Name: Joseph Picchi

UID: 605-124-511

Project 7 Report

a) Notable obstacles I overcame include the following:

* Parsing out the meanings and functions of various pieces of the skeleton code, especially since an understanding of these pre-defined functions was required in order to complete the rest of them. I also realized near the end of the project that although an understanding of all aspects of the skeleton code is necessary to gain an intuitive understanding of the functionality of the program, it was not necessarily needed to complete the stubbed-out functions that were given, as these could simply be completed based on the directions in the comments and the program would still be perfectly functional.
* Understanding the relationships between each of the classes. By spending time reading the program spec and understanding the skeleton code, I gradually developed an apprehension for the meaning of the “has-a” relationship between classes that Professor Stahl discussed in lecture.
* Knowing what functions should be called from what classes. The main issue with this also stemmed from a poor understanding of the predefined functions in the skeleton code at first. Once I took the time to review each of the myriad functions in each class, I understood the reasons for returning certain values in member functions (particularly the roll functions), which made it easier to implement successful function definitions.

b) A list of test data to thoroughly test my functions is provided below:

|  |
| --- |
| **Test Data and Reasons for Tests (given in comments)** |
| //Tests for the Die class:  Die d;  for (int i = 1; i <= 100; i++ )  {  d.roll();  int value = d.getValue();  assert( value >=1 && value <= 6 ); //Makes sure the 6-sided  //die always produces a  //value between 1 and 6.  } |
| //Tests for the Player class:  Die d;  Player p;  assert(p.getScore( ) == 0 ); //makes sure the constructor  //initializes Player.mScore to 0.  assert(p.getTotal( ) == 0 ); //makes sure the constructor  //initializes Player.mTotal to 0.  assert( p.roll( 6 ) == 6 ); //Makes sure Player::roll()  //rolls and returns a specific  //integer if that integer is passed  //to it as a parameter.  assert( p.getScore() == 6 ); //Makes sure Player::roll()  //correctly updates the value of  //Player.mScore, and ensures that  //Player::getScore returns  //Player.mScore  assert( p.roll( 5 ) == 5 );  assert( p.getScore() == 11 ); //Ensures that Player::roll()  //adds to Player.mScore for any  //roll other than 1  p.endTurn();  assert( p.getScore() == 0 ); //Makes sure Player::endturn()  //resets Player.mScore to 0  assert( p.getTotal() == 11 ); //Makes sure Player::endturn()  //updates Player.mTotal  assert( p.roll( 4 ) == 4 );  assert( p.getScore() == 4 );  assert( p.roll( 5 ) == 5 );  assert( p.getScore() == 9 );  assert( p.roll( 1 ) == 1 );  assert( p.getScore() == 0 ); //Ensures that a roll of 1 resets  //Player.mScore to 0  for (int i = 0; i < 100; i++) {  assert( p.roll( 1 ) == 1 );  }  assert(p.getScore() == 0 ); //Ensures that repetitively rolling a  // 1 keeps Player.mScore equal to 0  assert( p.getTotal() == 11 ); //Ensures that a roll of 1 does not  //change Player.mTotal |
| //Testing the Board class  Die d;  Player p;  Board board, b1;  assert( board.getComputerTotal() == 0 );  assert( board.getHumanTotal() == 0);  assert( board.getRolledValue() == 0);  assert( board.getComputerScore() == 0 );  assert( board.getHumanScore() == 0);  assert( board.isHumanWinner() == false );  assert( board.isHumanTurn() == true );  assert( board.isGameOver() == false ); //Makes sure the board  //constructor  //initializes member  //variables to the  //proper values  board.setComputerTurn();  board.setComputerTotal(1);  board.setComputerScore(2);  board.setHumanTurn();  board.setHumanScore(3);  board.setHumanTotal(4);  board.setRolledValue(7);  assert( board.getComputerTotal() == 1 );  assert( board.getHumanTotal() == 4);  assert( board.getRolledValue() == 7);  assert( board.getComputerScore() == 2 );  assert( board.getHumanScore() == 3);  assert( board.isHumanWinner() == false );  assert( board.isHumanTurn() == true );  assert( board.isGameOver() == false ); //Ensures that mutator  //functions can change  //member variables  //and accessor  //functions can see  //those changes  b1.setComputerTurn();  b1.setComputerTotal(21);  b1.setComputerScore(22);  b1.setHumanTurn();  b1.setComputerTotal( 50 );  b1.setComputerScore( 50 );  assert( b1.getComputerTotal() == 21 );  assert( b1.getComputerScore() == 22 ); //Prevents changes  //to the computer’s  //score and total if it  //isn’t the computer’s  //turn  b1.setHumanScore(23);  b1.setHumanTotal(24);  b1.setRolledValue(27);  b1.setRolledValue(250);  b1.setGameOver(true);  b1.markComputerAsWinner();  assert( b1.getComputerTotal() == 21 );  assert( b1.getHumanTotal() == 24);  assert( b1.getRolledValue() == 250);  assert( b1.getComputerScore() == 22 );  assert( b1.getHumanScore() == 23);  assert( b1.isHumanWinner() == false );  assert( b1.isHumanTurn() == true );  assert( b1.isGameOver() == true ); //Makes sure scores,  //totals, and winner  //do not reset once  //the game is over  b1.setComputerTurn();  b1.setComputerTotal( 50 );  b1.setComputerScore( 50 );  b1.setRolledValue( 50 );  b1.setHumanScore(50);  b1.setHumanTotal(50);  assert( b1.getComputerTotal( ) == 21 );  assert( b1.getComputerScore( ) == 22 );  assert( b1.getRolledValue( ) == 250 );  assert( b1.getHumanScore( ) == 23 );  assert( b1.getHumanTotal( ) == 24 ); //Ensures that no  //changes can be made  //to the board once  //the game ends |
| //Testing the Pig class  Pig game;  assert( game.isGameOver() == false );  assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER ); //Makes sure starting  //conditions for the  //game are correct  game.humanPlay( 5 );  game.humanEndTurn( );  game.computerPlay( 5 );  game.computerEndTurn( );  game.humanPlay( 5 );  game.humanPlay( 5 );  game.humanPlay( 5 );  game.humanEndTurn( );  game.computerPlay( 5 );  game.computerPlay( 5 );  game.computerPlay( 5 );  assert( game.isGameOver() == false );  assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  game.humanPlay( 5 );  game.humanPlay( 6 );  game.humanPlay( 1 );  assert( game.isGameOver() == false );  assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER ); //Ensures that the game  //is not over until a  //player’s total is  //greater than or equal  //to 30 points  // once the game is over, scores can't be changed  game.computerPlay( 6 );  game.computerPlay( 6 );  game.computerPlay( 6 );  game.computerEndTurn( );  assert( game.isGameOver() == true );  assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME ); //Ensures that the game  //is marked over and  //the proper winner is  //marked once a player  //reaches 30 points  game.humanPlay( 6 );  game.humanPlay( 6 );  game.humanPlay( 6 );  game.humanEndTurn( );  assert( game.isGameOver() == true );  assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME ); //Makes sure scores and  //outcomes can’t be  //changed once the game  //ends  //Testing a new game outcome when the human wins:  Pig g1;    assert( g1.isGameOver() == false );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  g1.humanPlay( 5 );  g1.humanEndTurn( );  g1.computerPlay( 5 );  g1.computerEndTurn( );  assert( g1.isGameOver() == false );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  g1.humanPlay( 5 );  g1.humanPlay( 5 );  g1.humanPlay( 5 );  g1.humanEndTurn( );  assert( g1.isGameOver() == false );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  g1.computerPlay( 5 );  g1.computerPlay( 5 );  g1.computerPlay( 5 );  assert( g1.isGameOver() == false );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  g1.humanPlay( 5 );  g1.humanPlay( 6 );  g1.humanPlay( 1 );  assert( g1.isGameOver() == false );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  g1.humanPlay( 6 ); //Ensures that the  //human’s turn does  //not end until  //Pig::humanEndturn();  //is called  g1.humanPlay( 6 );  g1.humanPlay( 6 );  g1.humanEndTurn( );  assert( g1.isGameOver() == true );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); //makes sure the human  //is marked winner if he  //reaches 30 or more  //points before the  //computer  g1.computerPlay( 6 );  g1.computerPlay( 6 );  g1.computerPlay( 6 );  g1.computerEndTurn( );  assert( g1.isGameOver() == true );  assert( g1.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); //Makes sure the outcome  //doesn’t change after  //the game ends and the  //human is the winner  //Testing what happens when the players repeatedly roll 1:  Pig g3;  assert( g3.isGameOver() == false );  assert( g3.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  for (int i = 0; i < 100; i++) {  g3.humanPlay( 1 );  }  g3.humanEndTurn( );  for (int i = 0; i < 100; i++) {  g3.computerPlay( 1 );  }  g3.computerEndTurn( );  assert( g3.isGameOver() == false );  assert( g3.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );  //Ensures that the game  //can never end is both  //players always roll 1 |

NOTE: The program I’ve created handles all of these test cases correctly.