SEG2106 Assignment 4 Report

Owen Daigle - 300359036

1 My Solution

My solution used the java implementation of threads. I used the interface `Runnable` for the `HillClimbingSearch` class. This required me to implement the `run()` method which would simply run the `runSearch()` method.

Then once `runSearch()` would terminate, the `run()` method would print out the result. However it can only print the result if no other threads have already printed the result. This check is done with an atomic boolean variable `found`. Before printing out the output the code will compare the value of `found` and only if it is false will it print the solution and then set `found` to true.

The only other thing I had to change was to handle the `Thread.currentThread().isInterrupted()` method. This method returns true if and only if the current thread has been interrupted. If the current thread is interrupted, I can stop calculating the board, so I put this check into the loop in the `runSearch()` method.

In the `Main` file, the only thing I really changed was made it so when starting the search, it iterates through a for loop up to the number of threads I want to start, and each time starts a new thread in a thread group `threadGroup`.

Then the main program waits on hold using a while loop that checks if the number of threads we started equals the number of active threads. Once at least one thread terminates the main program continues and then sends an interrupt to the `threadGroup` group. This interrupts all threads.

The only other modification to the code that I did was to allow the program to also get the number of queens from the console argumets `args[0]`. This is so I could automate some performance testing using scripts.

2 Performance Comparison

I created a simple `bash` script called `run.sh` which compiles all the single and multi threaded code, and then runs the java program `NUMBEROFRUNS` times for `NUMBEROFQUEENS` queens (both variables found in `run.sh`) firstly using the single threaded program, and next using the multi threaded programs using the number of threads defined in the `Main.java` file in the `THREAD_COUNT` variable. It then outputs a simple summary of the results.

All tests were run on a machine with a CPU with 16 threads. Some values were ommited from this report to save space, the entire results can be found in the file `tests.md`

Using **45 queens, with 40 runs and 16 threads** we get the following results which show as expected a large performance increase from using multi threaded. It is not 16 times as fast though, which could be due to the `HillClimbingSearch` starting at a random point, and some threads start at similar points and therefore both of their work overlaps.

```
For Single Threaded, average is: 8121
Individual times are: 1603 672 16158 10532 ...[other results omitted]
```

```
For Multi Threaded, average is: 1065
Individual times are: 682 169 946 1062 ...[other results omitted]
```

Reducing the number of queens to be found makes using multiple threads pointless. In fact it is less efficient to use multiple threads due to the extra complexity added to the code with very low number of queens such as **6 queens, with 40 runs, and 16 threads**.

```
For Single Threaded, average is: 8
Individual times are: 10 9 9 8 8 7 10 ...[other results omitted]
```

```
For Multi Threaded, average is: 14
Individual times are: 12 19 19 13 14 12 19 ...[other results omitted]
```

If we reduce the number of threads for the multi thread code, as expected the time it takes will increase. Using 45 queens with 40 runs and 4 threads we get:

```
For Single Threaded, average is: 9340
Individual times are: 8319 3389 3527 5561 ...[other results omitted]
```

```
For Multi Threaded, average is: 2119
Individual times are: 165 724 164 1089 ...[other results omitted]
```

If we increase the number of running threads past the number of physical threads on my machine, as expected the performace does not increase. Here I omitted the single threaded results in order to save time as it is unimportant for these tests.

Using **16 threads**:

```
For Multi Threaded, average is: 1136
Individual times are: 1157 3091 2918 454 ...[other results omitted]
```

Using **64 threads**:

```
For Multi Threaded, average is: 1280
Individual times are: 1257 464 1531 1858 ...[other results omitted]
```