CO2301: Games Development 1: Assignment - Path Finding

Deadline: Thursday 25th January 2018, 8.30am

Coursework counts 70% to the overall marks for CO2301, with the exam counting 30%.

Introduction

This assignment requires you to implement one or more path finding algorithms, with the option of providing a visual representation. There are two different routes permitted:

Route A. This route asks you to implement the **breadth-first** and **depth-first** algorithms.

- If you choose this route **You cannot gain more than a pass mark of 40%**.
- If you follow this route, then you do not need to implement a visual implementation.
- Essentially I am asking you to complete Exercise 6 and 7 from the Worksheet.

Route B. This route asks you to implement the **A* path finding** algorithm.

- Your implementation of the algorithm does not have to be perfect in order to *pass* the assignment. It can be possible to pass the assignment with errors.
- Marks will be lost for:
  - significant errors
  - inefficient implementation of the algorithm

This is an **individual** project and group work is not permitted.

Deliverables, Submission and Assessment

- Upload the following files to Blackboard by the deadline:
  - The **C++ source**(s) for your solution
  - Your **text output** files for each algorithm
  - Any **model or texture files** you use which are NOT supplied with the TL-Engine
  - **IF** you have implemented a graphical solution, also submit:
    - A word file containing labelled **screen shots** for each completed map.
  - **IF** you have implemented a **real-time** visual representation
    - A link to a video of your solution in action (e.g. on YouTube)
- Make sure your **name** is included as a comment at the top of EVERY source code file.
- All files should be sensibly named, and all code should compile and run without errors.

- **Assessment will be by demonstration in your lab on Thursday 25th January 2017.**

- Unless you have an extension (agreed in advance) or extenuating circumstances, then your work will be treated as **late** if you are not prepared to demonstrate in the lab on the 19th January. Late work will be capped with a maximum mark of 40%.
  - If you have a problem, then you need to formally request an extension or submit extenuating circumstances. We'll agree a later date for you to submit and demonstrate your work.
- If your assignment is late then you must show it to me in the next lab.

Regulations

The coursework regulations can be found on Blackboard in the “Computing Noticeboard”.
Pay particular attention to the regulations about **Plagiarism:** Discussing the assignment and helping one another is **good**. Copying someone’s work and submitting it as your own is **bad**.

Assignment Details

File Input and Output

- All files should be read relative to the current working directory.
- Two associated sets of map files have been provided:
    - **mMap.txt** and **dMap.txt**
    - You must first run the search for mMap and then for dMap.
    - The maps are both 10 by 10 in size.

- Your program must read the start and goal coordinates from the files provided:
    - **mCoords.txt** (for use with mMap) and **dCoords.txt** (for use with dMap).
    - The coordinates file contains two lines of text.
        - The first line of text gives the coordinates of the start location: **x** followed by **y**, with a space in between.
        - The second line of text gives the coordinates of the goal location: **x** followed by **y**, with a space in between.

- Your program should work for **ANY** map (following the naming convention *x*Map.txt etc)
    - Your program should **NOT NEED TO BE RECOMPILED** to load different maps!

- The results of the search should be written to an output file named **output.txt**.
    - This file should **only** contain the route of the selected paths.
    - The route should be presented as a list of all of the coordinates of the route in order.
        - It is *easier* to print out the route in reverse order from goal to start. However, you will obtain **better marks** if you print out the route from the start position to the goal position including all of the locations in between.

- If you implement the A* algorithm, you must **keep a count of how many times the open list gets sorted**, and output the total somewhere (to console or file) at the end of the search.

Coordinates and Movement

- You should use a square grid.
    - Movement is only along the horizontal and vertical – not diagonal
        - i.e. North, South, East and West.
    - Each grid reference is identified by an x and y co-ordinate.
    - The x axis is positive to the right and the y axis is positive upwards.
    - The coordinate numbering starts at 0. (see the figure overleaf)
- **You will lose marks** if you do not adhere to the coordinate system provided.

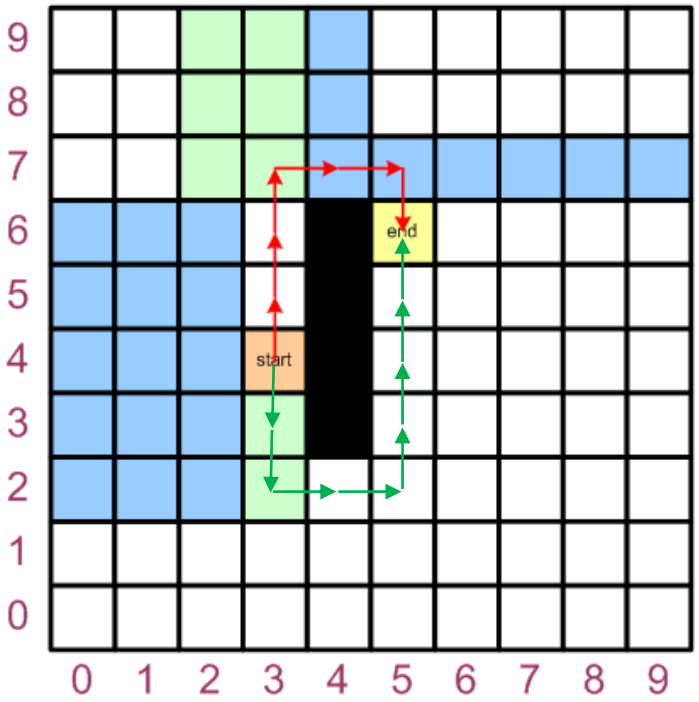
- For the purposes of this assignment, the heuristic is simply the Manhattan distance to the goal.

User Interaction

- I do not want to have to keep pressing keys in order to advance the search.
    - **You will lose marks** if you require unnecessary user input.
- After the search begins, the program should produce the final route without requiring any further user input.
    - The only exception to this is that **if** a visual representation of the search is provided, then you *may* want pause execution to show the search expanding.
    - You are not required to, but if you do then do it sensibly – i.e. pause the tree ply-by-ply, not node-by-node.

Example Maps and expected routes

mMap (*above*) and dMap (*below*)

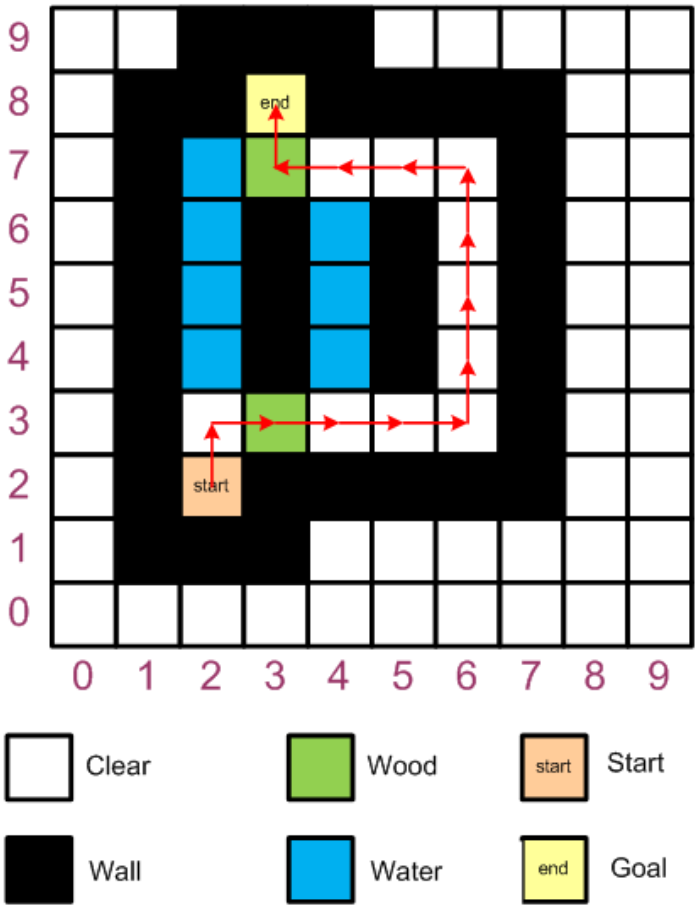


Movement costs
All of the algorithms use a map with a wall.

A* requires a map with terrain costs as follows:

- Clear 1
- Wood 2
- Water 3
- Wall 0

Therefore, the A* algorithm will choose the path shown in green arrows that goes south of the wall on the mMap (top map) rather than the shortest route shown in red arrows (which would be found by e.g. breadth first).



Path Finding Algorithm Implementation

- You must implement the **algorithm as provided** on the module page.
- You must use **STL container classes** to implement the algorithm.
- You must implement the assignment in **C++** using **smart pointers**.
- You will need to implement **map reads from file**.

Route A: *Breadth-first* and *depth-first algorithms for a maximum mark of 40%*

- Do not attempt this section if you want to attempt the full range of marks.
    - The task is basically to complete Exercises 6 and 7 from the lab worksheet.
- Implement a breadth-first search.
- Implement a depth-first search.
- You must include backtracking and avoid repetition. Use the Manhattan distance between the goal and the current location for your heuristic.
- You are being graded on:
    - Correct implementation of the algorithm.
    - Use of object-orientation
    - Use of STL
    - Map read.
    - Code quality, e.g. meaningful names, well commented and good style.

Route B: *A* algorithm for a mark potentially above 40%*

- Implement an A* search. You must include backtracking and avoid repetition. Use the Manhattan distance between the goal and the current location for your heuristic.
- Test your algorithm with other maps. I reserve the right to use a selection of maps to mark your work.

Code Style and Layout

- Your code needs to adhere to Gareth Bellaby's style guide.
- The style guide can be found inside the Advanced C++ module page on Blackboard.

Visual representation using the "output.txt"

- Use the **TL-Engine** to implement a visual representation of the searches. You will not be graded on the underlying graphics code but the clarity of the visual representation and successful execution of the program.
- Your program writes its results to a file called "output.txt". Use this file along with the map file to produce a visual representation of the search.
- In other words, the search algorithm should run in its entirety. After the algorithm has run, use the output to produce a visual representation of the progress of the search.

Advanced: real-time visual representation

- Implement a real-time visual representation of the algorithms.
- The visual representation will be updated as the search progresses.
- Note this is much more difficult than using the "output.txt".

More Advanced: spline applied to movement on map

- This must be done in conjunction with implementing a real-time visual representation of the algorithms.
- Move an actual model over the map following your generated route.
    - For better marks, the model should turn and face the direction of travel.
- Smooth the chosen path using a spline. A Bézier spline will probably produce the best effect.