Homework 5

Checkers

August 2, 2021

Abstract

The objective is to give you practice constructing and destructing instances of classes.

The context of this problem is the game of checkers. In checkers, there are two kinds of pieces, pawns and kings; there are rules about how pawns can move, and how kings can move. When a pawn has advanced to the furthest row, it ceases to be a pawn, and is replaced by a king. Rules for playing can be viewed at `https://www.thesprucecrafts.com/play-checkers-using-standard-rules-409287`. The program must, given an arrangement of pieces on the board, figure out the possible moves, and (possibly randomly) choose one of the possible moves, generating a new board configuration, unless there are no possible moves. If there are no possible moves, this must be reported. The program must choose a move. As a command line argument, the program must accept its status as first player or not. The program that is designated a first player must generate a (legal) move. Then the program must receive as input, the opponent's move. Early in the development phase, a person may invent the opponent's move. Two person teams of students will independently develop a checkers program, and eventually will compete the two programs against each other, by entering the move chosen by one program into the other program. Recorded zoom sessions, or email trails, can be submitted for the assignment, along with the usual source code submission. The game should proceed for a given number of moves, unless a no move situation occurs. If a pawn reaches the furthest row, it must be crowned a king; after this the greater number of possible moves to which it is entitled must be considered. If a pawn is captured, its instance must be destructed at the time of capture.

Note that the king can do everything a pawn can do. If we needed to substitute a king in, to carry out the role of a pawn, the king would be able to do so. Using the Liskov Substitutability Principle, we see that the king is-a-kind-of pawn (with additional powers).

Be sure to construct test cases for each function you build. Put thought into the test cases. The goal is to know that a function works, when it passes all of its test cases. One test case per function is usually not enough.

- Construct a sequence diagram for your project. You can draw it by hand and include a phone/photo, or draw it with a software tool.

- Use the test-driven development style for developing your code. Document this by taking before and after screen shots, as detailed in HW3. Do not allow the number of errors to get large.
- The program must display the configuration of the board after each move.
- The program must record the number of possible moves, the chosen move, and successive board configurations for each turn in a log file. You can choose the file format, but you must document that, and adhere to what you describe.

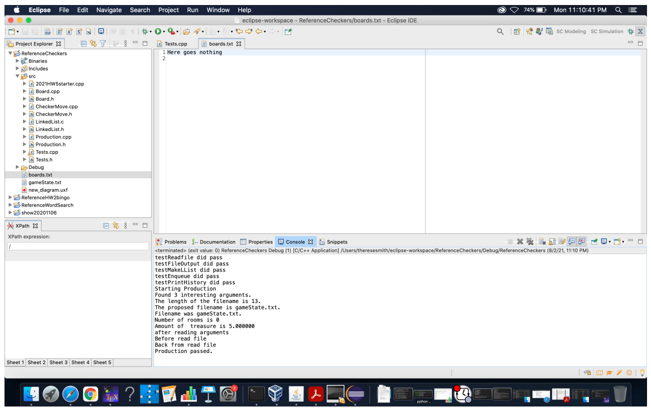
Things to do:

1. Either:
   (a) Make a C++ project from the Hello,World project.
   (b) Populate that project with at least one class for Test, and at least one class for Production, as well as at least one include file for tests and at least one include file for production.

   or use the starter code.
2. Create the sequence diagram and include the electronic file (diagram, screenshot or photo). Make sure your name appears within the sequence diagram.
3. Place function prototypes for all of your functions from the sequence diagram into several .h files.
4. As you work on the assignment, collect a sequence of before and after screen shots.
5. Be sure to build and run often; do not allow errors to build up.
6. Show the sequence of game board configurations, and the number of possible moves from each configuration.
7. Receive the values entered on the command line. Ask for values not provided in the command line. In either case, check for reasonable values. Negative number of turns implies no game should be conducted. Use these limit values in execution.

Grading

Criteria	Possible Points
Project that looks like starter code	20
Sequence diagram that reflects the problem statement	20
Documentation of code development (those screenshots) that clearly follows test-driven style	20
File of possible moves identified, chosen move, and resulting board configuration	30
Correct treatment of command line arguments	10
Total	100



```
Here goes nothing
```

```
<terminated> (exit value: 0) ReferenceCheckers Debug (1) [C/C++ Application] /Users/theresesmith/eclipse-workspace/ReferenceCheckers/Debug/ReferenceCheckers (8/2/21, 11:10 PM)
testReadfile did pass
testFileOutput did pass
testMakeLList did pass
testEnqueue did pass
testPrintHistory did pass
Starting Production
Found 3 interesting arguments.
The length of the filename is 13.
The proposed filename is gameState.txt.
Filename was gameState.txt.
Number of rooms is 0
Amount of  treasure is 5.000000
after reading arguments
Before read file
Back from read file
Production passed.
```

Figure 1: This is the starter code running. Note that the file boards.txt has appeared in the ProjectExplorer. That was written by the code. The file did not exist before the code ran.

Task / Level	full score	half score	quarter score
Use starter code	Do not change the test/production paradigm	some change	omit tests
Provide sequence diagram	identify team members, communications that solve the problem	team members or communications	neither
Provide before-and-after screenshots	show production code before and after with test results failing and passing respectively	showing test rather than production	not showing both of before and after
Screenshot results in console	correct answer	partially correct answer	no answer
Output file	correct	code to write, but no file	less
Command line arguments	obtain and use	obtain	less