Name: Avnish Sengupta

UCLA ID: 705299746

Notable Obstacles Overcome: -

* While creating the whatSpotIsNeededNext() function, the hasRolled Booleans that were true based on the value showed on the dice, from the roll or roll cheating function, remained true even after the turn ended and some of the higher hasRolled that were true but were inconsequential to the play of the player has they were not sequential still remained true and messed with the future spots that were returned by the function. To overcome this particular function, I manually made a long switch case that, depending on what the function returned, turned all the hasRolled function for the spots greater than or equal to the Boolean value to false.
* While creating the computerPlay and HumanPlay functions, it was hard for me to figure out a solution to check for multiple spots that might be available based on the dice values since putting in a while loop and updating the value using setHumanSpot() function did not update the value returned by whatSpotIsNeededNext() function was hard since it did not change any member functions of the player class since it is independent of the board class. To overcome this, I made the while loop with the spot getting updated and called the function before the dice were rolled and just made Player and Board functions return the correct values independent of the execution of either.
* Updating all the myriad of variables in the centennial class due to aggregation from the Board and Player and Die class.

Test Data to test my Code: -

int i, value;

Die d1;

Die d2;

Die d,d3,d4,d5,d6;

for (i = 1; i <= 50; i++)

{

d.roll();

value = d.getValue();

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

}

// Player test code

Player p, human, computer;

Player p2, human2, computer2; //to simulate multiple games.

// in the beginning of time, nothing has been rolled yet and the spot needed is 1...

assert(!p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(!p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 1);

// now the player has rolled 1... so the spot next needed is 2...

p.rolled(1);

assert(p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(!p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 2);

// only rolls from 1-12 are relevant...

p.rolled(100);

assert(p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(p.whatSpotIsNeededNext() == 2);

// rolls must be sequential for things to count...

//rolls must also turn the appropriate the hasRolled booleans to true.

p.rolled(3);

assert(p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(p.whatSpotIsNeededNext() == 2);

p.rolled(2);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(!p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 3);

p.rolled(3);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(!p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

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

p.rolled(4);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

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

p.rolled(5);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

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

p.rolled(6);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 7);

p.rolled(7);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 8);

p.rolled(8);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

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

p.rolled(9);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(p.hasRolledEight());

assert(p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 10);

p.rolled(10);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(p.hasRolledEight());

assert(p.hasRolledNine());

assert(p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

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

p.rolled(11);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(p.hasRolledEight());

assert(p.hasRolledNine());

assert(p.hasRolledTen());

assert(p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 12);

p.rolled(12);

assert(p.hasRolledOne());

assert(p.hasRolledTwo());

assert(p.hasRolledThree());

assert(p.hasRolledFour());

assert(p.hasRolledFive());

assert(p.hasRolledSix());

assert(p.hasRolledSeven());

assert(p.hasRolledEight());

assert(p.hasRolledNine());

assert(p.hasRolledTen());

assert(p.hasRolledEleven());

assert(p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 13);

// working the Player via Dies

d1.setValue(6);

d2.setValue(5);

d3.setValue(4);

p.roll(d1, d2, d3);

assert(p.whatWasRolled() == "Die1: 6 Die2: 5 Die3: 4\n");

// Board test code

Board b;

assert(b.getHumanSpot() == 0); //at the beginning both the players must be at 0.

assert(b.getComputerSpot() == 0);

assert(b.isGameOver() == false); //the game should not be over at the beginning.

assert(b.isHumanWinner() == false);

b.setHumanSpot(3);

b.setComputerSpot(6);

assert(b.getHumanSpot() == 3); //the mutilator should have changed the value returned by the accessor.

assert(b.getComputerSpot() == 6);

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

assert(b.isHumanWinner() == false);

b.setHumanSpot(12);

assert(b.getHumanSpot() == 12);

assert(b.getComputerSpot() == 6);

assert(b.isGameOver() == false); //since the Board is just a bunch of data to be used by Centennial and only keeps track of the spots of the Human and Computer.

assert(b.isHumanWinner() == false);

b.setGameOver(true);

b.markHumanAsWinner();

assert(b.isGameOver() == true); //only the markHumanAsWinner function should set the correct Booleans to true.

assert(b.isHumanWinner() == true);

// Centennial test code

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

assert(game.determineGameOutcome() == Centennial::GAMENOTOVER);

human = game.getHuman();

computer = game.getComputer();

assert(human.whatSpotIsNeededNext() == 1);

assert(computer.whatSpotIsNeededNext() == 1);

d1.setValue(1); //initializing die variables.

d2.setValue(2);

d3.setValue(3);

d4.setValue(4);

d5.setValue(5);

d6.setValue(6);

game.humanPlay(d6, d5, d4); //to check the response to decrementing sequential values.

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 1);

game.computerPlay(d1, d2, d3); // to check the response to incrementing sequential values.

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 7);

game.humanPlay(d4, d2, d1); //to check the response where the maximum combination is the desired combination.

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 8);

game.computerPlay(d5, d2, d1); //to check the response where the maximum combination is the desired combination.

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

game.humanPlay(d6, d2, d3); //to check if the game differentiates improper inputs based on the sequence of the board.

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 10);

game.computerPlay(d1, d2, d3); //to check if the function responds to ineffective input.

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

game.humanPlay(d4, d5, d6); //to check the response to reasonable input where the maximum is greater than the spot allotted to the board..

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 12);

game.computerPlay(d3, d2, d1); //to check the response to decrementing sequential values that form an ineffective input.

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

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

//to check if the correct enum value is returned although the spot needed to win for one of the players is 12.

assert(game.determineGameOutcome() == Centennial::GAMENOTOVER);

game.humanPlay(d2, d4, d6);

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

assert(game.determineGameOutcome() == Centennial::HUMANWONGAME);

game3.humanPlay(d1, d1, d1); //to check the response to a triple thrown.

assert(game3.isGameOver() == false);

human2 = game3.getHuman();

assert(human2.whatSpotIsNeededNext() == 4);

//to check the response to sequential incrementing values where the maximum is greater than points allotted to the board and where it is ineffective input.

game3.humanPlay(d4, d5, d6);

assert(game3.isGameOver() == false);

human2 = game3.getHuman();

assert(human2.whatSpotIsNeededNext() == 7);

computer2 = game3.getComputer();

game3.computerPlay(d3, d2, d1); //to check the response to sequential decrementing values that amount to effective input.

assert(game3.isGameOver() == false);

computer2 = game3.getComputer();

assert(computer2.whatSpotIsNeededNext() == 7);

game3.computerPlay(d1, d3, d5); //check the response to only odd values returned.

assert(game3.isGameOver() == false);

computer2 = game3.getComputer();

assert(game3.determineGameOutcome() == Centennial::GAMENOTOVER);

assert(computer2.whatSpotIsNeededNext() == 7);

game3.humanPlay(d4, d2, d6); //to check the response to even values returned.

assert(game3.isGameOver() == false);

assert(game3.determineGameOutcome() == Centennial::GAMENOTOVER);

human2 = game3.getHuman();

assert(human2.whatSpotIsNeededNext() == 7);

game3.humanPlay(d4, d1, d5); //to check the response to odd and even values mixed.

assert(game3.isGameOver() == false);

assert(game3.determineGameOutcome() == Centennial::GAMENOTOVER);

human2 = game3.getHuman();

assert(human2.whatSpotIsNeededNext() == 7);