SAIT:
Simultaneous Alternative Implementation Testing

Imagine a JUnit test that not only verified the correctness of one implementation but also of multiple alternatives simult

aneously.

Motivation

- Supplements the testing process
  - Keeps code clean
  - Tests bug origins easier
  - Allows for comparison of run time
  - Allows for comparison of consumed space
- Current alternatives are… what?
  Litter your method with commented blocks of code? nah.
  Version control? Not applicable.

```
            BestMove<M> curBestMove;
//
//          if(this.pCO == 1) { // if we have reached the plyCutoff
//              BestMove<M> antiAlpha = new BestMove<M>(this.alphaBeta[0].move, -this.alphaBeta[0].value);
//              BestMove<M> antiBeta = new BestMove<M>(this.alphaBeta[1].move, -this.alphaBeta[1].value);
//              curBestMove = AlphaBetaSearcher.alphaBeta(this.evaluator, this.board, this.depth-1, antiBeta, antiAlpha);
//          } else {
//              //goes to base case C:
//              JamboreeTask<M,B> newThread = new JamboreeTask<M,B>(this.evaluator, this.board, moves, new int[] {i,i}, this.dCO, this.pCO, this.depth, this.alphaBeta, false);
//              curBestMove = newThread.compute();
//          }

            board.undoMove();

            curBestMove.value *= -1; // this flips the value because the current player wants to do the opposite of the opponent
            if(curBestMove.value > this.alphaBeta[0].value) {
                this.alphaBeta[0] = new BestMove<M>(move, curBestMove.value);
            }

            if(this.alphaBeta[0].value >= this.alphaBeta[1].value) {
                return this.alphaBeta[0];
            }
        }

//      //parallelize second half
//      int[] midHi = {midpoint, hi};
//      JamboreeTask<M,B> jimjam = new JamboreeTask<M,B>(this.evaluator, this.board, moves, midHi, this.dCO, this.pCO, this.depth, this.alphaBeta, false);
//      BestMove<M> parallel = jimjam.compute();

        return this.alphaBeta[0].value < parallel.value ? parallel : this.alphaBeta[0];
    } else {
//      if (this.hi - this.lo <= this.dCO) {
//          if (this.pCO == 1) {
//              for (int i = this.lo; i < this.hi; i++) {
//                  M move = this.moves.get(i);
//                  this.board.applyMove(move);
//
//                  BestMove<M> antiAlpha = new BestMove<M>(this.alphaBeta[0].move, -this.alphaBeta[0].value);
//                  BestMove<M> antiBeta = new BestMove<M>(this.alphaBeta[1].move, -this.alphaBeta[1].value);
//
//                  BestMove<M> curBestMove = AlphaBetaSearcher.alphaBeta(this.evaluator, this.board, this.depth-1, antiBeta, antiAlpha);
//                  board.undoMove();
//
//                  curBestMove.value *= -1; // this flips the value because the current player wants to do the opposite of the opponent
//                  if(curBestMove.value > this.alphaBeta[0].value) {
//                      this.alphaBeta[0] = new BestMove<M>(move, curBestMove.value);
//                  }
//
//                  if(this.alphaBeta[0].value >= this.alphaBeta[1].value) {
//                      return this.alphaBeta[0];
//                  }
//              }

            return this.alphaBeta[0];
        }
```

Approach

High level? Implement a branch of JUnit tests

>Include additional file (java? txt?) that contains alternate methods

>Indicate above tests the use of alternate methods:

```
23
24      ////////////////////////////////////////////////////////////////////////////////
25      ////Constructor
26      ////////////////////////////////////////////////////////////////////////////////
27      @Test
28      @Alternate
29      public void testConstructor() {
30          DLUMGraph<String, String> empty = new DLUMGraph<String, String>();
31          assertEquals(empty.toString(), "Vertices: 0, Edges: 0");
32          assertEquals(empty.hashCode(), 0);
33      }
```

>Console will print out additional info regarding comparison of run

IE: Exceptions thrown, run time, spatial usage, etc.

Limitations:

Only functions as an extension of JUnit (or in tandem)

Hard to infer alternative implementations in cases where methods use multiple classes or other methods

Challenges and Risks

- Requires learning and understanding JUnit
  - JUnit might not be possible to extend in the way we want
    -> if so, this project requires learning how to set up something like JUnit or figuring out how to modify a java object to replace methods with alternate