Worksheet 5

COMP110: Principles of Computing

Ed Powley

February 2016

Introduction

In this assignment, you will implement the **A* pathfinding algorithm** within a provided template application.

This worksheet tests your ability to translate a complex algorithm, given as pseudocode, into C++ code, with appropriate choices of data structures.

Submission instructions

The GitHub repository at the following URL contains the skeleton project for this worksheet.

`https://github.com/Falmouth-Games-Academy/comp110-worksheet-5`

Fork this repository into your own GitHub account. To submit, create a GitHub pull request.

Please do not rename or delete the project or files provided, and please do not create new projects. This will ensure that GitHub's "diff" view highlights only the parts of the code that you have edited. Creating new source and/or header files is permitted, if doing so improves the structure of your programs.

Marking

Timely submission: 40%

The deadline for this worksheet is **6pm, Wednesday 9th March 2016**. To obtain the marks for timely submission, you must submit (as a GitHub pull request) your progress towards the worksheet by this time. As with other worksheets, you may resubmit after these deadlines in order to collect extra correctness or quality marks. This 40% is awarded as long as you submit *something* by the deadline, even if your submission has bugs or other issues.

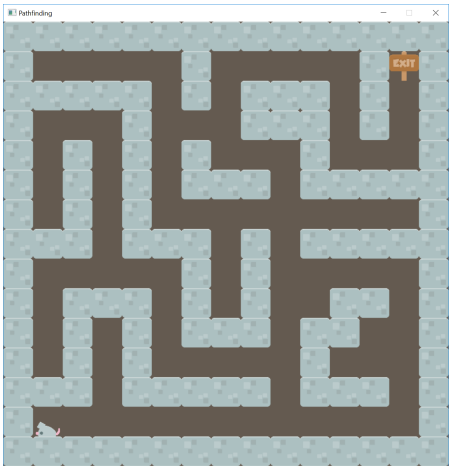

Figure 1: A screenshot of the skeleton project.

Correctness: 30%

To obtain the marks for correctness, you must submit a working solution. Note that this is a threshold: the full 30% is awarded for work which is *complete* and contains *no clear errors*. In particular you will not be penalised for trivial errors which do not affect the overall functioning of your program, nor will you receive extra credit for a highly polished solution.

Quality: 30%

The extra quality criteria for this worksheet are as follows. All are weighted equally, so each is worth 6% of the overall mark for this worksheet, with the exception of the **stretch goal** which is worth 12%.

1. **Presentation.** Your solutions are appropriately presented in GitHub, with descriptive commit messages and appropriate documentation in `readme.md` files. You have edited the provided skeleton projects, and refrained from renaming the provided files or creating new projects. You have checked code into GitHub regularly whilst working on the worksheet.

2. **Code quality.** Your code is well formatted. Variable and function names are clear and descriptive. Comments are used where appropriate, and are well written. Your formatting, naming and commenting follow consistent conventions.

3. **Choice of data structures.** You have chosen appropriate data structures for your implementation, and have justified your choices in the `readme.md` file.

4. **Stretch goal (weighted double).** You have submitted a working solution for the Stretch Goal task.

The task

The skeleton project provides code to load a simple maze from a text file, and display it on screen using SDL (Figure 1). **Implement** the function `Pathfinder::findPath` in `Pathfinder.cpp` to find the shortest path from the start (the mouse) to the goal (the exit sign). Base your implementation on the pseudocode for the A* algorithm provided in Algorithm 1. The skeleton project is already set up to call `Pathfinder::findPath` and display the result on screen.

A selection of test maps is provided in the `Maps` directory, and can be chosen between by changing the constant definition near the top of `PathfindingApp.cpp`. Make sure that your implementation works for all the provided maps.

You should pay particular attention to your choice of **data structures**. The C++ Standard Template Library (STL) provides many standard data structures such as vectors, linked lists, sets, maps, queues, priority queues and stacks. Research these, consider which operations are most frequent on the collections of nodes used by the algorithm, and choose appropriately. **Briefly** justify your choices of data structures in your `readme.md` file.

You are strongly encouraged to make use of **object oriented programming** concepts for this task. At the very least, you will probably want to define your own `Node` class. Add your own fields and methods to the `Pathfinder` class to improve the readability of your `findPath` implementation.

Stretch goal

Adapt your program to visualise the operation of the A* algorithm. At each step of the algorithm, your program should display at least:

- The **current node**;
- The **partial path** from the start node to the current node;
- Which nodes are in the **open set**;
- Which nodes are in the **closed set**.

If you attempt this part, do so in a new **branch** within your GitHub repository: your basic A* implementation should remain in the `master` branch, and your visualisation should be in the new branch.

Algorithm 1 The A* algorithm

```
1: procedure A*(startNode, goalNode)
2:     closedSet ← {}
3:     openSet ← {startNode}
4:     startNode.g ← 0
5:     startNode.h ← EuclideanDistance(startNode, goalNode)
6:     startNode.cameFrom ← null

7:     while openSet is not empty do
8:         currentNode ← node in openSet with lowest g + h score
9:         if currentNode = goalNode then
10:            return ReconstructPath(goalNode)
11:        end if

12:        remove currentNode from openSet
13:        add currentNode to closedSet

14:        for each neighbourNode adjacent to currentNode do
15:            if neighbourNode is not a wall and neighbourNode not in closedSet then
16:                g_tentative ← currentNode.g + EuclideanDistance(currentNode, neighbourNode)
17:                if neighbourNode not in openSet or g_tentative < neighbourNode.g then
18:                    neighbourNode.g = g_tentative
19:                    neighbourNode.h ← EuclideanDistance(neighbourNode, goalNode)
20:                    neighbourNode.cameFrom ← currentNode
21:                    add neighbourNode to openSet (if it is not already in)
22:                end if
23:            end if
24:        end for
25:    end while

26:    return "Failed to find a path"
27: end procedure

28: procedure ReconstructPath(goalNode)
29:    path ← {}
30:    currentNode ← goalNode
31:    while currentNode ≠ null do
32:        add currentNode to the beginning of path
33:        currentNode ← currentNode.cameFrom
34:    end while
35:    return path
36: end procedure
```