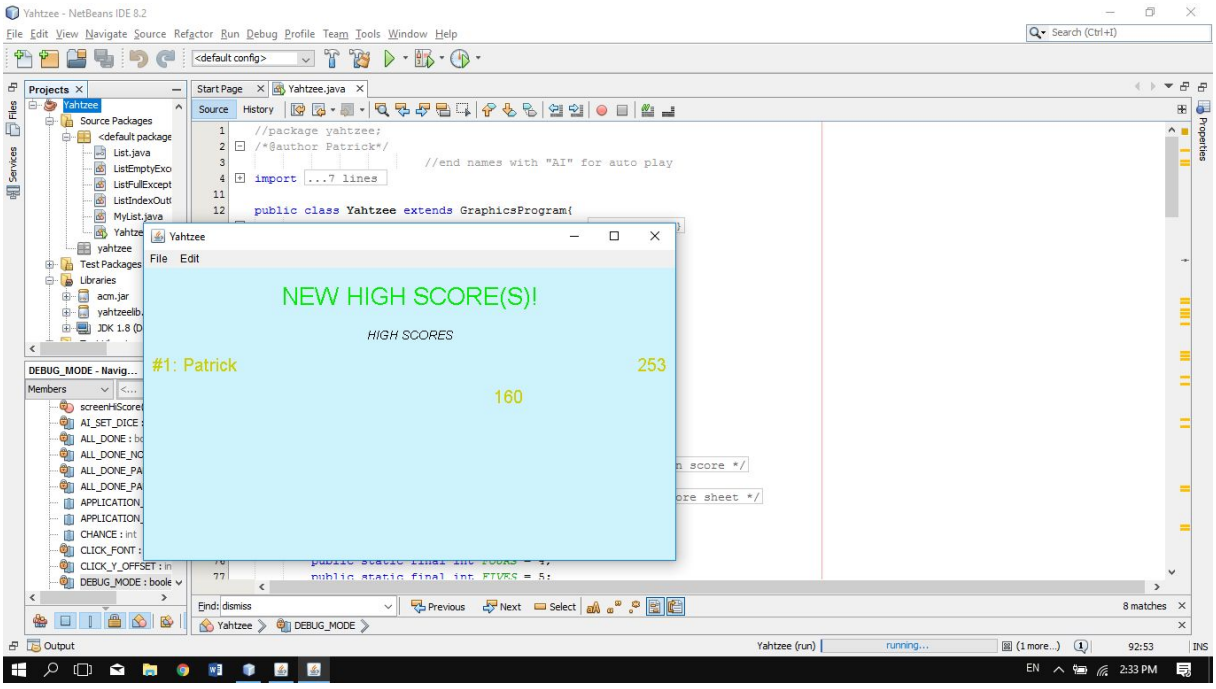
The gameplay continues in this manner until all the categories are filled.



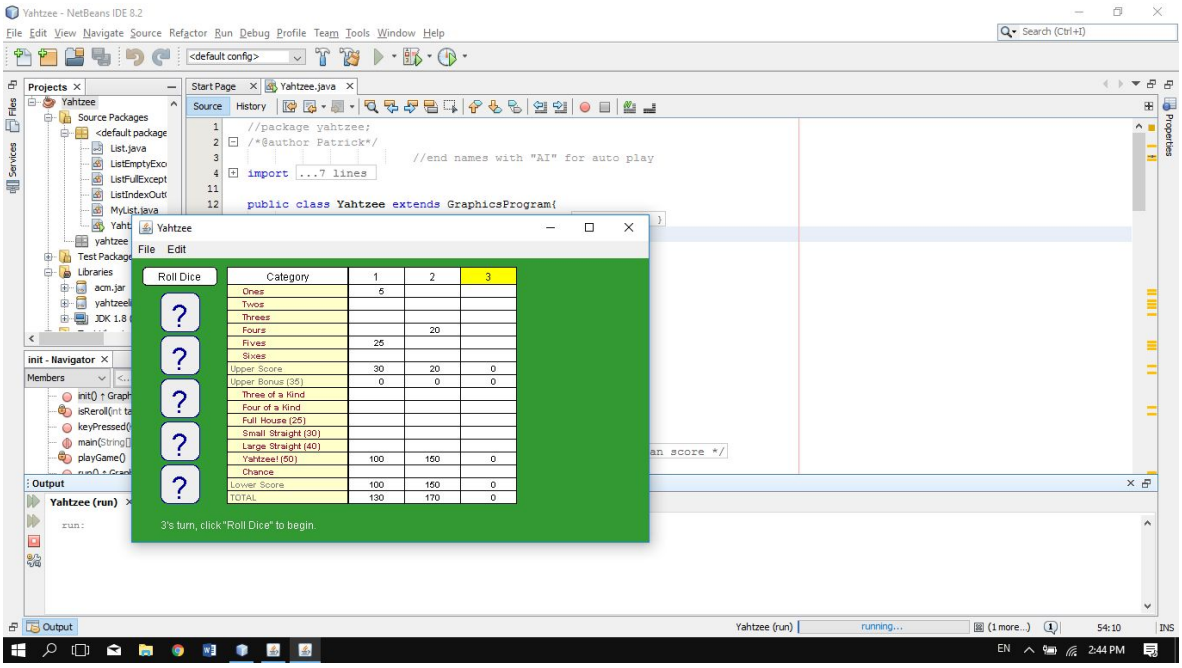
The player is informed when an invalid(occupied) category is chosen.



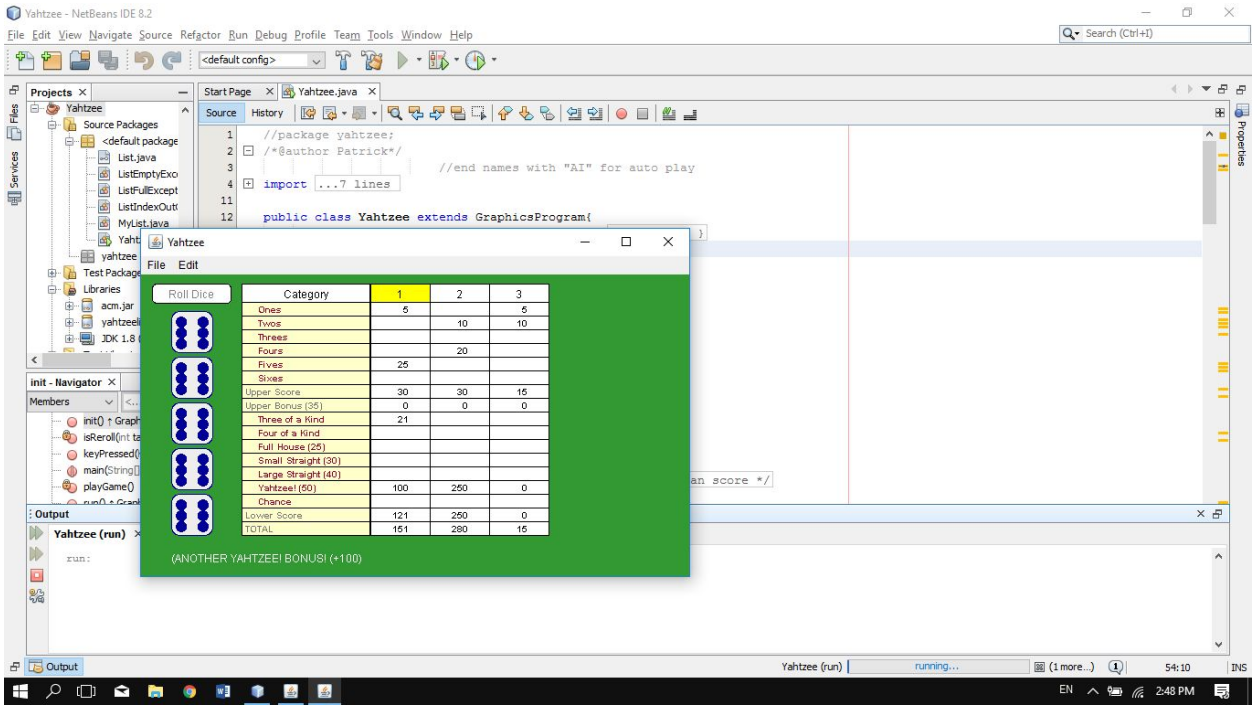
When all the categories are filled, the game ends. The program chooses the winner by score, and displays it below.



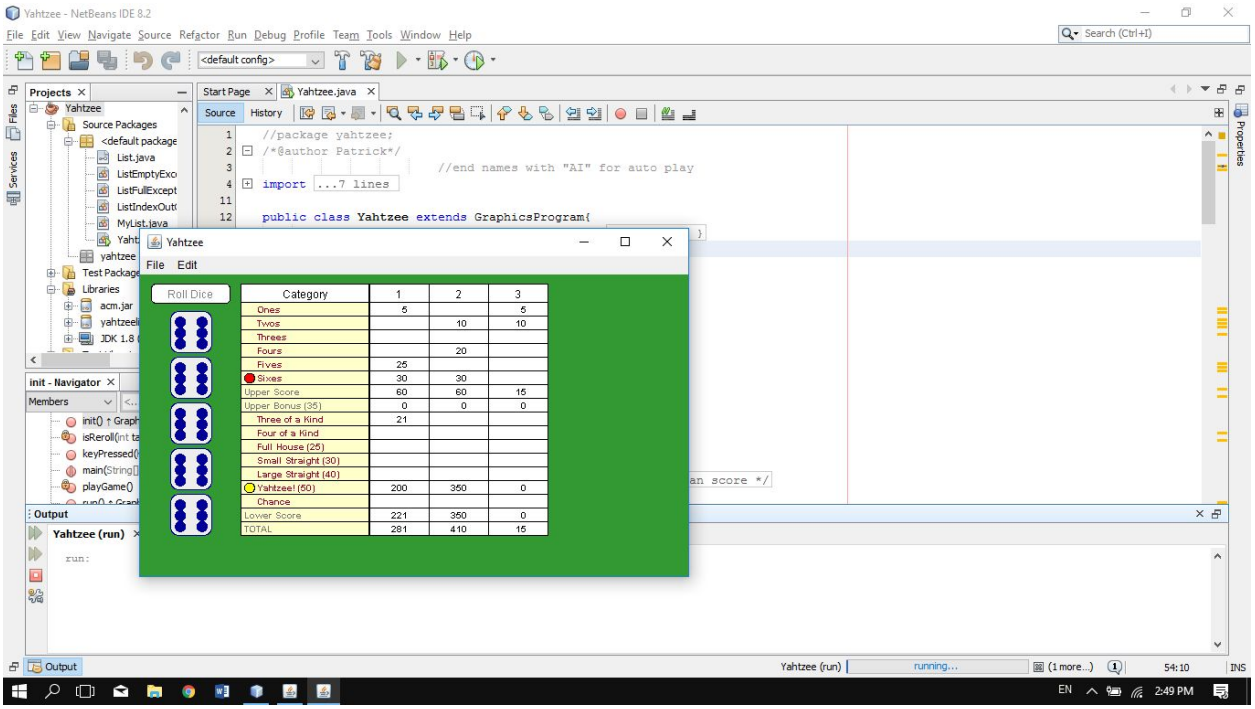
The high-score screen is then displayed, where each entry is animated in coming to the list. Each player's score is checked, to be added to the list.



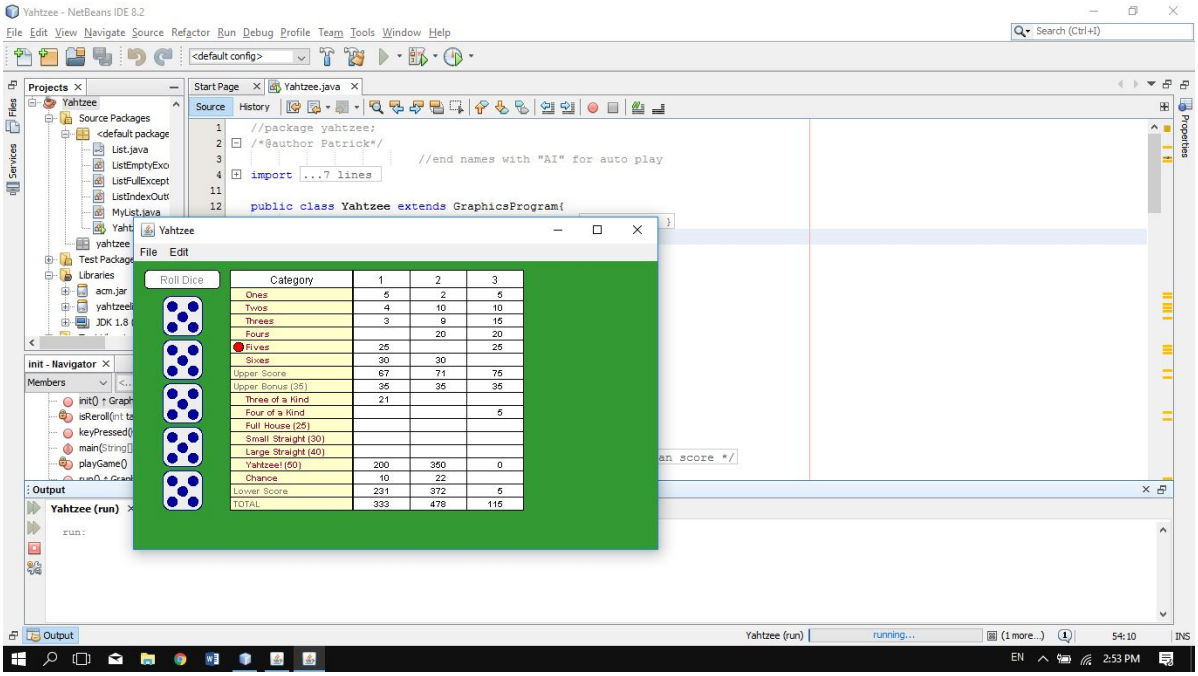
When a player gets another Yahtzee, as long as the Yahtzee category has not been previously filled with a zero, the player gets an additional bonus 100 points.



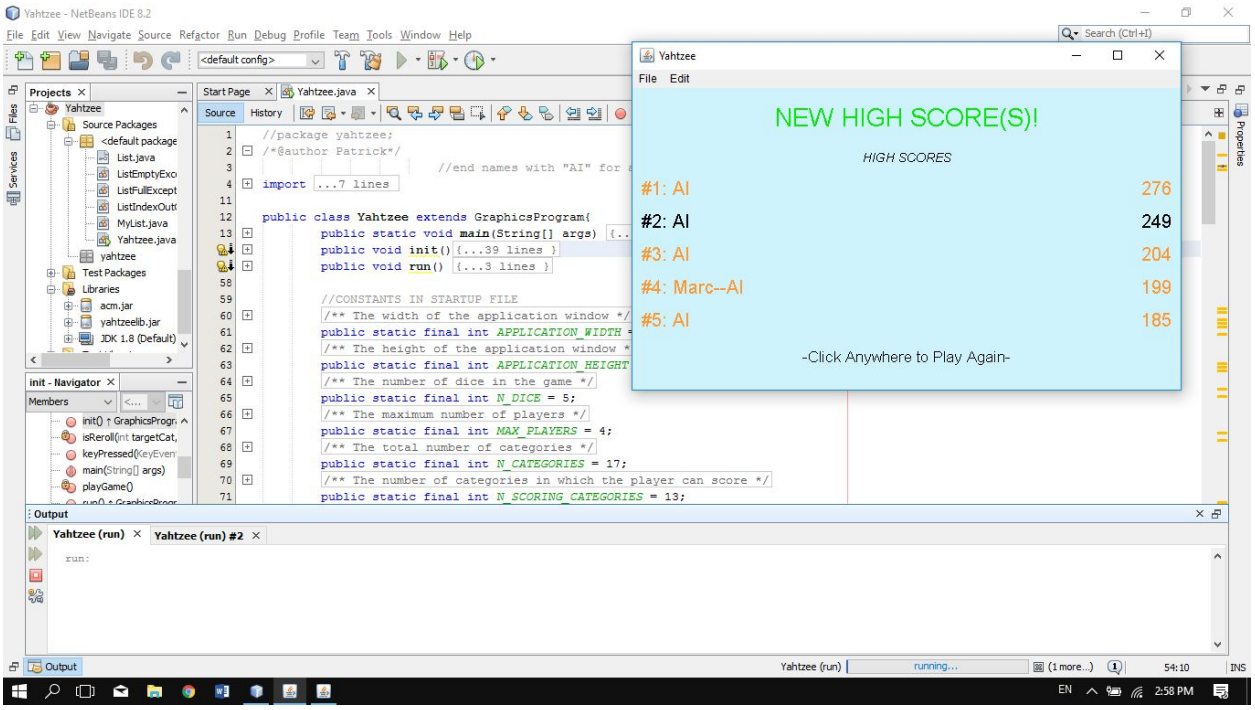
Program briefly pauses, and indicates that a yahtzee bonus has been awarded.



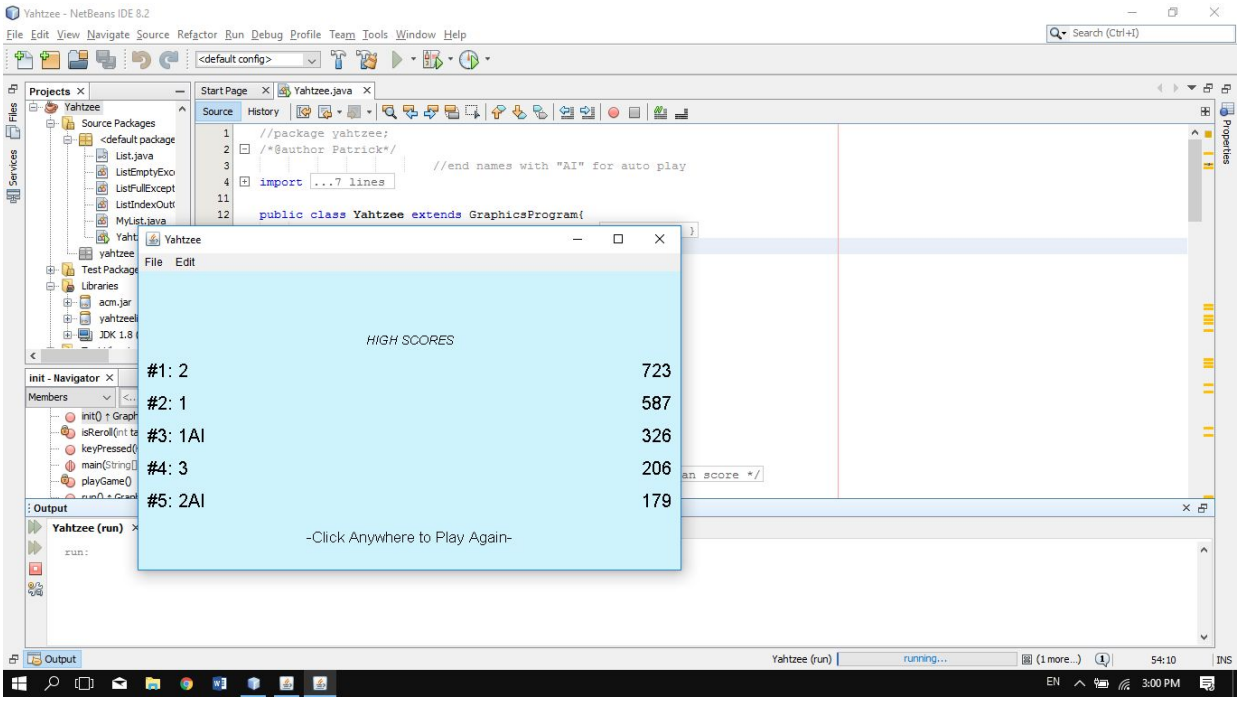
Aside from the red dot indicating the chosen category, a yellow dot is also briefly shown to indicate that the yahtzee bonus has been added to the score in the yahtzee category.



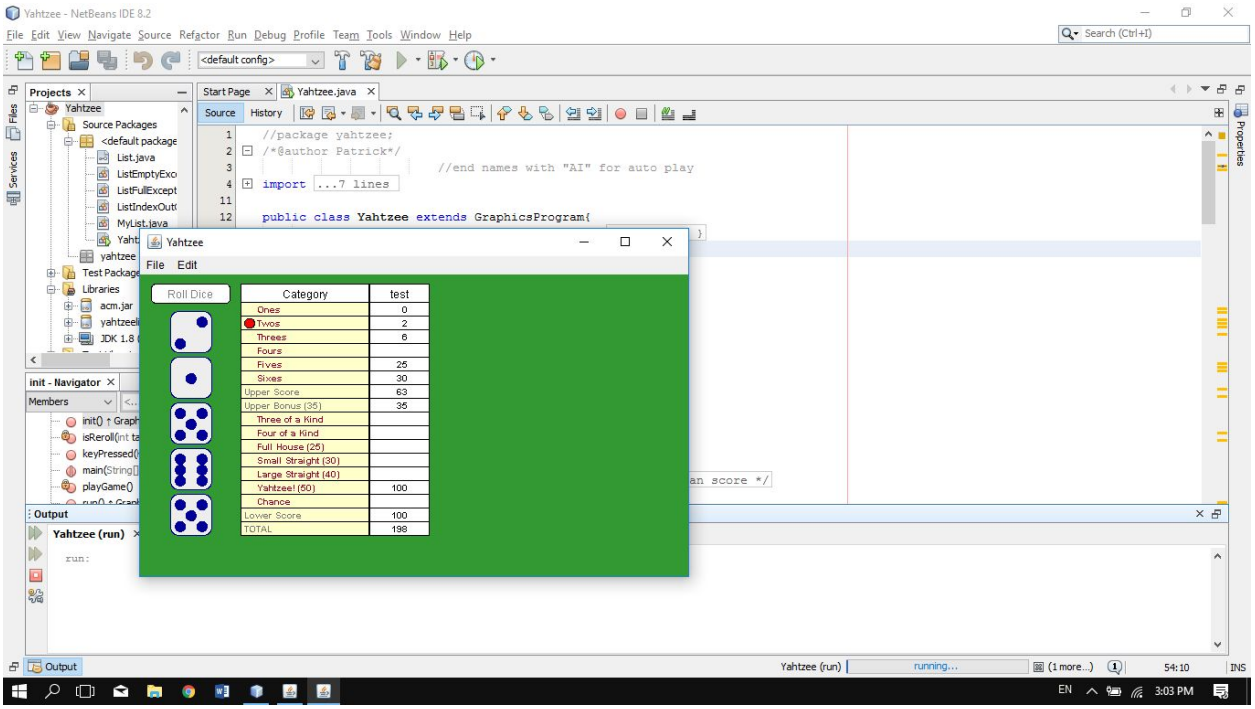
On the other hand, the yahtzee bonus is not awarded when the yahtzee category has already been previously filled with a zero.



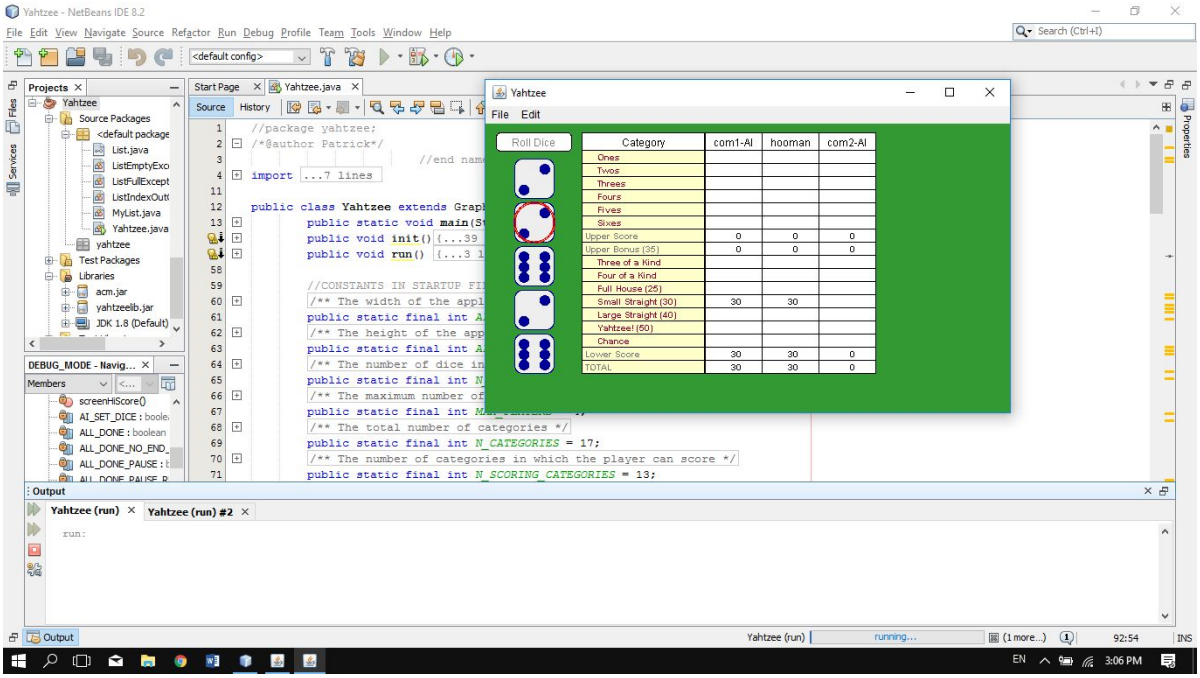
New high scores are given a highlight, and new AI scores get an orange highlight instead of a yellow one.



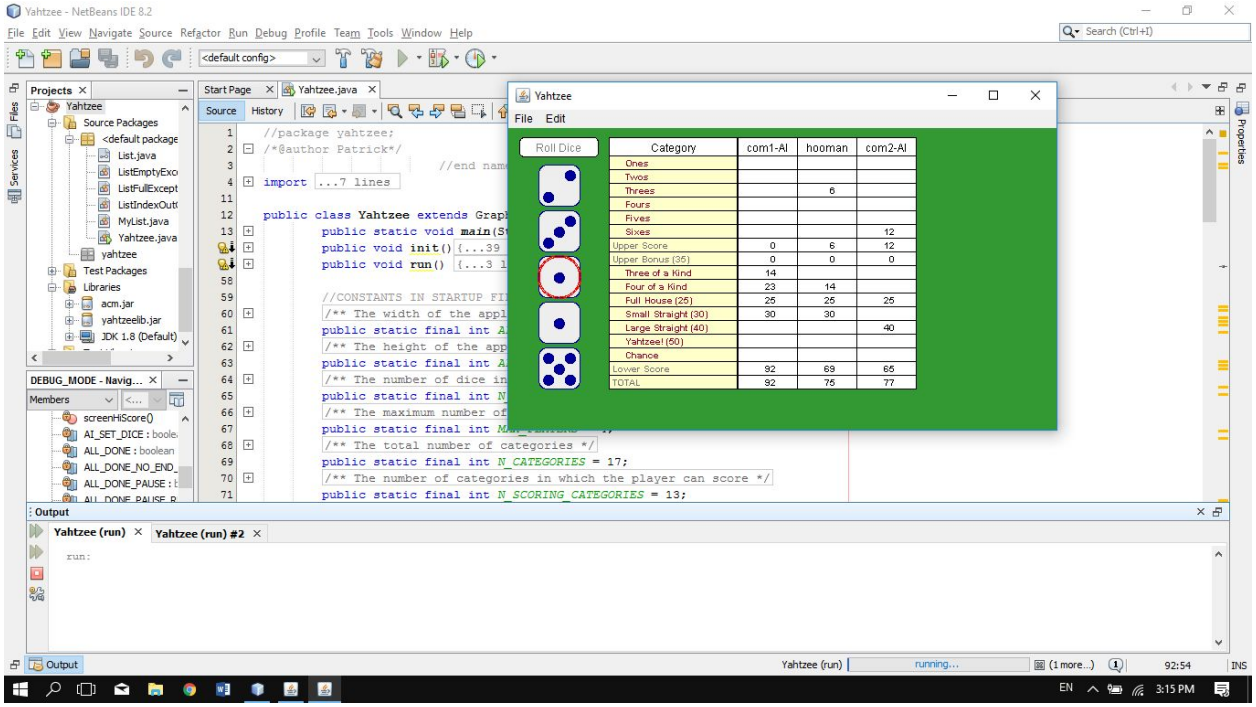
If the score(s) are lower than the existing high scores, the new one is not recorded, and the text “NEW HIGH SCORE!” is not displayed.



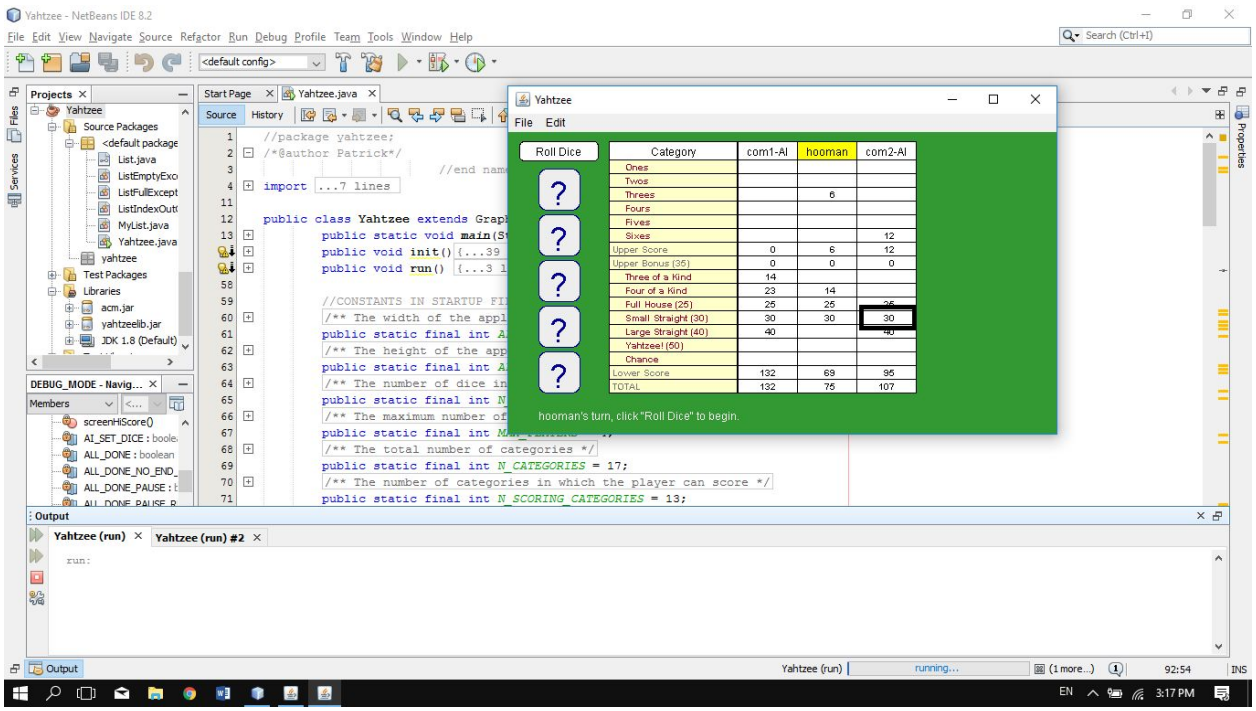
When the player gets a score of 63 or higher in the upper categories, an upper bonus of 35 points is awarded to the player.



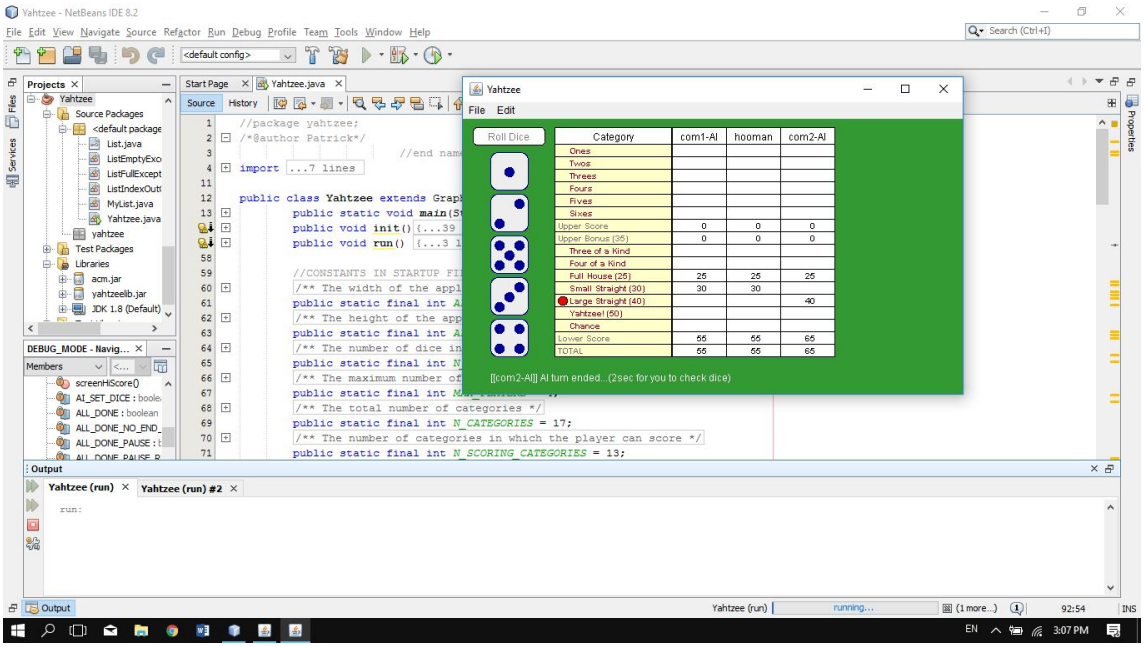
Human vs AI gameplay-- the computer chooses the appropriate dice to reroll, and the dice to reroll are shown with the red circle.



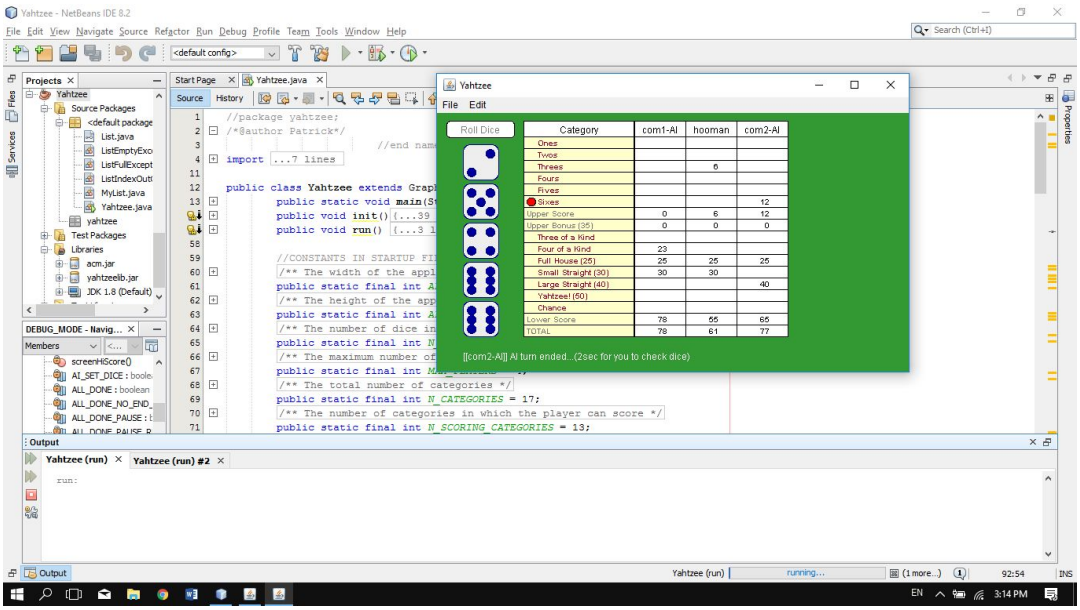
Here, the computer chooses to reroll the “1”, since it targets the “Small Straight” category.



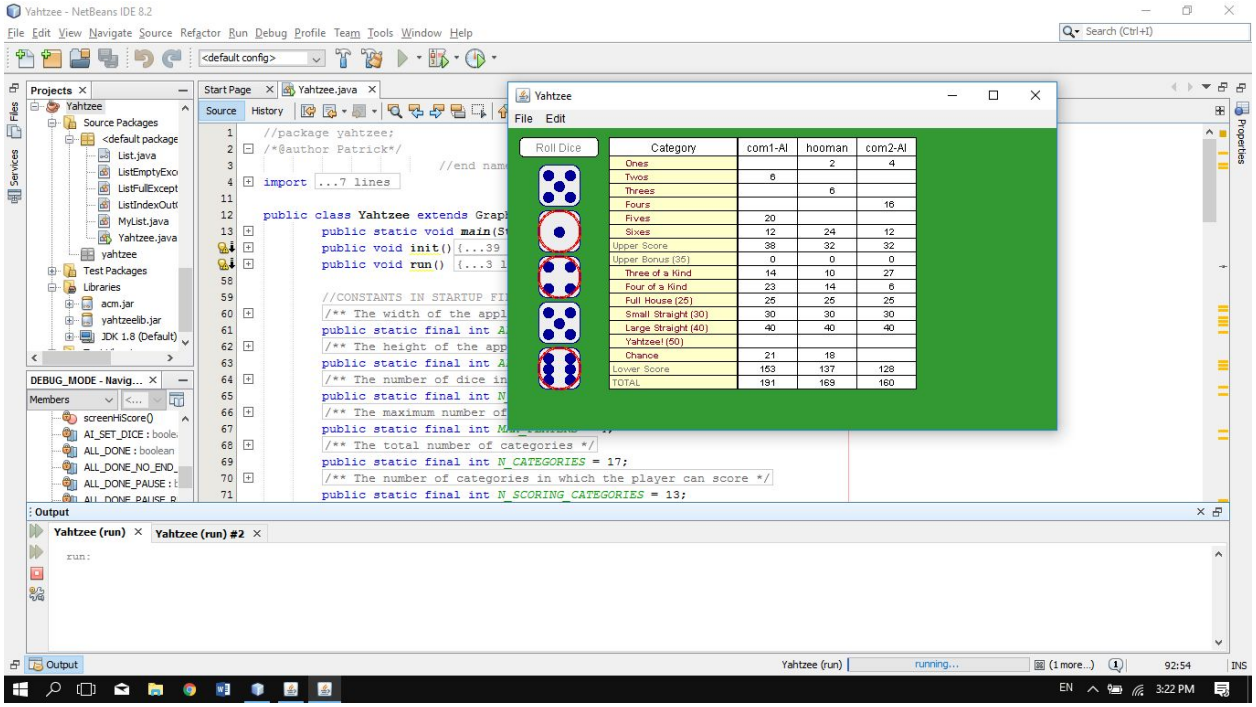
Since the computer(com2--AI) was successful, it chose the “Small Straight” category, and the score was credited.



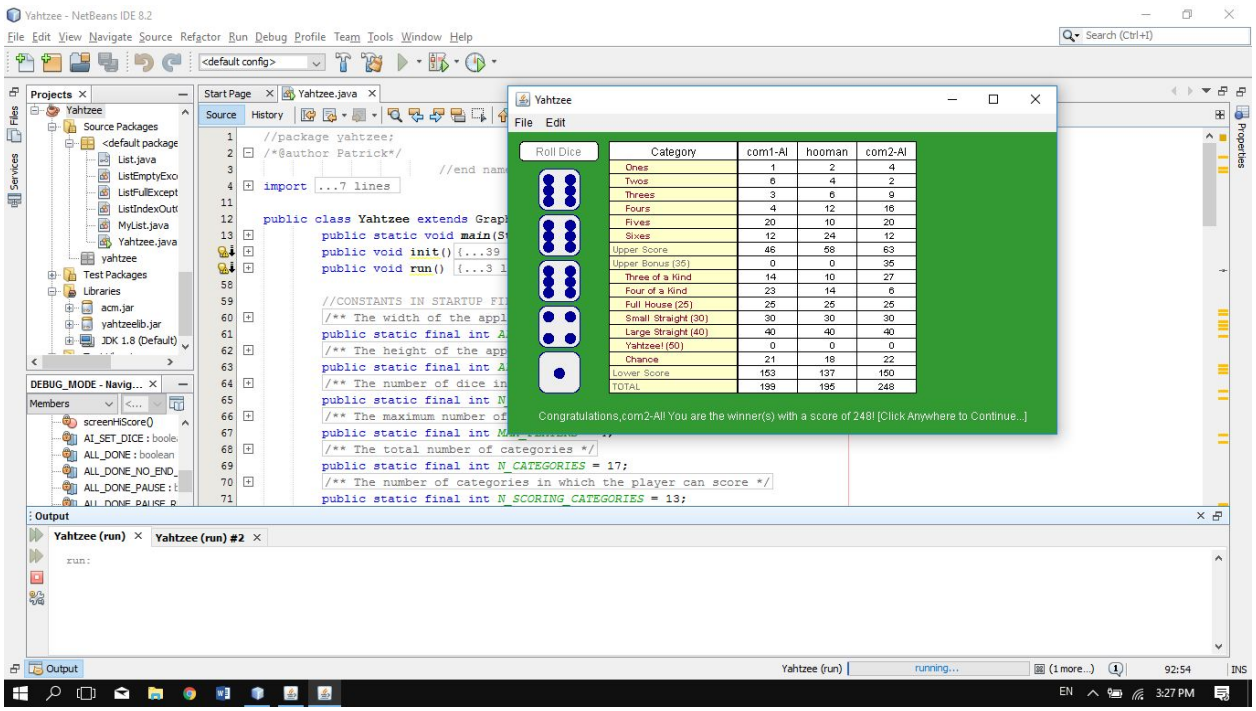
After a computer's turn, the program pauses for 2 seconds in order to show the chosen category, and as an allowance to let the user understand what happened on that specific turn.



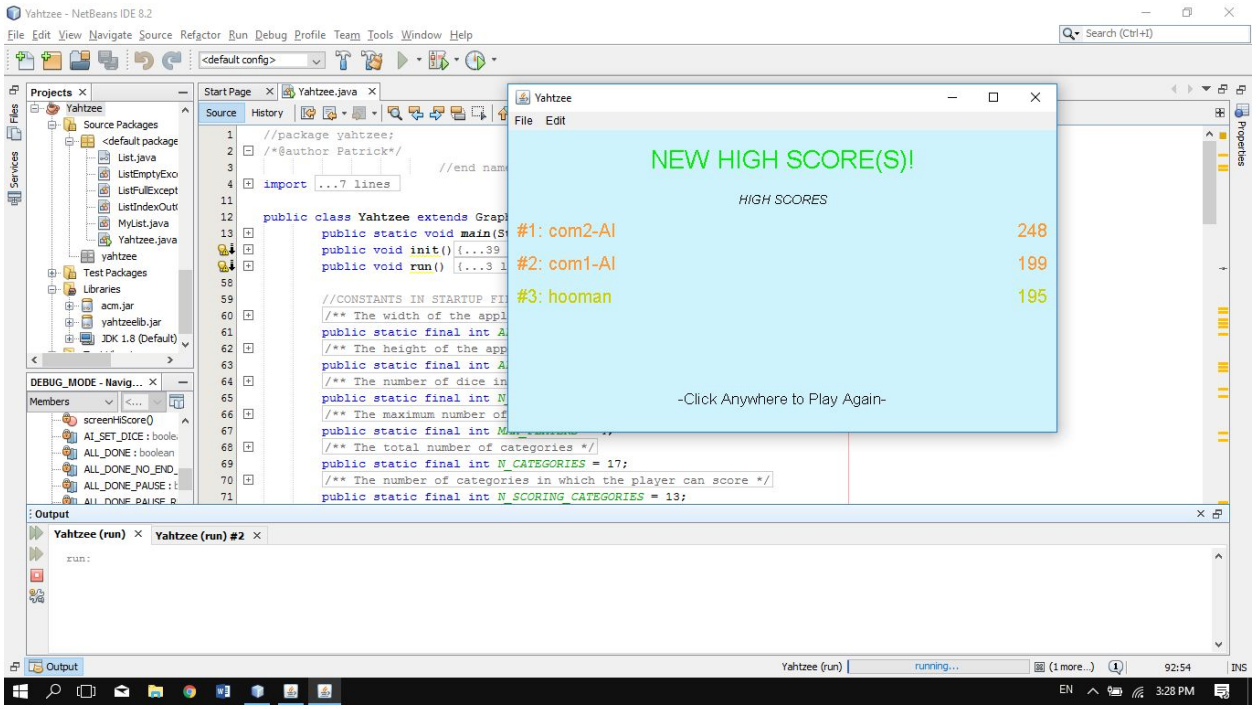
The chooses the best category based on the roll. The computer also only reserves the "Chance" category for really bad rolls. Actually, for that case, if the rerolls have not been used up, the computer would try to aim for a higher score in CHANCE by trying to get rid of the low dice.



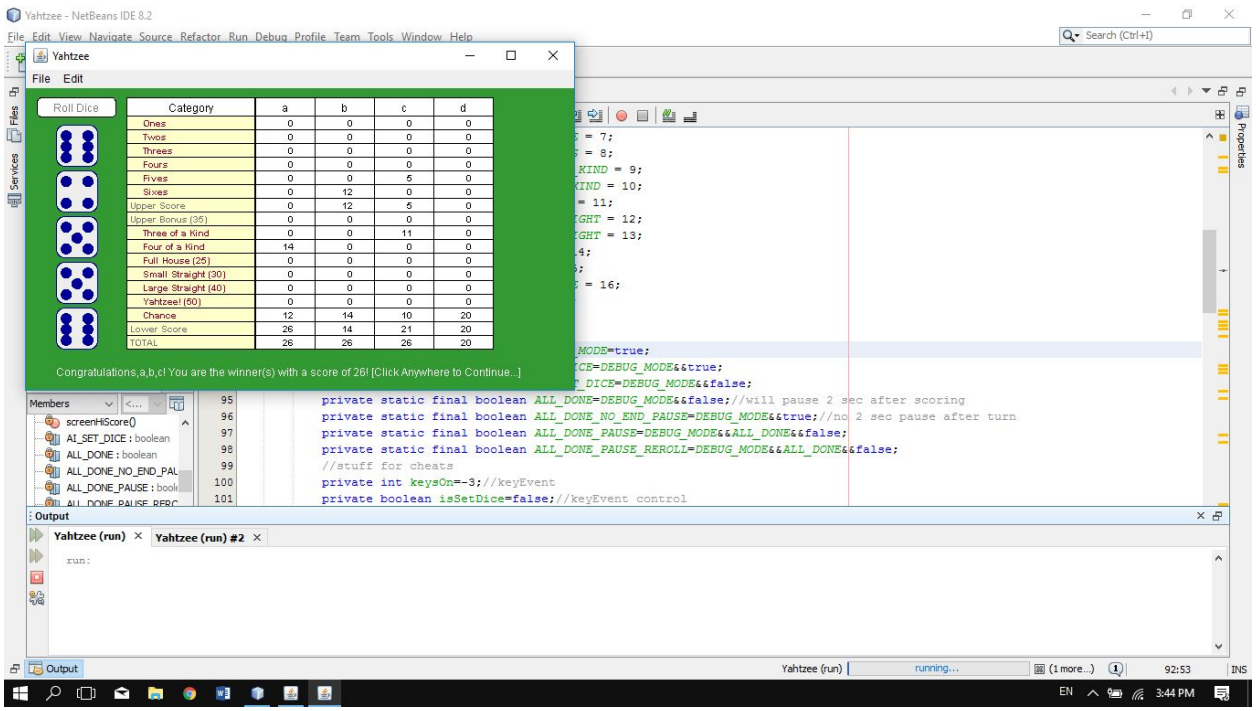
com2--AI here targets the Yahtzee category, and will choose the FIVES if unsuccessful.



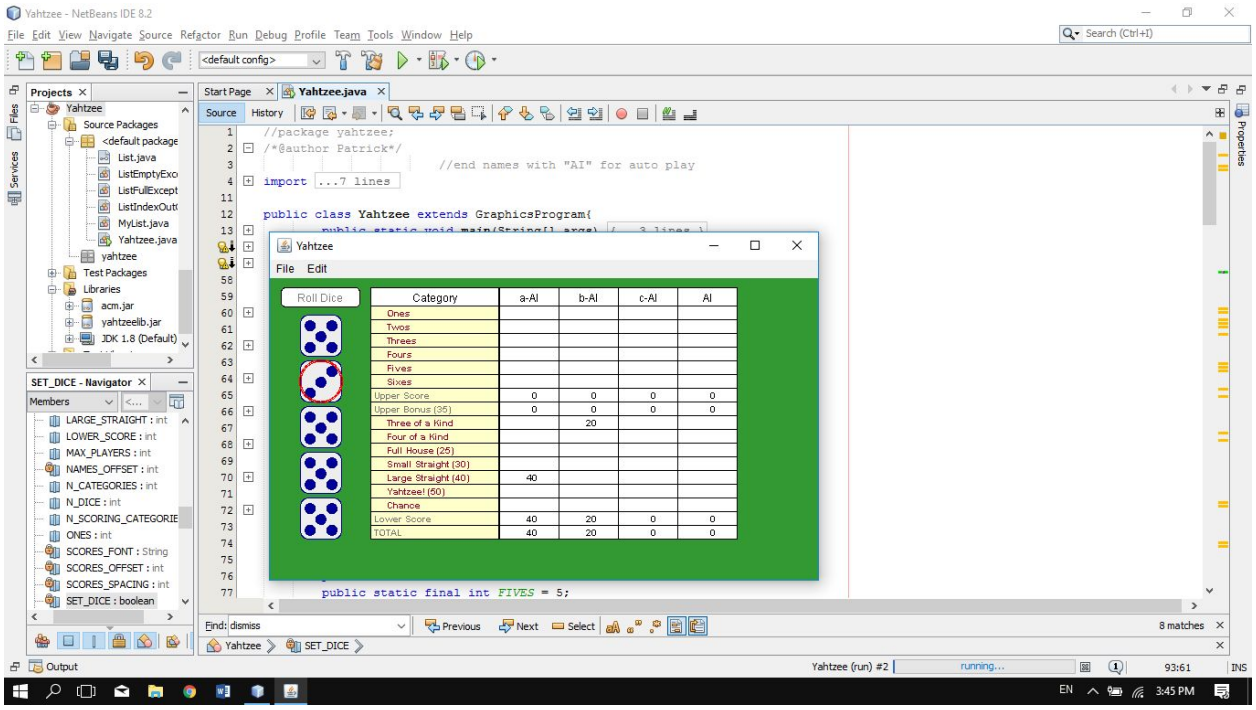
The computer players have beaten the hooman player.



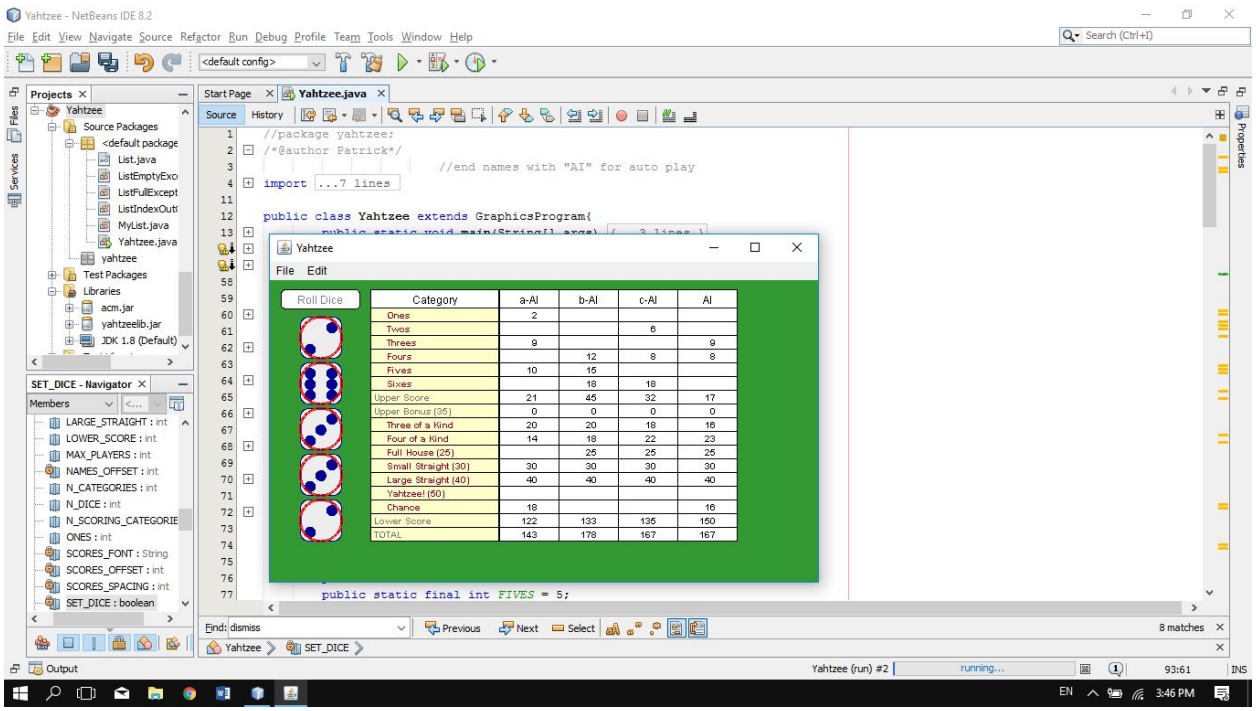
All high scores (human and computer) are recorded. (with different highlight colors).



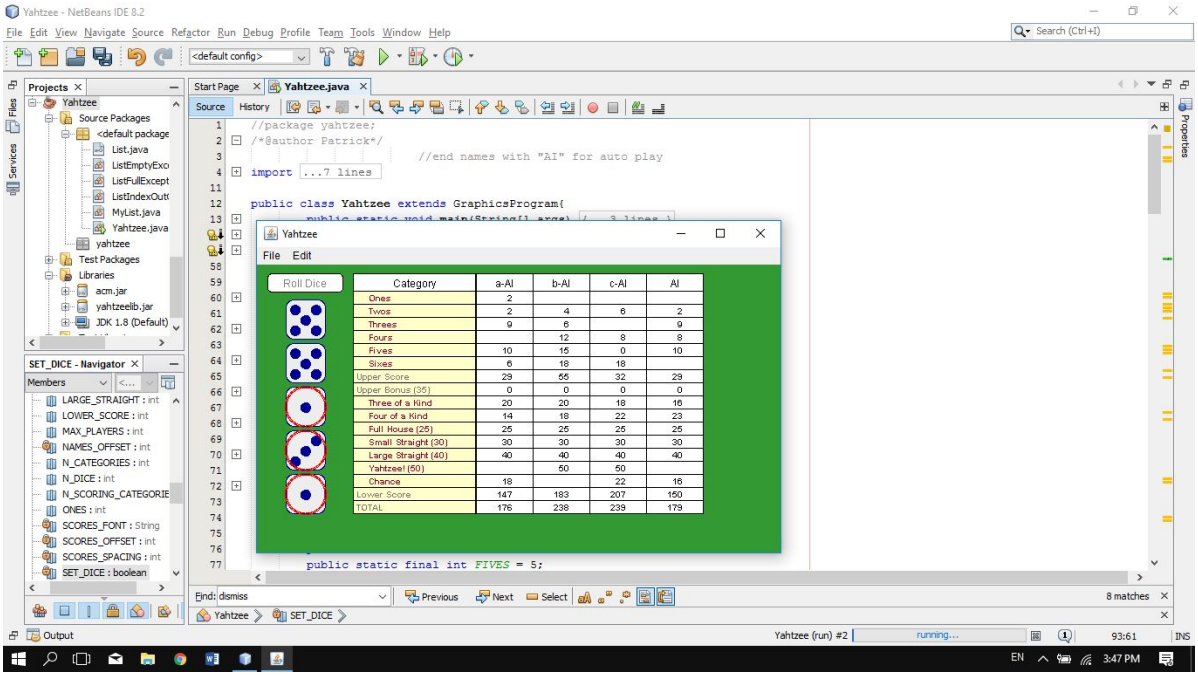
Program also recognizes multiple winners.



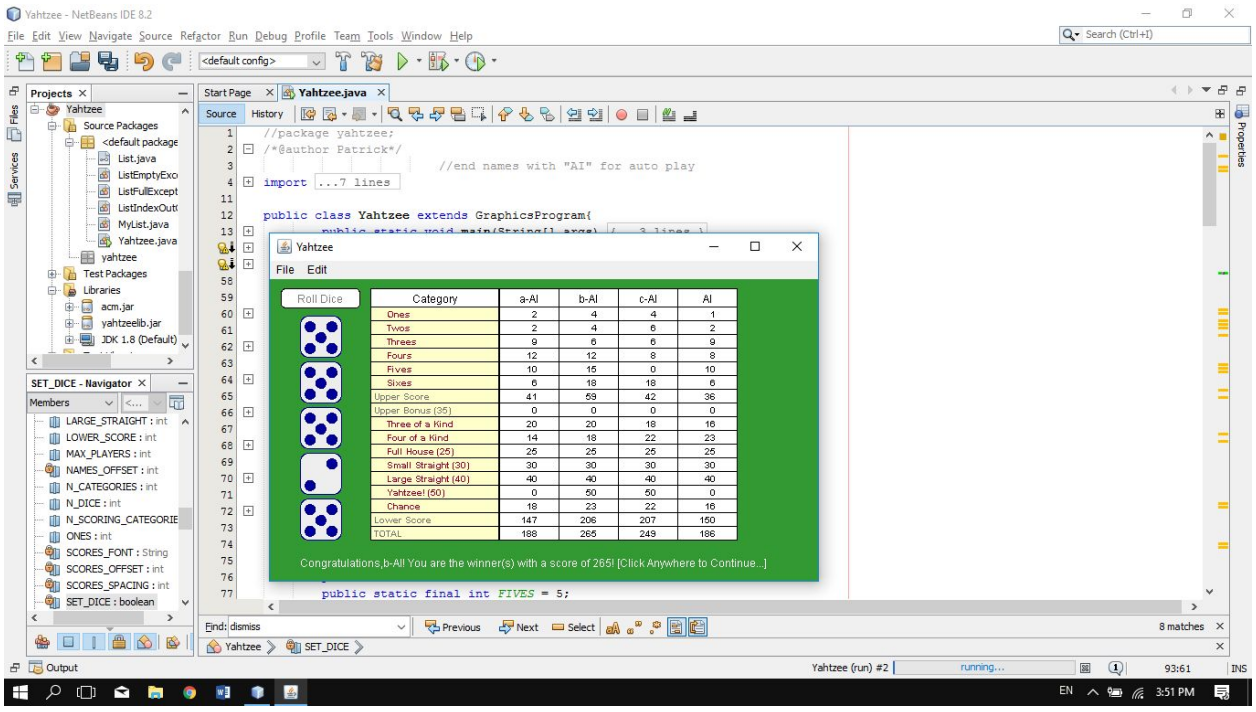
Full computer gameplay.



When Chance is already filled, a really bad roll is fully rerolled.



When Full House is already filled, the computer would retain the higher of 2 pairs and reroll the rest (unlike how it normally only rerolls the unpaired die in hopes of a full house)



The full computer played game, completed.

SUMMARY

In this activity, I have once again reviewed array data manipulation that I have learned from my LBYEC71 class, with the exception that I am now using Java instead of C/C++. Also, similar to the first activity, I was able to understand the methods and use the .jar package included in the starter files.

The activity itself required a lot of array data, and I had to know how to distinguish each one from one another, and retrieve the correct index from each array in every iteration of the loops I've used. On of the major parts of coding this, was the scoring system of the game. In here, given an array of dice, I have to think of methods to check for the validity of the roll, which would vary a lot, based on the chosen category.

For this activity, it was nice that the graphics part of the program was already finished, so it immediately provided a channel for the user to send in data, and for the programmer (me) to be able to check these data by giving the computer a set of instructions to follow, in order to correctly process the user input. After finishing the game, the necessary List ADT usage was also a great review on what I have learned on the previous two activities. In fact, as I realized the usefulness of this ADT, and the convenience that it offers, I also used it as part of the code regarding the computer players, aside from the high score list.

For the computer player extension, it was quite challenging to think of the proper algorithms and implement them, but this proved to be fun. In doing this, I implemented the concept of a histogram, in which I used another array to take note of the frequency of each die value in the roll, and used this data to make the computer reroll the dice that ruin the potential combination. As I was extending the program, there were even times that, as I was trying to convert part of the way I play into a fixed algorithm, I even learned about possible better alternative strategies and further improved the code in choosing which dice to reroll for the computer, and choosing the best category for the computer.

APPENDIX

The Code

```
//package yahtzee;
/*@author Patrick*/
                    //end names with "AI" for auto play
import acm.graphics.*;
import acm.io.*;
import acm.program.*;
import acm.util.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.*;

public class Yahtzee extends GraphicsProgram{
          public static void main(String[] args) {
                    new Yahtzee().start(args);
          }
       public void init(){
          //addMouseListeners();
          GCanvas canvas = getGCanvas();
          canvas.addKeyListener(this);
          names.createList();
          scores.createList();
          //selected Die
          final int arcDieExcess=3;//6-spacing bet dice
          final int arcDiam=44+arcDieExcess;
          for(int i=0;i<N_DICE;i++){
             selectedDie[i]=new GArc(33.0-arcDieExcess/2,40.0-arcDieExcess/2
                   +i*50,arcDiam,arcDiam,270,0);
             selectedDie[i].setColor(Color.RED);
             //GDie is
             //GRoundRect outline = new GRoundRect(0.0D, 0.0D, 44.0D, 44.0D, 18.0D);
             //and
             //this.canvas.add(this.diceArray[i], 33.0D, 40 + i * 50);
          }
          for(int i=N_DICE;i<2*N_DICE;i++){
             selectedDie[i]=new GArc(3.0/2+33.0-arcDieExcess/2,3.0/2+40.0-arcDieExcess/2
                   +(i-N_DICE)*50,arcDiam-3,arcDiam-3,270,0);

             selectedDie[i].setColor(Color.RED);
          }
```

```
    for(int i=0;i<N_CATEGORIES;i++){
       //I choose diameter: 10
       selectedCat[i]=new GOval(110.0+(18.0-10)/2,30.0+(15.0-10)/2+15.0
          *(i),10,10);
       selectedCat[i].setFilled(true);
       selectedCat[i].setFillColor(Color.RED);
       selectedCat[i].setVisible(false);
       //GScorecardEntry is
       //double x = 110.0D;//double y = 10.0D;//y += 20.0D;
       //then
       //entry = new GScorecardEntry(140.0D, 15.0D, c);//y += 15.0D;
       //then alignment of selectables:
       //case 1:  return 18.0D;
    }
}
    public void run() {
    addPlayers();
    }

//CONSTANTS IN STARTUP FILE
/** The width of the application window */
    public static final int APPLICATION_WIDTH = 600;
/** The height of the application window */
    public static final int APPLICATION_HEIGHT = 350;
/** The number of dice in the game */
    public static final int N_DICE = 5;
/** The maximum number of players */
    public static final int MAX_PLAYERS = 4;
/** The total number of categories */
    public static final int N_CATEGORIES = 17;
/** The number of categories in which the player can score */
    public static final int N_SCORING_CATEGORIES = 13;
/** The constants that specify categories on the score sheet */
    public static final int ONES = 1;
    public static final int TWOS = 2;
    public static final int THREES = 3;
    public static final int FOURS = 4;
    public static final int FIVES = 5;
    public static final int SIXES = 6;
    public static final int UPPER_SCORE = 7;
    public static final int UPPER_BONUS = 8;
    public static final int THREE_OF_A_KIND = 9;
    public static final int FOUR_OF_A_KIND = 10;
    public static final int FULL_HOUSE = 11;
    public static final int SMALL_STRAIGHT = 12;
    public static final int LARGE_STRAIGHT = 13;
    public static final int YAHTZEE = 14;
```

```
        public static final int CHANCE = 15;
        public static final int LOWER_SCORE = 16;
        public static final int TOTAL = 17;

    //cheats
    private static final boolean DEBUG_MODE=true;
    private static final boolean SET_DICE=DEBUG_MODE&&true;
    private static final boolean AI_SET_DICE=DEBUG_MODE&&false;
    private static final boolean ALL_DONE=DEBUG_MODE&&false;//will pause 2 sec
after scoring
    private static final boolean ALL_DONE_NO_END_PAUSE=DEBUG_MODE&&true;//no
2 sec pause after turn
    private static final boolean
ALL_DONE_PAUSE=DEBUG_MODE&&ALL_DONE&&false;
    private static final boolean
ALL_DONE_PAUSE_REROLL=DEBUG_MODE&&ALL_DONE&&false;
    //stuff for cheats
    private int keysOn=-3;//keyEvent
    private boolean isSetDice=false;//keyEvent control
    //global variables
    private int nPlayers;
        private String[] pNames;
        private YahtzeeDisplay disp;
        private RandomGenerator rng = RandomGenerator.getInstance();
    private int[] dice=new int[N_DICE];
    private boolean[][] isValid;//keeps track if selected category can be selected
    private int[][] pScores;//player scores
    private MyList<String> names=new MyList<>();
    private MyList<Integer> scores=new MyList<>();
    private MyList<Integer> newHSIndices=new MyList<>();
        ////////////////////////////HI-SCORE SCREEN////////////////////////////////
    private final static int HI_SCORES=5;
    private int newHSCount=0;
    private final static int YAY_OFFSET=7;
    private final static int SCORES_SPACING=10;
    private final static int NAMES_OFFSET=10;//x
    private final static int SCORES_OFFSET=10;//x
    private static final String YAY_FONT = "SansSerif-27";
    private static final String HS_DISP_FONT = "SansSerif-14-ITALIC";
    private static final String SCORES_FONT = "SansSerif-20";
    private static final String CLICK_FONT = "SansSerif-17";
    private static final int CLICK_Y_OFFSET = 7;
    private GLabel yay=new GLabel("NEW HIGH SCORE(S)!");
    private GLabel hiScoreDisp=new GLabel("HIGH SCORES");
    private GLabel HSNames[]=new GLabel[HI_SCORES];
    private GLabel HSScores[]=new GLabel[HI_SCORES];
    private GLabel click;
```

```
    private double ctrX(GObject g){
       return (getWidth()-g.getWidth())/2;
    }
    private double ctrY(GObject g){
       return (getHeight()-g.getHeight())/2;
    }
    private int countYahtzees[];
    //ai select dice
    private GArc selectedDie[]=new GArc[2*N_DICE];
    //ai select cat
    private GOval selectedCat[]=new GOval[N_CATEGORIES];









    //methods
    //error stream
    public class MyException extends RuntimeException{
       public MyException(String s){
          super(s);
       }//end constructor
    }
    //assigns 1-6 randomly to specified die
    private void dieRNG(int dieIndex){
       dice[dieIndex]=rng.nextInt(1,6);
    }
    private void addPlayers(){
       removeAll();
       setBackground(new Color(206,243,253));
       IODialog dialog = getDialog();
       dialog.setAllowCancel(true);
       try{
          nPlayers = dialog.readInt("Enter number of players (Max:
"+MAX_PLAYERS+").");
       } catch (CancelledException e){
          GLabel ex=new GLabel("Dialog Box Cancelled, Program will now EXIT.");
          add(ex);
          ex.setFont(new Font("SansSerif", 3, 25));
          ex.setColor(Color.RED);
          ex.move((getWidth()-ex.getWidth())/2,(getHeight()-ex.getHeight())/2);
          pause(1000);
          exit();
       }
       dialog.setAllowCancel(false);
       while(nPlayers>MAX_PLAYERS || nPlayers<=0){
```

```
            dialog.showErrorMessage("Invalid number of players!");
            nPlayers = dialog.readInt("Enter number of players (Max:
"+MAX_PLAYERS+").");
         }
         pNames = new String[nPlayers];
         isValid = new boolean[nPlayers][N_CATEGORIES];//1st-player index,2nd-cat
index-1 (since 0 based all)
         pScores = new int[nPlayers][N_CATEGORIES];//1st-player index,2nd-cat index-1
(since 0 based all)
         countYahtzees=new int[nPlayers];
         for (int i = 1; i <= nPlayers; i++) {
            //show how to AI
            GLabel ex=new GLabel("Note: End player name(s) with \"AI\" to create
computer player(s).");
            add(ex);
            ex.setFont(new Font("SansSerif", 3, 16));
            ex.setColor(Color.BLACK);
            ex.move((getWidth()-ex.getWidth())/2,(getHeight()-ex.getHeight())/2-100);

            //dialog
            pNames[i - 1] = dialog.readLine("Enter name for player " + i);
            //avoid begin with '\b' (used in hi-scores)
            if(pNames[i-1].length()>0&&pNames[i-1].charAt(0)=='\b'){
               dialog.showErrorMessage("Invalid character!");
               i--;
               continue;
            }
            //initialize isValid
            for(int j=0;j<N_CATEGORIES;j++){
               isValid[i - 1][j]= true;
               pScores[i - 1][j]= 0;
            }
            isValid[i-1][UPPER_SCORE-1]= false;
            isValid[i-1][UPPER_BONUS-1]= false;
            isValid[i-1][LOWER_SCORE-1]= false;
            isValid[i-1][TOTAL-1]= false;
            countYahtzees[i-1]=0;

            //remove ex
            remove(ex);
         }
         disp = new YahtzeeDisplay(getGCanvas(), pNames);
         for(int i=0;i<2*N_DICE;i++){
            add(selectedDie[i]);
         }
         for(int i=0;i<N_CATEGORIES;i++){
            add(selectedCat[i]);
```

```
    }
    playGame();
}
private int calculateScore(int choice){
    int score=0;
    int z=0;//indicator label for scoring cases
    switch(choice){
        case SIXES:
            z++;//6
        case FIVES:
            z++;//5
        case FOURS:
            z++;//4
        case THREES:
            z++;//3
        case TWOS:
            z++;//2
        case ONES:
            z++;//1//just so ONES=1 can be changed
            for(int i=0;i<N_DICE;i++){
                if(dice[i]==z){
                    score+=dice[i];
                }
            }
            z=0;
            break;

        case YAHTZEE:
            z++;//5
        case FOUR_OF_A_KIND:
            z++;//4
        case THREE_OF_A_KIND:
            z=z+3;//3
        {   boolean test=false;
            int count[]=new int[6];//ex: count[1] is the no of 2's
            int sum=0;//scoring for 3,4_of_a_kind
            for(int i=0;i<6;i++){//initialize
                count[i]=0;
            }
            for(int i=0;i<N_DICE;i++){//count
                count[dice[i]-1]++;
                sum+=dice[i];
            }
            for(int i=0;i<6;i++){//check
                if(count[i]>=z){
                    test=true;
                }
```

```
            }
            //credit score
            if(test){
               if(z==5){
                  score=50;
               } else {
                  score=sum;
               }
            }
         }
            z=0;
            break;

         case FULL_HOUSE:{
            boolean test=false;
            int count[]=new int[6];//ex: count[1] is the no of 2's
            for(int i=0;i<6;i++){//initialize
               count[i]=0;
            }
            for(int i=0;i<N_DICE;i++){//count
               count[dice[i]-1]++;
               if(count[dice[i]-1]>3){
                  z=0;
                  break;//check this if glitch
               }
               if(count[dice[i]-1]==3){
                  z=dice[i];//the number w/c is 3 of a kind
               }
            }
            if(z!=0){
               for(int i=0;i<6;i++){//check
                  if(i!=z-1 && count[i]==2){
                     test=true;
                  }
               }
            }
            //credit score
            if(test){
               score=25;
            }
            break;
         }

         case LARGE_STRAIGHT:
            z++;//5 //expected minimum sorted size
         case SMALL_STRAIGHT:
            z=z+4;//4
```

```
            boolean test=false;
            MyList<Integer> sorted = new MyList<>();
            sorted.createList();
            //modified insertion sort (removes equals)
            sorted.add(1,dice[0]);
            for(int i=1;i<N_DICE;i++){//sorted ones
               int j=1;//current intended insertion index in list
               for(;j<=sorted.size();j++){
                  if(dice[i]==sorted.get(j)){
                     j=sorted.size()+100;//terminate loop & ignore value
                  }else if(dice[i]<sorted.get(j)){
                     sorted.add(j,dice[i]);
                     j=sorted.size()+100;//terminate loop & ignore value
                  }
               }
               if(j==sorted.size()+1){//if highest among sorted
                  sorted.add(j,dice[i]);
               }
            }
            if(sorted.size()>=z){
               test=true;//to test
               for(int i=2;i<=z;i++){//check if next item is one more than prev
                  if(sorted.get(i)!=sorted.get(i-1)+1){
                     test=false;
                  }
               }
            }
            //credit score
            if(test){
               score=30+10*(z-4);//40 if z=5 (large straight)
            }
            z=0;
            break;

         case CHANCE:
            for(int i=0;i<N_DICE;i++){//scoring
               score+=dice[i];
            }
            break;

         default:
            throw new MyException("Invalid category: "+choice);
      }
      return score;
   }
```

```
    public void keyPressed(KeyEvent e){//press enter thrice to set dice values
        if(SET_DICE||AI_SET_DICE){
            char a=e.getKeyChar();
            //System.out.printf("isSetDice=%b\nKeylistener input:%c,
ASCII#%d\n\n",isSetDice,a,(int)a);
            if(!isSetDice){
                keysOn=-3;
            }
            if(keysOn<0){
                if(a=='\n'||a=='\r'){//3 consecutive enters turn it on
                    keysOn++;
                    if (keysOn==0){
                        disp.printMessage("CHEAT MODE: Enter value (1~6) for dice
#"+(keysOn+1)+".");
                    }
                }
            } else if (keysOn<N_DICE){//keysOn is the dice index manipulated
                if(a>='1'&&a<='6'){
                    dice[keysOn]=a-'0';
                    keysOn++;
                    disp.displayDice(dice);
                    //ask for next die (as keysOn has already been incremented)
                    if(keysOn<N_DICE){
                        disp.printMessage("CHEAT MODE: Enter value (1~6) for dice
#"+(keysOn+1)+".");
                    } else {
                        disp.printMessage("CHEAT complete, proceed with the game.");
                    }
                } else {
                    disp.printMessage("CHEAT: dice#"+(keysOn+1)+", invalid input: "+a);
                }
            } else {
                keysOn=-3;
            }
        }
    }



    //show selected dice
    private void dieSelectShow(int i){
        selectedDie[i].setVisible(true);
        selectedDie[i+N_DICE].setVisible(true);
        for(int j=4;j%360!=0;j=j+4){
            selectedDie[i].setSweepAngle(j);
            selectedDie[i+N_DICE].setSweepAngle(j);
            pause(1);
```

```
    }
  }
  private void dieSelectDismiss(){
    for(int i=0; i<2*N_DICE;i++){
      selectedDie[i].setSweepAngle(0);
    }
  }
  //show selected cat    (1-based)
  private void catSelectShow(int i){
    selectedCat[i-1].setFillColor(Color.RED);
    selectedCat[i-1].setVisible(true);
  }
  private void catSelectShow(int i,boolean isRepeatYahtzee){
    selectedCat[i-1].setFillColor(Color.RED);
    if(isRepeatYahtzee){
      selectedCat[i-1].setFillColor(Color.YELLOW);
    }
    selectedCat[i-1].setVisible(true);
  }
  private void catSelectDismiss(){
    for(int i=0; i<N_CATEGORIES;i++){
      selectedCat[i].setVisible(false);
    }
  }




  ////////////////////////////////////////////////
  //++AI
  //reroll decision basis
  private int[] allDiceArr(){
    int[] a=new int[N_DICE];
    for(int i=0;i<N_DICE;i++){
      a[i]=i;
    }
    return a;
  }
  ////////////////////////////////////THE BRAIN!/////////////////////////////////////
  private int[] isReroll(int targetCat,int player){//dice to reroll
    MyList<Integer> rerolls=new MyList<>();//reroll indices
    rerolls.createList();


    if(targetCat<1 || targetCat>=CHANCE){//chance is the last box
      double ave=0;
```

```
            for(int i=0;i<N_DICE;i++){
               ave=ave+dice[i];
            }
            ave=ave/N_DICE;
            //reroll all lower than ave
            for(int i=0;i<N_DICE;i++){
               if(dice[i]<ave){
                  rerolls.add(1,i);
               }
            }
         } else {
            int hi[]=new int[6];//histogram, 0-based
            for(int i=0;i<6;i++){
               hi[i]=0;
            }
            for(int i=0;i<N_DICE;i++){
               hi[dice[i]-1]++;
            }
            int maxRep=-1;//repetitions (overall)
            int maxRepValid=-1;//repetitions (valids only)
            int maxDieVal=-1;//1-based(overall)
            int maxDieValValid=-1;//1-based(valids)
            int tempos=0;//if 2 pairs
            for(int i=0;i<6;i++){//findmax
               if(isValid[player-1][(i+1)-1]){//if not occupied
                  if(hi[i]>=maxRepValid){
                     maxRepValid=hi[i];
                     maxDieValValid=i+1;
                     if(hi[i]==2){
                        tempos++;
                     }
                  }
               } else {//invalids
                  if(hi[i]>=maxRep){
                     maxRep=hi[i];
                     maxDieVal=i+1;
                     if(hi[i]==2){
                        tempos++;
                     }
                  }
               }
            }
            if(maxRepValid>maxRep){
               maxRep=maxRepValid;
               maxDieVal=maxDieValValid;
            }
            if(maxRep!=2){
```

```
            tempos=-1;
          }
          if(ALL_DONE_PAUSE_REROLL || AI_SET_DICE){

System.err.println(tempos+"."+maxRep+".."+hi[0]+"."+hi[1]+"."+hi[2]+"."+hi[3]+"."+hi[4]
+"."+hi[5]);}


          //shd not be 2 pairs, it goes to after full house
          if(tempos!=2 && maxRep<=2 && (isValid[player-1][LARGE_STRAIGHT-1] ||
isValid[player-1][SMALL_STRAIGHT-1])){
            //go for straight
            if(calculateScore(LARGE_STRAIGHT)>0){
              ;//stop
            } else {//reroll copies
              int[] t=hi;
              for(int i=0;i<N_DICE;i++){
                if(t[dice[i]-1]>=2){
                  rerolls.add(1,i);
                  t[dice[i]-1]--;
                } else if (dice[i]==1 && (hi[2-1]==0 || hi[3-1]==0)){
                  rerolls.add(1,i);
                  t[dice[i]-1]--;
                } else if (dice[i]==6 && (hi[5-1]==0 || hi[4-1]==0)){
                  rerolls.add(1,i);
                  t[dice[i]-1]--;
                }
              }
            }
          } else if(maxRep==3 && isValid[player-1][FULL_HOUSE-1]){
            //go for full house
            if(calculateScore(FULL_HOUSE)>0){
              ;//stop
            } else {
              for(int i=0;i<N_DICE;i++){
                if(hi[dice[i]-1]!=3){
                  rerolls.add(1,i);
                }
              }//reroll the 2
            }
          } else{
            //of a kinds

            //reroll different than higher histograms
            if(maxRep>=2){
              //chk if sp case:2 pairs
              boolean is2Pairs=false;
```

```
            if(maxRep==2){
               int aa=0;//increment if hi[]==2
               int temp=-1;//maxdievalue--highest
               for(int i=0;i<6;i++){
                  if(hi[i]==2){
                     temp=i+1;
                     aa++;
                  }
               }
               if(aa==2){//2 pairs
                  is2Pairs=true;
                  maxDieVal=temp;//set maxDieVal to be the highest
               }
            }


            //2 pairs
            if(is2Pairs && isValid[player-1][FULL_HOUSE-1]){
               for(int i=0;i<N_DICE;i++){
                  if(hi[dice[i]-1]==1){
                     rerolls.add(1,i);
                  }
               }
            } else {
               //determining the max histograms to look at
               if(targetCat<9){//invalids pointless
                  maxRep=maxRepValid;
                  maxDieVal=maxDieValValid;
               }

               for(int i=0;i<N_DICE;i++){
                  if(dice[i]!=maxDieVal){
                     rerolls.add(1,i);
                  }
               }
            }

         } else {//chance is better (no same, cant straight)
            //go for chance
            if(isValid[player-1][CHANCE-1]){
               double ave=0;
               for(int i=0;i<N_DICE;i++){
                  ave=ave+dice[i];
               }
               ave=ave/N_DICE;
               //reroll all lower than ave
               for(int i=0;i<N_DICE;i++){
```

```
                        if(dice[i]<ave){
                            rerolls.add(1,i);
                        }
                    }
                } else {//just reroll all
                    return allDiceArr();
                }
            }
        }
        if(rerolls.isEmpty()){
            return new int[]{};
        } else {
            int[] a=new int[rerolls.size()];
            for(int i=0;i<a.length;i++){
                a[i]=rerolls.get(1);
                rerolls.remove(1);
            }
            return a;
        }
    }
    ////////////////////////////////////////////////
    private void playGame() throws MyException{
        //display totals as 0
        for(int player=1;player<=nPlayers;player++){
            disp.updateScorecard(UPPER_SCORE, player,
pScores[player-1][UPPER_SCORE-1]);
            disp.updateScorecard(UPPER_BONUS, player,
pScores[player-1][UPPER_BONUS-1]);
            disp.updateScorecard(LOWER_SCORE, player,
pScores[player-1][LOWER_SCORE-1]);
            disp.updateScorecard(TOTAL, player, pScores[player-1][TOTAL-1]);
        }
        //game loop
        for(int cat=1;cat<=N_SCORING_CATEGORIES;cat++){
            for(int player=1;player<=nPlayers;player++){//player is 1-based in
YahtzeeDisplay class

                ////////////////////////////// A I //////////////////////////////
                if(ALL_DONE || pNames[player-1].endsWith("AI")){
                    int targetCat=9;//9
                    while(!isValid[player-1][targetCat-1] && targetCat<=YAHTZEE){
                        targetCat++;
                    }
                    if(targetCat==YAHTZEE+1){
                        targetCat=6;
                        while(targetCat>=ONES && !isValid[player-1][targetCat-1]){
```

```
                        targetCat--;
                    }
                    if(targetCat==ONES-1){
                        targetCat=-1;//must be less than 9(3 of a kind)
                    }
                }



                //1st roll
                disp.printMessage(pNames[player-1]+"'s turn, click \"Roll Dice\" to
begin.");
                for(int i=0;i<N_DICE;i++){
                    dieRNG(i);
                }
                if(ALL_DONE_PAUSE_REROLL){pause(5000);}
                else
if(AI_SET_DICE){isSetDice=true;disp.waitForPlayerToSelectDice();isSetDice=false;keysO
n=-3;}
                disp.displayDice(dice);
                //2nd roll
                disp.printMessage("To reroll (2 left), select dice (may choose none) to "
                    + "reroll and click \"Roll Dice\" to proceed.");
                int[] testArr=isReroll(targetCat,player);
                if(testArr.length>0){if(ALL_DONE_PAUSE){waitForClick();}
                    for(int i=0;i<testArr.length;i++){
                        dieRNG(testArr[i]);
                        dieSelectShow(testArr[i]);
                    }
                    if(ALL_DONE_PAUSE_REROLL){pause(5000);}
                    else
if(AI_SET_DICE){isSetDice=true;disp.waitForPlayerToSelectDice();isSetDice=false;keysO
n=-3;}
                    disp.displayDice(dice);
                    dieSelectDismiss();

                    //3rd roll [dapat nested]
                    disp.printMessage("To reroll (1 left), select dice (may choose none) to "
                        + "reroll and click \"Roll Dice\" to proceed.");
                    testArr=isReroll(targetCat,player);
                    if(testArr.length>0){if(ALL_DONE_PAUSE){waitForClick();}
                        for(int i=0;i<testArr.length;i++){
                            dieRNG(testArr[i]);
                            dieSelectShow(testArr[i]);
                        }
                        if(ALL_DONE_PAUSE_REROLL){pause(5000);}
                        else
```

```
if(AI_SET_DICE){isSetDice=true;disp.waitForPlayerToSelectDice();isSetDice=false;keysO
n=-3;}
                        disp.displayDice(dice);
                        dieSelectDismiss();
                    }
                }

                //check possible scores AI (CHANCE is special)
                int maxscore=-1;
                int maxcat=1;
                for(int i=1;i<=YAHTZEE;i++){//chance removed
                    if(i!=7&&i!=8){
                        int scoore=calculateScore(i);
                        //System.out.println("maxscore "+maxscore+",contender:"+scoore);
                        if(scoore>=maxscore && isValid[player-1][i-1]){//highers take priority
                            //SAVE YAHTZEE FOR last if di makuha
                            if(i==YAHTZEE && scoore==0 && isValid[player-1][maxcat-1]){
                                ;//do nothing
                            } else {
                                maxscore=scoore;
                                maxcat=i;
                            }
                            //System.out.println("contender chosen:"+i);
                        }
                    }
                }
                if(maxscore==0 && maxcat==YAHTZEE && isValid[player-1][CHANCE-1]){
                    maxcat=CHANCE;
                } else if (maxcat<=SIXES){
                    int hi[]=new int[6];//histogram, 0-based
                    for(int i=0;i<6;i++){
                        hi[i]=0;
                    }
                    for(int i=0;i<N_DICE;i++){
                        hi[dice[i]-1]++;
                    }
                    int maxRep=-1;//repetitions
                    int maxDieVal=-1;//1-based
                    for(int i=0;i<6;i++){//findmax
                        if(isValid[player-1][(i+1)-1]){//if not occupied
                            if(hi[i]>maxRep){
                                maxRep=hi[i];
                                maxDieVal=i+1;
                            }
                        }
                    }
                    if(maxRep>=2){
```

```
                        maxcat=maxDieVal;
                    } else if(calculateScore(CHANCE)<maxscore) {//chance is lower
                        ;//RETAIN MAXCAT
                    } else if(!isValid[player-1][CHANCE-1]){//chance is occupied
                        ;//RETAIN MAXCAT
                    } else {//chance is better
                        maxcat=CHANCE;
                    }
                }
                //System.out.println("player:"+pNames[player-1]+"--endTurn----------");
                int choice=0;
                choice=maxcat;//method is 1 based
                while(!isValid[player-1][choice-1]){
                    disp.printMessage("Invalid Category! Please choose a BLANK "
                            + "category for scoring to end turn.");
                    System.err.println("player "+player+" AI-invalid choice:"+choice);
                    disp.waitForPlayerToSelectCategory();//anti mouse released glitch
                    if(!isValid[player-1][choice-1]){
                        choice=(choice+1)%N_CATEGORIES;
                    }
                }
                isSetDice=false;//cheat control
                keysOn=-3;//cheat control

                //yahtzee addtl chips
                if(calculateScore(YAHTZEE)==50){
                    countYahtzees[player-1]++;
                }
                if(countYahtzees[player-1]>1 && (pScores[player-1][YAHTZEE-1]!=0 ||
isValid[player-1][YAHTZEE-1])){
                    disp.printMessage("(ANOTHER YAHTZEE! BONUS! (+100)");
                    countYahtzees[player-1]--;
                    pause(500);
                    pScores[player-1][YAHTZEE-1]+=100;
                    disp.updateScorecard(YAHTZEE, player,
pScores[player-1][YAHTZEE-1]);
                    pScores[player-1][TOTAL-1]+=100;//increment to toal score
                    disp.updateScorecard(TOTAL, player, pScores[player-1][TOTAL-1]);
                    pScores[player-1][LOWER_SCORE-1]+=100;//increment to lower score
                    disp.updateScorecard(LOWER_SCORE, player,
pScores[player-1][LOWER_SCORE-1]);
                    catSelectShow(YAHTZEE,true);
                }

                //calculate score
                isValid[player-1][choice-1]=false;
                int score=calculateScore(choice);
```

```
                    //update scorecard
                    pScores[player-1][choice-1]+=score;
                    disp.updateScorecard(choice, player, pScores[player-1][choice-1]);
                    catSelectShow(choice);

                    //pmc--I want to update all scores everytime, so players see real standing
                    if(choice<UPPER_SCORE){
                       pScores[player-1][UPPER_SCORE-1]+=score;
                       disp.updateScorecard(UPPER_SCORE, player,
pScores[player-1][UPPER_SCORE-1]);
                       if(pScores[player-1][UPPER_BONUS-1]!=35 &&
pScores[player-1][UPPER_SCORE-1]>=63){
                          pScores[player-1][UPPER_BONUS-1]=35;
                          pScores[player-1][TOTAL-1]+=35;
                          disp.updateScorecard(UPPER_BONUS, player,
pScores[player-1][UPPER_BONUS-1]);
                       }
                    } else {
                       pScores[player-1][LOWER_SCORE-1]+=score;
                       disp.updateScorecard(LOWER_SCORE, player,
pScores[player-1][LOWER_SCORE-1]);
                    }
                    pScores[player-1][TOTAL-1]+=score;//increment to toal score
                    disp.updateScorecard(TOTAL, player, pScores[player-1][TOTAL-1]);

                    if(ALL_DONE_PAUSE){
                       waitForClick();
                    } else if(!ALL_DONE_NO_END_PAUSE){
                       disp.printMessage("[["+pNames[player-1]+"]] AI turn ended"
                             + "...(2sec for you to check dice)");
                       pause(2000);
                    }
                    catSelectDismiss();
                    continue;
                 }
                 ////////////////////////////////// A I    END////////////////////////////////////



                 //first roll
                 disp.printMessage(pNames[player-1]+"'s turn, click \"Roll Dice\" to begin.");
                 disp.waitForPlayerToClickRoll(player);
                 for(int i=0;i<N_DICE;i++){
                    dieRNG(i);
                 }
                 disp.displayDice(dice);
                 //reroll
```

```
isSetDice=true;//cheat control
disp.printMessage("To reroll (2 left), select dice (may choose none) to "
      + "reroll and click \"Roll Dice\" to proceed.");
disp.waitForPlayerToSelectDice();
isSetDice=false;//cheat control
keysOn=-3;//cheat control
for(int i=0;i<N_DICE;i++){
   if(disp.isDieSelected(i)){
      dieRNG(i);
      dieSelectShow(i);
   }
}
disp.displayDice(dice);
dieSelectDismiss();
//reroll (the same thing)
isSetDice=true;//cheat control
disp.printMessage("To reroll (1 left), select dice (you may choose "
      + "none) to reroll and click \"Roll Dice\" to proceed.");
disp.waitForPlayerToSelectDice();
isSetDice=false;//cheat control
keysOn=-3;//cheat control
for(int i=0;i<N_DICE;i++){
   if(disp.isDieSelected(i)){
      dieRNG(i);
      dieSelectShow(i);
   }
}
disp.displayDice(dice);
dieSelectDismiss();


//categories
//category validity
isSetDice=true;//cheat control
disp.printMessage("Out of rerolls! Please choose a blank category"
      + " for scoring to end turn.");
int choice=0;
choice=disp.waitForPlayerToSelectCategory();//method is 1 based
while(!isValid[player-1][choice-1]){
   disp.printMessage("Invalid Category! Please choose a BLANK "
         + "category for scoring to end turn.");
   choice=disp.waitForPlayerToSelectCategory();
}
isSetDice=false;//cheat control
keysOn=-3;//cheat control

//yahtzee addtl chips
```

```
                if(calculateScore(YAHTZEE)==50){
                   countYahtzees[player-1]++;
                }
                if(countYahtzees[player-1]>1 && (pScores[player-1][YAHTZEE-1]!=0 ||
isValid[player-1][YAHTZEE-1])){
                   disp.printMessage("(ANOTHER YAHTZEE! BONUS! (+100)");
                   countYahtzees[player-1]--;
                   pause(500);
                   pScores[player-1][YAHTZEE-1]+=100;
                   disp.updateScorecard(YAHTZEE, player, pScores[player-1][YAHTZEE-1]);
                   pScores[player-1][TOTAL-1]+=100;//increment to toal score
                   disp.updateScorecard(TOTAL, player, pScores[player-1][TOTAL-1]);
                   pScores[player-1][LOWER_SCORE-1]+=100;//increment to lower score
                   disp.updateScorecard(LOWER_SCORE, player,
pScores[player-1][LOWER_SCORE-1]);
                   catSelectShow(YAHTZEE,true);
                }

                //calculate score
                isValid[player-1][choice-1]=false;
                int score=calculateScore(choice);
                //update scorecard
                pScores[player-1][choice-1]+=score;
                disp.updateScorecard(choice, player, pScores[player-1][choice-1]);
                catSelectShow(choice);

                //pmc--I want to update all scores everytime, so players see real standing
                if(choice<UPPER_SCORE){
                   pScores[player-1][UPPER_SCORE-1]+=score;
                   disp.updateScorecard(UPPER_SCORE, player,
pScores[player-1][UPPER_SCORE-1]);
                   if(pScores[player-1][UPPER_BONUS-1]!=35 &&
pScores[player-1][UPPER_SCORE-1]>=63){
                      pScores[player-1][UPPER_BONUS-1]=35;
                      pScores[player-1][TOTAL-1]+=35;
                      disp.updateScorecard(UPPER_BONUS, player,
pScores[player-1][UPPER_BONUS-1]);
                   }
                } else {
                   pScores[player-1][LOWER_SCORE-1]+=score;
                   disp.updateScorecard(LOWER_SCORE, player,
pScores[player-1][LOWER_SCORE-1]);
                }
                pScores[player-1][TOTAL-1]+=score;//increment to toal score
                disp.updateScorecard(TOTAL, player, pScores[player-1][TOTAL-1]);

                pause(100);
```

```
                catSelectDismiss();
            }
        }
        MyList<Integer> winners=new MyList<>();//winner player indices (for ties)
        int winner=0;//NOTE: Indices are 0-based
        winners.createList();
        winners.add(1,winner);
        for(int player=1;player<nPlayers;player++){//actual player index (0 based)
            if(pScores[player][TOTAL-1]>pScores[winner][TOTAL-1]){
                winner=player;
                winners.createList();//reset
                winners.add(1,winner);
            } else if (pScores[player][TOTAL-1]==pScores[winner][TOTAL-1]){
                winners.add(winners.size()+1,player);
            }
        }
        String winnerString=new String("");
        for(int i=1;i<=winners.size();i++){
            winnerString=winnerString.concat(","+pNames[winners.get(i)]);
        }//System.err.println(winnerString);//debug
        disp.printMessage("Congratulations"+winnerString+"! You are the winner(s)
with a"
                + " score of "+pScores[winner][TOTAL-1]+"! [Click Anywhere to
Continue...]");
        waitForClick();
        screenHiScore();
        }




    ////////////////////////////HI-SCORE SCREEN////////////////////////////////
    private void screenHiScore() throws HiScoreListSyncException{
        for(int i=0;i!=nPlayers;i++){
            //System.out.println(pNames[i]+"--"+pScores[i][TOTAL-1]);
        }

        newHSIndices.createList();
        setBackground(new Color(206,243,253));
        removeAll();
        //check integrity
        if(names.size()!=scores.size()){
            throw new HiScoreListSyncException("ERROR: HiScore List Size Mismatch.");
        } else if (names.size()>HI_SCORES){
            throw new HiScoreListSyncException("ERROR: HiScore List Size Exceeds
```

```
"+HI_SCORES+".");
        }
        //input scores
        //int newScorers=0;
        for(int player=1;player<=nPlayers;player++){
           checkHiScore(pNames[player-1],pScores[player-1][TOTAL-1]);
           /*if(checkHiScore(pNames[player-1],pScores[player-1][TOTAL-1])!=-1){
              newScorers++;
           }*/
        }
        for(int i=1;i<=names.size();i++){
           if(names.get(i).charAt(0)=='\b'){
              newHSIndices.add(1, i);
              names.add(i, names.get(i).substring(1));//replacing string
              names.remove(i+1);
           }
        }
        displayHiScores();
     }
     //exception
     public class HiScoreListSyncException extends RuntimeException{
        public HiScoreListSyncException(String s){
           super(s);
        }//end constructor
     }
     //functions
     private int checkHiScore(String name,int score){//insert into list
        int size=scores.size();
        if(size==0){
           names.add(1, ('\b'+name));
           scores.add(1, score);
           return 1;
        } else {
           int i=size;
           for(;i>0&&score>scores.get(i);i--){//"outputs" index of higher score
           }
           //name=("\b"+name);
           names.add(i+1, ('\b'+name));
           scores.add(i+1, score);
           if(scores.size()>HI_SCORES){
              names.remove(HI_SCORES+1);
              scores.remove(HI_SCORES+1);
              if(i+1==HI_SCORES+1){
                 return -1;
              }
           }
           return i+1;
```

```
        }
        //enterName(HI_SCORES+1,score);
    }
    private void displayHiScores(){//index is where to insert entry
        double curX=0;
        double curY=0;
        yay.setVisible(!newHSIndices.isEmpty());
        yay.setFont(YAY_FONT);
        yay.setColor(new Color(0,237,0));
        add(yay,ctrX(yay),YAY_OFFSET+yay.getHeight());
        curY=YAY_OFFSET+yay.getHeight()+2*SCORES_SPACING;
        hiScoreDisp.setFont(HS_DISP_FONT);
        add(hiScoreDisp,ctrX(hiScoreDisp),curY+hiScoreDisp.getHeight());
        curY=curY+hiScoreDisp.getHeight();
        add(hiScoreDisp,ctrX(hiScoreDisp),curY);
        for(int i=1;i<=names.size();i++){
            //System.out.println("#"+i+":
"+names.get(i)+"--"+scores.get(i)+">>"+names.size());
            HSNames[i-1]=new GLabel("#"+i+": "+names.get(i));
            HSScores[i-1]=new GLabel(scores.get(i)+"");
            HSNames[i-1].setFont(SCORES_FONT);
            HSScores[i-1].setFont(SCORES_FONT);
            int x1=NAMES_OFFSET;
            int x2=(int)(getWidth()-SCORES_OFFSET-HSScores[i-1].getWidth());
            curY=curY+SCORES_SPACING+HSNames[i-1].getHeight();
            for(int j=1;j<=newHSIndices.size();j++){//based on list
                //System.out.println("HSINDICES"+newHSIndices.get(j));
                if(i==newHSIndices.get(j)){
                    if(HSNames[i-1].getLabel().endsWith("AI")){
                        HSNames[i-1].setColor(new Color(255,153,51));
                        HSScores[i-1].setColor(new Color(255,153,51));
                    } else {
                        HSNames[i-1].setColor(new Color(206,206,0));
                        HSScores[i-1].setColor(new Color(206,206,0));
                    }
                }
            }
            //animation
            int xnow2=(int)(-1*HSScores[i-1].getWidth());
            int xnow1=(int)(-1*getWidth()+xnow2);
            add(HSNames[i-1],xnow1,curY);
            add(HSScores[i-1],xnow2,curY);
            boolean isDone=false;
            while(!isDone){
                pause(1);
                isDone=true;
                if(x1>xnow1){
```

```
                HSNames[i-1].move(1,0);
                isDone=false;
            }
            if(x2>xnow2){
                HSScores[i-1].move(1,0);
                isDone=false;
            }
            xnow1=(int)HSNames[i-1].getX();
            xnow2=(int)HSScores[i-1].getX();
        }
      }
      backToLevel();
    }
    private void backToLevel(){
      GLabel click=new GLabel("-Click Anywhere to Play Again-");
      click.setFont(CLICK_FONT);
      add(click,ctrX(click),getHeight()-click.getHeight()-CLICK_Y_OFFSET);
      waitForClick();
      removeAll();
      addPlayers();
    }
}
```

Note that in here, as we were instructed before that we may use either the given linked list or static list implementation, I made use of the same static list implementation that I have used in the breakout game. Converting the MyList into the ArrayList was not done as the difference in bases between the two data types would make the program very prone to errors and bugs.

REFERENCES

1. E Roberts. *Art and Science of Java*. Pearson; 2013.
2. E Roberts, M Sahami, and M Stepp, *CS 106A: Programming Methodology (Java) Handouts*, Stanford University.