First Assignment (Group Project)
AI Programming for Games
COMP09041

Issue Date: Tuesday, 11th February, 2025
Due Date: **5pm, Monday, March 3rd, 2025**

A* Game: Pathfinder

In this assignment you are provided with an interactive C++ program which builds on the pathfinding topic, involving Red Blob Games' `Graph` class and A* pathfinder; as well as the raylib videogame framework.

You are tasked with creating a simple 2D pathfinding game. When complete, the screen will display a weighted graph; and a start & end node. The player must click the nodes in sequence from the start to the end, while minimising traversal costs. There is also a time limit. The score is increased by clicking on each level's final node. The reward is the sum of the edge costs on the ideal path from start to end; as calculated by the Red Blob Games' A* algorithm (`astar_pathfind`).

This is a group project. Your team allocations are provided to you on Aula.

You should modify the provided program in a number of specific ways, and these should each be explained in a short accompanying report. Your zipped submission (just one file) should include your report along with the modified source code. Only one team member should submit.

Background

You are provided with a graph, with nodes labelled from 'A' to 'G'. The graph class `Graph` is defined in graph.hpp, and adopts the simple Red Blob Games interface requirement of two member functions called `neighbors` and `cost`.

The main data member of the `Graph` class is an `std::unordered_map` called `nodes`, which is basically a *hash map*. A hash map can be used like an array, except the index type will vary; in this case, we are using a `char` as the index (type aliased as `node_t`), and the value returned is a `std::vector<char>`, which provides the *neighbours* to the current node.

This is a simple but effective representation of a graph. Two things are missing though: information on each node's spatial coordinates; and edge costs. These are provided by two global variables defined in graph.hpp as shown below:

```
using node_t = char;
using edge_t = std::pair<node_t, node_t>;
using coord_t = ai::Vector2;
std::unordered_map<node_t, coord_t> node_info;
std::unordered_map<edge_t, double> edge_info;
```

The `node_info` object provides additional information about each *node* in the graph: the node's coordinates as a `coord_t`. So, `node_info['A'].x` would provide the *x coordinate* of a node called 'A'. Meanwhile, `edge_info` provides similar auxiliary information about each *edge* in a graph. An edge is defined as a *pair* of nodes. As the graph is directed, there may be one edge from, say, 'B' to 'C'; and another edge back from 'C' to 'B'. The `std::pair` class template is then used to enquire from `edge_info` on the *cost* of traversing an edge; so `edge_info[std::pair<node_t>('B','C')]` would return the `double` value corresponding to the travel cost from 'B' to 'C'.

```
void add_node(Graph& g, const node_t& n, const coord_t& xy)
void add_double_edge(Graph& g, const node_t& n1, const node_t& n2);
```

The function declarations for `add_node` and `add_double_edge` are defined in graph-util.hpp, and shown above. The `add_node` function will both add a node `n` to the `Graph g`; as well as associate it with the position encoded in the value held in `xy`. Meanwhile, the `add_double_edge` function will add *two* edges to the graph, between nodes `n1` and `n2`; *in both directions*. The *cost* associated with each direction is simply the distance between the two nodes.

raylib Audio

One task below asks you to play a sound. To play a sound, first call the `InitAudioDevice` function *once*, at the start of the program. At the same place in the code, create a few `Sound` objects using the `LoadSound` function. Audio files can be found in the resources directory provided within raylib-cpp. Given that you will be working on different machines, you will find it convenient to use *relative* paths such as "../../deps/raylib-cpp/examples/audio/resources/coin.wav".

Assignment Brief

Attempt the following 15 tasks. In addition, a 1000 word report should be provided in pdf format. The report should start by briefly introducing the context of the assignment, before describing the approach taken for each of the completed tasks. You are encouraged to include figures; which might include screenshots, or short code excerps (up to 5 lines for each one). Include a conclusion.

1. Display the `score`, `tokens`, `high_score`, and `t` (time) on the screen using raylib's text rendering functions; such as `DrawText` and `TextFormat`. **(1 point)**
2. Highlight the start node in green, and the end node in red. **(2 points)**
3. Add a node to the *player path* (`player_path`) by clicking on it with the mouse button. **(1 points)**
4. Add a sound effect when a node is added to the player path. **(1 points)**
5. Highlight the player path; which is built up by each node added. **(2 points)**
6. Ensure the first node added by a click is a neighbour of the start node. **(1 points)**
7. Ensure you can only add a node which shares a connection/edge with the previous one. **(2 points)**
8. Adding a connection costs tokens. Remove (from `tokens`) the cost of each new connection added to the player path. **(2 points)**
9. Allow the user to remove a node (and have the token cost reimbursed) from the player path, by clicking with the mouse on the last node added. **(2 points)**
10. When the last node (i.e the red node) is clicked, update the player score by the cost of the "ideal" path; calculated using `astar_pathfind` and `path_cost`. Also award some more tokens; clear the player path; and start again. **(2 points)**
11. Use euclidean distance rather than the manhattan distance for the heuristic in graph.hpp. **(1 points)**
12. Update the timer each second. It should count down from 60 to 0. **(2 points)**
13. The game is over if either: a) the timer reaches zero; or b) clicking the last node brings the token count below 0. When the game is over, update the high score, reset the score, and restart the level. **(1 points)**
14. Rather than simply restart the level (on game over or last node selection), *randomly* choose a new start node and end node. **(2 points)**
15. Assign a mark for each team mates' efforts, along with a sentence supporting each mark. You will receive the average of the marks awarded to you. A separate Aula assignment for this is provided. **(3 points)**

Resources

As well as the main C++ file src/pathfinder.cpp, the implementation.hpp header file from Red Blob Games is also included, along with two header files which help with the non-grid graph that we need (graph.hpp and graph-util.hpp). The usual Raylib C++ library, and UWS `raylib-extras` directory, are also included. Note that the `ai::Vector2` class defined in vec.hpp includes support for equality comparisons, and for streaming to standard output via `std::cout`. Audio resource files from Raylib are included with raylib-cpp. Use CMake, Visual Studio 2022, and VCPKG as usual.

If you think the graph provided is too simple, feel free to add more nodes to it. So too, feel free to change the background colour; use (load) another font; or use different colours.

The assignment is worth 30% of the marks awarded for the entire COMP09041 module. The following provides a summary breakdown of the marking scheme:

1000 word report with figures	5
1. Display score & other game stats	1
2. Highlight start & end nodes (e.g. green & red)	2
3. Add nodes by clicking	1
4. Play a sound when a new node is added	1
5. Highlight the path selected by the player	2
6. Ensure the first node added is a neighbour of the start node	1
7. Ensure each node added shares a connection with its predecessor	2
8. Update the token count with each connection added	2
9. Allow the player to "undo" by clicking the same node again	2
10. Update score & restart level when the end node is reached	2
11. Modify the heuristic function used by A* (to use *distance*)	1
12. Update the countdown timer each second (from 60 to 0)	2
13. When the game is over, update the scores & start again	1
14. Select a random start & end node rather than 'A' & 'G'	2
15. Average of your peer review marks from your team	3

Plagiarism

Ensure your work is developed only by your own team. You can discuss ideas with other teams, regarding how to prepare a solution, but the *copying or sharing of code is not permitted*.

Anonymity

Please use only the Banner IDs of your team members to identify yourselves in your submission. Ensure the Banner IDs of all team members are on the first page of your report.