UML Diagram

Scorecard-----------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- slots[16]

- used[13]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+ int getSlots(int)

+ void setSlots(int, int)

+ bool getUsed(int)

+ void setUsed(int, bool)

+ int evaluateUpperScore(vector<Dice\*>, int)

+ int evaluateTotal(vector<Dice\*>)

+ int getOnes();

+ void setOnes(vector<Dice\*>);

+ int getTwos();

+ void setTwos(vector<Dice\*>);

+ int getThrees();

+ void setThrees(vector<Dice\*>);

+ int getFours();

+ void setFours(vector<Dice\*>);

+ int getFives();

+ void setFives(vector<Dice\*>);

+ int getSixes();

+ void setSixes(vector<Dice\*>);

+ int get3OAK();

+ void set3OAKs(vector<Dice\*>);

+ int get4OAK();

+ void set4OAK(vector<Dice\*>);

+ int getFullHouse();

+ void setFullHouse(vector<Dice\*>);

+ int getSStraight();

+ void setSStraight(vector<Dice\*>);

+ int getLStraight();

+ void setLStraight(vector<Dice\*>);

+ int getYahtzee();

+ void setYahtzee(vector<Dice\*>);

+ int getChance();

+ void setChance(vector<Dice\*>);

+ int getUpperScore();

+ int getLowerScore();

+ int getScore();

UML Diagram

------------<>Player

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- string name

- int gamesPlayed

- int highScore

- int lowScore

- double averageScore

- vector<int> gameScores

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+ Scorecard

+ Player(string, int)

+ saveGame()

+ getStats()

+ int getGamesPlayed()

+ vector<int> getScores()

+ string getName()

+ int getHighScore()

+ int getLowScore()

+ double getAVGscore()

+ void incrementGamesPlayed()

+ void addFinalScore()

+ void clearGame()

UML Diagram

Dice------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-int value;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+ Dice();

+ Dice(Dice\*);

+ Dice(int);

+ int getValue();

+ void setValue(int);

+ bool operator==(int);

+ void roll();

UML Diagram

------------<>Game

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+ Game()

+ void addDie(Dice\*, vector<Dice\*>&)

+ void firstRoll(vector<Dice\*>&, int&)

+ void replaceDie(Dice\*, vector<Dice\*>&, int)

+ void newRoll(vector<Dice\*>&, int, int&)

+ void chooseKept(vector<Dice\*>&, int&, int&)

For our final project, we require an easy to manipulate, easy to understand, and easy to use game of Yahtzee played entirely on the program console. It must display the dice rolled by the user, have a scorecard that is easily accessible in order to allow the user to make proper rolling decisions, and properly enforce the rules of Yahtzee to insure against cheating or misunderstanding of the rules. Ideally, the game will be fun to play and games could be saved while in the middle of execution.

Our program will include four classes, as shown above in the UML diagrams – Player, Scorecard, Game, and Dice. The Dice class is used to simulate the rolling of standard 6-sided dice as in real life. The game class contains dice and facilitates the actual playing of the game – rolling of the dice, choosing which dice to keep, etc. The scorecard class is used to keep track of player scores in each of the categories used in the game (ones, twos, small straight, Yahtzee, etc.) and displays the entire scorecard to the screen should the user ask for it (and after every dice scoring). The player class is used to track the people who play the game. It includes a string for the name of the player and data fields for their statistics such as high score, average score and number of games played. Each player employs a “has a” relationship with scorecard, such that during the game each player has their own scorecard for play.