Kyra Noguchi

705954527

Overall, this project was pretty difficult for me. I tend to have some trouble and confusion with classes and working with different files. However, I rewatched some of the BruinCast lectures and relooked over the discussion slides/worksheets to familiarize myself better with what I was working with. Initially, all of the code that was given to us to work with was overwhelming at first sight. I had to look through each file and the instructions to see what variable did what and how each class functioned. In addition, when building my own code, I had to ensure that I was using the correct variables that were given to us.

As expected from what the professor said, working with scoring “maybe” was pretty difficult. Initially, I had created a nested for loop. But upon further looking at how the scoring worked, I realized that my code would not function properly for certain test cases. In order to solve this problem, I created a separate array called storeArray that stored whether a score had already been assigned to an index of the inputted answer. In addition, when I had put my code into CodeBoard for the first time, I kept receiving an error at runtime. This was because of a small issue where my functions for getMoveRound and getScoreForRound were returning a variable that did not do anything and was not initialized. Instead, I retrieved the past move/score and returned it.

One more obstacle I ran into was my asserts failing when testing moves and answers that were reversed of each other. For example, a move being “BBOP” and the answer being “POBB”. The assert would fail for the assert(s.to\_string() == “MMMM”). Upon further testing and inspection, I discovered that I needed to break out of the loop in my Score constructor that set the maybes.

I want to note that in the main.cpp I am submitting, I commented out all of my test cases.

Test Data:

Piece p;

assert( p.getColor() == NOTVALID );

p = Piece( "G" );

assert( p.getColor() == GREEN );

assert( p.getColorAsString() == "G" );

* Tests if NOTVALID is returned when a piece is not assigned yet
* Tests if the getter functions return the right color or color as a string when “G: is assigned to the piece (applies to all colors)

Move m;

m.setPieces( "GOPY" );

p = m.getPiece( 2 );

assert( p.getColor() == PURPLE );

m.setPiece( 2, 'b' );

p = m.getPiece( 2 );

assert( p.getColor() == BLUE );

* Sets a move and checks if the right pieces are in that move
* Sets a piece within the move to a new color and uses getter function to check if the piece was set correctly

Score s;

m.setPieces( "POGB" );

Move theAnswer;

theAnswer.setPieces( "RYRY" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **false** );

assert( s.to\_string() == "\_\_\_\_" );

theAnswer.setPieces( "pogb" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **true** );

assert( s.to\_string() == "RRRR" );

* checks that isExactMatch() and s.to\_string() return the right values

Board b;

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

m.setPieces( "POPO" );

theAnswer.setPieces( "YYOP" );

s = Score( m, theAnswer );

b.endRound( m, s );

assert( b.getCurrentRound() == 1 );

assert( b.getMoveForRound( 0 ).to\_string() == "POPO" );

assert( b.getScoreForRound( 0 ).to\_string() == "MM\_\_" );

* Checks if getCurrentRound returns the right value, even when no round has been played yet
* Checks if getMoveForRound() returns the right move for the move during the specified round
* Checks if getScoreForRound() returns the right score for the move during the specified round

Mastermind game( "rbyo" );

assert( game.answer() == "RBYO" );

assert( game.gameIsOver() == **false** );

m = game.play( "BBBB" );

s = game.endRound( m );

assert( s.to\_string() == "\_R\_\_" );

* Checks is gameIsOver() returns the right boolean value when the game is not yet oover since the answer is not correct OR 4 rounds haven’t been played

m.setPieces( "GGOP" );

Score s;

Move theAnswer;

theAnswer.setPieces( "OPGG" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **false** );

assert( s.to\_string() == "MMMM" );

* Checks if words that are reverse of eachother return all maybes

Move m;

m.setPieces( "BBBR" );

Score s;

Move theAnswer;

theAnswer.setPieces( "PBBG" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **false** );

assert( s.to\_string() == "\_RR\_" );

* checks that isExactMatch() and s.to\_string() return the right values

Move m;

m.setPieces( "PPOB" );

Score s;

Move theAnswer;

theAnswer.setPieces( "YYGG" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **false** );

assert( s.to\_string() == "\_\_\_\_" );

* checks that isExactMatch() and s.to\_string() return the right values

Move m;

m.setPieces( "BBRR" );

Score s;

Move theAnswer;

theAnswer.setPieces( "RPBP" );

s = Score( m, theAnswer );

assert( s.isExactMatch() == **false** );

assert( s.to\_string() == "M\_M\_" );

* checks that isExactMatch() and s.to\_string() return the right values

Move m;

m.setPieces( "PYBR" );

p = m.getPiece( 3 );

assert( p.getColor() == RED);

m.setPiece( 0, 'Y' );

p = m.getPiece( 0 );

assert( p.getColor() == YELLOW );

* Sets a move and checks if the right pieces are in that move
* Sets a piece within the move to a new color and uses getter function to check if the piece was set correctly

Board b;

m.setPieces( "YBBR" );

theAnswer.setPieces( "YBBR" );

s = Score( m, theAnswer );

b.endRound( m, s );

assert( b.getCurrentRound() == 1 );

assert( b.getMoveForRound( 0 ).to\_string() == "YBBR" );

assert( b.getScoreForRound( 0 ).to\_string() == "RRRR" );

* Checks if getCurrentRound returns the right value, even when no round has been played yet
* Checks if getMoveForRound() returns the right move for the move during the specified round
* Checks if getScoreForRound() returns the right score for the move during the specified round