Bryan Wong

805111517

Professor Stahl

CS31

March 15, 2019

Project 7 Report

1. Over the course of completing this project, I ran into minor obstacles. The most difficult part of the project was figuring out how the different classes interacted with each other, and what actions were already performed automatically by functions that I was not assigned to write. For example, I had originally thought that I would need to implement endturn() inside the roll function of Player when a 1 was rolled. However, the endturn() function was already being mediated by the main file and the humanEndTurn() function in Pig. I was unclear at first how the game was exactly organized, and came to realize later that the main.cpp file actually dictated most of the flow of the game, while the other classes helped it perform this function. The main difficulty in this project was analyzing existing code to familiarize myself with and build upon.

2. Here are my test cases:

(As given by the spec)

**First, we create some objects to test with:**

Die d;

Player p;

Board board, b1;

Pig game;

**// test code for Die to make sure all values are from 1-6**

for (int i = 1; i <= 100; i++ )

{

d.roll();

int value = d.getValue();

assert( value >=1 && value <= 6 );

}

**// test code for Player**

**// test the default constructor to make sure it sets score and total to 0 and a 6-sided die correctly**

assert(p.getScore( ) == 0 );

assert(p.getTotal( ) == 0 );

assert( p.roll( 6 ) == 6 ); **// When a non-zero int is inputted into roll( ), it “cheats” that value as the output**

assert( p.getScore() == 6 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 11 ); **// Test that the individual scores of 6 and 5 indeed add up to 11**

p.endTurn(); **// Test that ending the turn resets the score, but not the total**

assert( p.getScore() == 0 );

assert( p.getTotal() == 11 );

assert( p.roll( 4 ) == 4 );

assert( p.getScore() == 4 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 9 );

assert( p.roll( 6 ) == 6 );

assert( p.getScore() == 15 ); **// Once again, we add 3 individual scores, to 15 this time**

p.endTurn();

assert( p.getScore() == 0 );

assert( p.getTotal() == 26 ); **// Test that totals are cumulative over multiple turns**

assert( p.roll( 4 ) == 4 );

assert( p.getScore() == 4 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 9 );

assert( p.roll( 1 ) == 1 ); **// Test that rolling a 1 sets the score to 0**

assert( p.getScore() == 0 );

assert( p.getTotal() == 26 ); **// Total is not changed, since the score for the last round was 0**

cout << "Player tests passed!" << endl;

**// test code for Board**

assert( board.getComputerTotal() == 0 ); **// This block tests that the default constructor for Board is behaving correctly in setting its default values**

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 );

board.setComputerTurn(); **// This block tests the getters and setters for Board**

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 );

**// Once again, we set Board b1’s values using setters**

b1.setComputerTurn();

b1.setComputerTotal(21); **// Changing the Computer’s values are ok if it’s the Computer’s turn**

b1.setComputerScore(22);

b1.setHumanTurn();

b1.setComputerTotal( 50 ); **// Changing the Computer’s values is NOT ok if it’s the Human’s turn. We notice that its total and score are unchanged**

b1.setComputerScore( 50 );

assert( b1.getComputerTotal() == 21 );

assert( b1.getComputerScore() == 22 );

b1.setHumanScore(23); **// We use the Board’s setters again**

b1.setHumanTotal(24);

b1.setRolledValue(27);

b1.setRolledValue(250);

b1.setGameOver(true);

b1.markComputerAsWinner();

assert( b1.getComputerTotal() == 21 ); **// We check that the setters work correctly by using the getters**

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 );

**// We check that board changes don’t work now, since the game is over. They remain their previous values.**

b1.setComputerTurn();

b1.setComputerTotal( 50 );

b1.setComputerScore( 50 );

b1.setRolledValue( 50 );

assert( b1.getComputerTotal( ) == 21 );

assert( b1.getComputerScore( ) == 22 );

assert( b1.getRolledValue( ) == 250 );

**// test code for Game**

assert( game.isGameOver() == false ); **// Game should not be over by default**

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER ); **// Testing determineGameOutcome( )**

game.humanPlay( 5 ); **// From here, we “cheat” out rolls for testing by feeding humanPlay( ) defined ints.**

game.humanEndTurn( );

game.computerPlay( 5 );

game.computerEndTurn( );

assert( game.isGameOver() == false ); **// Game shouldn’t be over yet. Check both ways.**

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

game.humanPlay( 5 );

game.humanPlay( 5 );

game.humanPlay( 5 ); **// Human now has 20 points**

game.humanEndTurn( );

assert( game.isGameOver() == false );

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

game.computerPlay( 5 );

game.computerPlay( 5 );

game.computerPlay( 5 ); **// Computer now has 20 points**

assert( game.isGameOver() == false );

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

game.humanPlay( 5 );

game.humanPlay( 6 ); **// Human now has 31 points, more than the 30 required to win.**

game.humanEndTurn( );

assert( game.isGameOver() == true ); **// When we check if the game is over, it returns true**

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME );

**// Once one player wins the game, no more board changes can be made.**

game.computerPlay( 6 );

game.computerPlay( 6 );

game.computerPlay( 6 ); **// If scoring was still allowed, this would cause the computer to have more points**

game.computerEndTurn( );

assert( game.isGameOver() == true );

assert( game.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); **// The human should still have won the game**

**If we were to test what the private members of mBoard inside game included, isHumanWinner() would return true since the function isGameOver() sets it as such.**

Pig newGame;

assert(newGame.humanPlay(31) == 31); **// Setting the human to roll huge amounts works!**

assert(newGame.isGameOver() == false); **// Game isn't over until we end turn**

assert(newGame.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER);

display( newGame, "", false); **// We display to check**

newGame.humanEndTurn();

assert(newGame.isGameOver() == true); **// Now, the game is over**

assert(newGame.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME);

display( newGame, "", false);

Pig newGame2;

newGame2.humanEndTurn();

assert(newGame2.computerPlay(-1) == -1); **// Rolling -1 numbers works too. Verify this with assert.**

assert(newGame2.computerPlay(30) == 30);

newGame2.computerEndTurn();

display(newGame2,"", false);

assert(newGame2.isGameOver() == false); **// Game isn't over, since the computer only has 29 points. Proof the -1 works.**

assert(newGame2.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER);

newGame2.humanEndTurn();

assert(newGame2.computerPlay(-1) == -1); **// We do this because playing a 1 instead resets the score to 0**

assert(newGame2.computerPlay(2) == 2);

newGame2.computerEndTurn();

display(newGame2, "", false);

assert(newGame2.isGameOver() == true); **// Now, the game is over**

assert(newGame2.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME); **// The computer can win, too**

display(newGame2,"", false);

Pig newGame3;

assert(newGame3.humanPlay(30) == 30);

assert(newGame3.humanPlay(1) == 1);

newGame3.humanEndTurn();

assert(newGame3.isGameOver() == false); **// Rolling a 1 resets the score, even if you had a winning score before**

assert(newGame3.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER);