Maze Solver

StudentMTMazeSolver.java

The StudentMTMazeSolver class implements a multi threaded DFS algorithm to search for the end of the maze from the maze start.

The class conatins an inner class ***DFSTask*** which extends **RecursiveAction** from *java.util.concurrent.RecursiveAction* which results in the class becoming a task which can be executed in the fork join pool. For larger mazes a higher level of parllaisation is used.

Logic

```
class StudentMTMazeSolver extends SkippingMazeSolver
{
    //constructor
    StudentMTMazeSolver(Maze maze)
    {
        1. execute  super(maze) to initilise maze
        2. check if maze is large or not
    }

    public List<Direction> solve()
    {
        initialize the fork join pool
        if(maze large)
        {
            for(every choice in first cell)
            {
                create DFSTask by following the choice
                execute the DFSTask *asynchronosly* on forkjoin pool
            }
            join all executing tasks
            shutdown the fork join pool
        }
        else
        {
            for(every choice in first cell)
            {
                create DFSTask by following the choice
                execute the DFSTask on forkjoin pool
                join the DFSTask
            }
        }

        print the total number of choices made
        if maze display is on mark the path
        return the solution
    }
```

```
    private class DFSTask extends RecursiveAction
    {
        //constructor
        public DFSTask(Choice rootChoice, Direction comingFrom)
        {
            rootchoice is the choice(node) the current task will start from
            comingFrom is the direction from which the current task was
invoked
        }
        public void compute ()
        {
            initialise choiceStack
            try
            {
                pushRootChoice on top of the choice stack
                while(choice stack is not empty)
                {
                    if(choice on top oof the stack is a deadend)
                    {
                        pop the top choice
                        pop the direction which lead to the deadend choice
from the current top elemnt of the choice stack
                        continue;
                    }
                    push the next choice from the first direction on top of
the choiceStack
                }
            }
            catch(SolutionFound s)
            {
                initilise solutionPath list
                itterate throuch choiceStack{
                    push the last direction followed by every choice to
solutionPath
                }
                set the solution list in outer StudentMTMazeSolver to
solution path
            }
        }
    }
}
```