1. A brief description of notable obstacles you overcame.

I kept using mTurnCount instead of .TurnsLeft() function which caused problems for the program. I also was using mDie1, mDie2 instead of the .getValue() function when trying to compare the values of the die. In the end I learned my conceptual misunderstanding and fixed my code. I also did not notice that theBoard member of the BeatThat class was supposed to be independent from how the game is played but just be a display of the scores.

1. A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

Note: the the skeleton code that was provided prevents entering bad data so bad data such as Die d1; d1.setValue(-100) will not be considered since it is not accepted by the program.

**CLASS PLAYER**

**TEST CASE 1** Void roll()

**Reason: must roll a random number between 1 and the (number of sides of the die +1)**

Die d(17);

d.roll();

d.setValue(1) // repeat for 1-18 , shouldn’t work for 18

assert( (d.getValue() < 18) && (d.getValue() > 0));

Die e;

e.roll();

e.setValue(1) // repeat for 1-7 , shouldn’t work for 7

assert( (e.getValue() < 7) && (e.getValue() > 0));

**TEST CASE 2** void wonARound()

**Reason: wonAround should increment the Roundswon once after it is called**

Player p;  
assert( p.getRoundsWon() == 0 );  
p.wonARound();

assert( p.getRoundsWon() == 1 );  
p.wonARound();

p.wonARound();

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

**TEST CASE 3** int getRoundsWon()

**Reason: getRoundsWon should be zero at the start**

Player p;  
assert( p.getRoundsWon( ) == 0 );

Die largestDie() const

Die smallestDie() const

**TEST CASE 4**

**Reason: getRoundsWon should get the correct amount of rounds won**

Player p;  
assert( p.getRoundsWon( ) == 0 );  
p.wonARound();  
p.wonARound();  
assert( p.getRoundsWon( ) == 2 );

p.wonARound();

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

**TEST CASE 5** void largestDie(), void smallestDie()

**Reason: Player largestDie and smallestDie must work correctly**

Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Player p;  
p.roll( d1, d2 );

Die d3;

d3.setValue( 3 );  
Die d4;  
d4.setValue( 3 );  
 p.roll( d1, d2 );  
assert( p.largestDie( ).getValue( ) == 2 );  
assert( p.smallestDie( ).getValue( ) == 1 );

p.roll(d3, d4);

assert( p.largestDie( ).getValue( ) == 3 );  
assert( p.smallestDie( ).getValue( ) == 3 );

**CLASS BOARD**

**TESTCASE 6** bool didComputerWin() const

**Reason: must return true if computer is marked as winner, false otherwise**

Board b;  
b.setHumanRoundsWon( 3 );  
b.setComputerRoundsWon( 2 );  
b.markHumanAsWinner();  
b.setTurnsLeft(0);  
assert( b.getHumanRoundsWon() == 3);  
assert( b.getComputerRoundsWon() == 2 );  
assert( b.getTurnsLeft() == 0 );  
assert( !b.didComputerWin() );

b.setComputerRoundsWon( 4 );

assert(!b.didComputerWin() );

Board c;  
c.setHumanRoundsWon( 1 );  
c.setComputerRoundsWon( 2 );  
c.markHumanAsWinner();  
c.setTurnsLeft(0);  
assert( c.getHumanRoundsWon() == 1);  
assert( c.getComputerRoundsWon() == 2 );  
assert( c.getTurnsLeft() == 0 );  
assert( !c.didComputerWin() );

Board d;  
d.setHumanRoundsWon( 0 );  
d.setComputerRoundsWon( 0 );  
d.markHumanAsWinner();  
d.setTurnsLeft(0);  
assert( d.getHumanRoundsWon() == 0);  
assert( d.getComputerRoundsWon() == 0);  
assert( d.getTurnsLeft() == 0 );  
assert( !d.didComputerWin() );

assert( d.didHumanWin());

Board e;  
e.setHumanRoundsWon( 0 );  
e.setComputerRoundsWon( 0 );  
e.markComputerAsWinner();  
e.setTurnsLeft(0);  
assert( e.getHumanRoundsWon() == 0);  
assert( e.getComputerRoundsWon() == 0);  
assert( e.getTurnsLeft() == 0 );  
assert( e.didComputerWin() );

assert( !e.didHumanWin());

Board f;  
f.setHumanRoundsWon( 2 );  
f.setComputerRoundsWon( 0 );  
f.markComputerAsWinner();  
f.setTurnsLeft(0);  
assert( f.getHumanRoundsWon() == 2);  
assert( f.getComputerRoundsWon() == 0);  
assert( f.getTurnsLeft() == 0 );  
assert( f.didComputerWin() );

assert( !f.didHumanWin());

Board h;  
h.setHumanRoundsWon( 1 );  
h.setComputerRoundsWon( 2 );  
h.markComputerAsWinner();  
h.setTurnsLeft(0);  
assert( h.getHumanRoundsWon() == 1);  
assert( h.getComputerRoundsWon() == 2 );  
assert( h.getTurnsLeft() == 0 );  
assert( h.didComputerWin() );

Board b;  
assert( b.getHumanRoundsWon( ) == 0 );  
assert( b.getComputerRoundsWon( ) == 0 );  
assert( b.getTurnsLeft( ) == 0 );  
assert( ! b.didHumanWin( ) );  
assert( ! b.didComputerWin( ) );  
assert( ! b.isGameOver() );  
  
b.setHumanRoundsWon( 1 );  
b.setComputerRoundsWon( 2 );  
b.setTurnsLeft( 3 );  
assert( b.getHumanRoundsWon( ) == 1 );  
assert( b.getComputerRoundsWon( ) == 2 );  
assert( b.getTurnsLeft( ) == 3 );

b.markComputerAsWinner( );  
  
assert( ! b.didHumanWin( ) );  
assert( b.didComputerWin( ) );

**TEST CASE 7**

**Reason: must return false if game is marked tied even if computer was marked as winner before**

Board b;  
b.setHumanRoundsWon( 1 );  
b.setComputerRoundsWon( 1 );

b.markComputerAsWinner();  
b.markTied();  
b.setTurnsLeft(0);  
assert( b.getHumanRoundsWon() == 1 );  
assert( b.getComputerRoundsWon() == 1 );  
assert( b.getTurnsLeft() == 0 );  
assert( !b.didComputerWin() );

Board k;  
k.setHumanRoundsWon( 1 );  
k.setComputerRoundsWon( 2 );  
k.markTied();  
k.setTurnsLeft(0);  
assert( k.getHumanRoundsWon() == 1 );  
assert( k.getComputerRoundsWon() == 2 );  
assert( k.getTurnsLeft() == 0 );  
assert( !k.didComputerWin() );

**TEST CASE 8** bool didHumanWin() const

**Reason: must return true if human is marked as winner and false otherwise**

Board b;  
b.setHumanRoundsWon( 2);  
b.setComputerRoundsWon( 3);  
b.markComputerAsWinner();  
b.setTurnsLeft(0);  
assert( b.getHumanRoundsWon() == 2);  
assert( b.getComputerRoundsWon() == 3 );  
assert( b.getTurnsLeft() == 0 );  
assert( b.didComputerWin() );

assert( !b.didHumanWin());

b.setHumanRoundsWon( 4 );

assert(b.didComputerWin() );

assert( !b.didHumanWin());

assert( b.didHumanWin());

Board c;  
c.setHumanRoundsWon( 2 );  
c.setComputerRoundsWon( 1 );  
c.markComputerAsWinner();  
c.setTurnsLeft(0);  
assert( c.getHumanRoundsWon() == 2);  
assert( c.getComputerRoundsWon() == 1 );  
assert( c.getTurnsLeft() == 0 );  
assert( c.didComputerWin() );

assert( !c.didHumanWin());

Board d;  
d.setHumanRoundsWon( 0 );  
d.setComputerRoundsWon( 0 );  
d.markComputerAsWinner();  
d.setTurnsLeft(0);  
assert( d.getHumanRoundsWon() == 0);  
assert( d.getComputerRoundsWon() == 0);  
assert( d.getTurnsLeft() == 0 );  
assert( d.didComputerWin() );

assert( !d.didHumanWin());

Board e;  
e.setHumanRoundsWon( 0 );  
e.setComputerRoundsWon( 0 );  
e.markHumanAsWinner();  
e.setTurnsLeft(0);  
assert( e.getHumanRoundsWon() == 0);  
assert( e.getComputerRoundsWon() == 0);  
assert( e.getTurnsLeft() == 0 );  
assert( !e.didComputerWin() );

assert( e.didHumanWin());

Board f;  
f.setHumanRoundsWon( 0 );  
f.setComputerRoundsWon( 2 );  
f.markHumanAsWinner();  
f.setTurnsLeft(0);  
assert( f.getHumanRoundsWon() == 0);  
assert( f.getComputerRoundsWon() == 2);  
assert( f.getTurnsLeft() == 0 );  
assert( !f.didComputerWin() );

assert( f.didHumanWin());

Board h;  
h.setHumanRoundsWon( 2 );  
h.setComputerRoundsWon( 0 );  
h.markComputerAsWinner();  
h.setTurnsLeft(0);  
assert( h.getHumanRoundsWon() == 2);  
assert( h.getComputerRoundsWon() == 0 );  
assert( h.getTurnsLeft() == 0 );  
assert( !h.didComputerWin() );

assert( h.didHumanWin());

Board b;  
assert( b.getHumanRoundsWon( ) == 0 );  
assert( b.getComputerRoundsWon( ) == 0 );  
assert( b.getTurnsLeft( ) == 0 );  
assert( ! b.didHumanWin( ) );  
assert( ! b.didComputerWin( ) );  
  
  
b.setHumanRoundsWon( 1 );  
b.setComputerRoundsWon( 2 );  
b.setTurnsLeft( 3 );  
assert( b.getHumanRoundsWon( ) == 1 );  
assert( b.getComputerRoundsWon( ) == 2 );  
assert( b.getTurnsLeft( ) == 3 );

b.markComputerAsWinner( );  
  
assert( ! b.didHumanWin( ) );  
assert( b.didComputerWin( ) );  
assert( b.isGameOver( ) );

**TEST CASE 9**

**Reason: must return false if game is marked tied even if computer was marked as winner before**

Board b;  
b.setHumanRoundsWon( 1 );  
b.setComputerRoundsWon( 1 );

b.markHumanAsWinner();  
b.markTied();  
b.setTurnsLeft(0);  
assert( b.getHumanRoundsWon() == 1 );  
assert( b.getComputerRoundsWon() == 1 );  
assert( b.getTurnsLeft() == 0 );  
assert( !b.didComputerWin() );

assert( !b.didHumanWin());

Board k;  
k.setHumanRoundsWon( 1 );  
k.setComputerRoundsWon( 2 );  
k.markTied();  
k.setTurnsLeft(0);  
assert( k.getHumanRoundsWon() == 1 );  
assert( k.getComputerRoundsWon() == 2 );  
assert( k.getTurnsLeft() == 0 );  
assert( !k.didComputerWin() );

assert( !k.didHumanWin());

**TEST CASE 10** void markComputerAsWinner()

**Reason: didComputerWin function must return true after this is called. This can be tested using all the test cases for the didComputerWin function**

**TEST CASE 11** void markHumanAsWinner()

**Reason: didHumanWin function must return true after this is called. This can be tested using all the test cases for the didHumanWin funciton**

**TEST CASE 12** void markTied()

**Reason: didHumanWin and didComputerWin must both return false if the board is markTied. This can be tested using all the test cases for the didHumanWin and didComputerWin funcitons**

**TEST CASE 13** bool isGameOver() const

**Reason: The game must be over if the board is markTied, markHumanAsWinner, markComputerAsWinner**

Board b;  
assert( b.getHumanRoundsWon( ) == 0 );  
assert( b.getComputerRoundsWon( ) == 0 );  
assert( b.getTurnsLeft( ) == 0 );  
assert( ! b.didHumanWin( ) );  
assert( ! b.didComputerWin( ) );  
assert( ! b.isGameOver() );  
  
b.setHumanRoundsWon( 1 );  
b.setComputerRoundsWon( 2 );  
b.setTurnsLeft( 3 );  
assert( b.getHumanRoundsWon( ) == 1 );  
assert( b.getComputerRoundsWon( ) == 2 );  
assert( b.getTurnsLeft( ) == 3 );

b.markComputerAsWinner( );  
  
assert( ! b.didHumanWin( ) );  
assert( b.didComputerWin( ) );  
assert( b.isGameOver( ) );

Board c;  
c.setHumanRoundsWon( 1 );  
c.setComputerRoundsWon( 2 );  
c.setTurnsLeft( 3 );

c.markHumanAsWinner( );  
  
assert( c.didHumanWin( ) );  
assert(! c.didComputerWin( ) );  
assert( c.isGameOver( ) );

Board d;  
d.setHumanRoundsWon( 1 );  
d.setComputerRoundsWon( 2 );  
d.setTurnsLeft( 3 );

d.markTied( );  
  
assert( ! d.didHumanWin( ) );  
assert(! d.didComputerWin( ) );  
assert( d.isGameOver( ) );

**CLASS BEATTHAT**

**TEST CASE 14** void humanPlay()

**Reason: must randomly roll humans die**

BeatThat g;  
g.humanPlay( );  
g.computerPlay();  
g.endTurn( );  
assert( g.turnsLeft( ) == 4 );  
assert( !g.isGameOver( ) );

std:: cout << G.getHuman().whatWasRolled(); // run at least twice to see if values are random

**TEST CASE 15** void humanPlay(Die d1, Die d2) and void computerPlay(Die d1, Die d2)

**Reason: human and computer must roll dies and the dies must have the specified value**

BeatThat g;  
Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Die d3;  
d3.setValue( 3 );  
Die d4;  
d4.setValue( 4 );  
g.humanPlay( d1, d2 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
assert( g.turnsLeft( ) == 4 );  
assert( !g.isGameOver( ) );  
assert( g.getHuman().getRoundsWon( ) == 0 );  
assert( g.getComputer().getRoundsWon( ) == 1);

**TEST CASE 16** void computerPlay()

**Reason: must randomly roll computers die**

BeatThat g;  
g.humanPlay( );  
g.computerPlay();  
g.endTurn( );  
assert( g.turnsLeft( ) == 4 );  
assert( !g.isGameOver( ) );

std:: cout << G.getComputer().whatWasRolled(); // run at least twice to see if values are random

**TEST CASE 18** GAMEOUTCOME determineGameOutcome() const

**Reason: The function must output the correct game outcome**

BeatThat g;  
Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Die d3;  
d3.setValue( 3 );  
Die d4;  
d4.setValue( 4 );  
g.humanPlay( d1, d2 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
assert( g.turnsLeft( ) == 4 );  
assert( !g.isGameOver( ) );  
assert( g.determineGameOutcome( ) == cs31::BeatThat::GAMENOTOVER );  
assert( g.getHuman().getRoundsWon( ) == 0 );  
assert( g.getComputer().getRoundsWon( ) == 1);

------------------------------------------------------------------------------------------------------------------------------------------

BeatThat g;  
Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Die d3;  
d3.setValue( 3 );  
Die d4;  
d4.setValue( 4 );  
Die d5;  
d5.setValue( 5 );  
Die d6;  
d6.setValue( 6 );  
g.humanPlay( d1, d2 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d1, d2 );  
g.computerPlay( d2, d1 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d5, d3 );  
g.endTurn();  
g.humanPlay( d6, d6 );  
g.computerPlay( d6, d5 );  
g.endTurn( );  
assert( g.turnsLeft( ) == 0 );  
assert( g.isGameOver( ) );  
assert( g.determineGameOutcome( ) == cs31::BeatThat::HUMANWONGAME );  
assert( g.getHuman().getRoundsWon( ) == 2 );  
assert( g.getComputer().getRoundsWon( ) == 1);

------------------------------------------------------------------------------------------------------------------------------------------

BeatThat g;  
Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Die d3;  
d3.setValue( 3 );  
Die d4;  
d4.setValue( 4 );  
Die d5;  
d5.setValue( 5 );  
Die d6;  
d6.setValue( 6 );  
g.humanPlay( d1, d2 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d1, d2 );  
g.computerPlay( d2, d1 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d5, d3 );  
g.endTurn();  
g.humanPlay( d1, d6 );  
g.computerPlay( d6, d5 );  
g.endTurn( );  
assert( g.turnsLeft( ) == 0 );  
assert( g.isGameOver( ) );  
assert( g.determineGameOutcome( ) == cs31::BeatThat::COMPUTERWONGAME );  
assert( g.getHuman().getRoundsWon( ) == 1 );  
assert( g.getComputer().getRoundsWon( ) == 2);

------------------------------------------------------------------------------------------------------------------------------------------

BeatThat g;  
Die d1;  
d1.setValue( 1 );  
Die d2;  
d2.setValue( 2 );  
Die d3;  
d3.setValue( 3 );  
Die d4;  
d4.setValue( 4 );  
Die d5;  
d5.setValue( 5 );  
Die d6;  
d6.setValue( 6 );  
g.humanPlay( d1, d2 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d3, d4 );  
g.endTurn( );  
g.humanPlay( d1, d2 );  
g.computerPlay( d2, d1 );  
g.endTurn( );  
g.humanPlay( d3, d5 );  
g.computerPlay( d5, d3 );  
g.endTurn();  
g.humanPlay( d1, d6 );  
g.computerPlay( d6, d1 );  
g.endTurn( );  
assert( g.turnsLeft( ) == 0 );  
assert( g.isGameOver( ) );  
assert( g.determineGameOutcome( ) == cs31::BeatThat::TIEDGAME );  
assert( g.getHuman().getRoundsWon( ) == 1 );  
assert( g.getComputer().getRoundsWon( ) == 1);