CS302

Assignment 8: Save Ashley

Description

Resident Evil 4. An action horror game where you fight off zombies (well they're not called zombies in RE4 but basically they act like zombies so they're basically zombies) and you constantly lose Ashley and have to find her over and over throughout the game. At the beginning of the game you start out in the village (and of course you shoot the church bell, to get the zombies to stop attacking) and then you must make your way to church as Leon and find Ashley. There are multiple locations along the way to the church and you need to fight off zombies. With Leon's perfect aim, you fire a shot at a zombie's head to daze him/her in order to move past the zombie since you need to conserve ammo. Thus at each location, you need at least as much ammo as there are zombies since you cannot risk getting defeated and leave Ashley all alone. The objective is to find a path such that Leon can take to get to Ashley at the church without getting hit (so use your ammo wisely).

Your program is going to be a modified version of DFS on a graph. Each node in the graph is a location in resident evil 4 and an edge connects two locations, the locations are listed below:

- Village (Starting Location)
- Lake
- Castle
- GrandHall
- Clocktower
- Mines
- Island
- CargoDepot
- Ballroom
- Church (Ending location)

Each location has a set of zombies that Leon needs to fight through. For each input case, Leon starts at the Village (there will be 0 enemies at the village) and needs to find a path to the Church. The amount of initial ammo for Leon will always be 10, and at each node/location ammo is used up, if at a location Leon does not have enough ammo, Leon needs to reload the game back at the previous location and try a different

edge, however Leon can revisit a node multiple times if a different path to a certain reaches a better result (a path is found to this node where Leon has more ammo).

Data Structures

You will need an adjacency list, since each node is labeled by a string, a hash map will be used, the structure is declared below.

```
std::unordered_map< std::string, std::list<std::string> > adjList;
```

This maps a location to a linked list of its neighbor locations. You will also need a map to track the amount of enemies at a location

```
std::unordered_map< std::string, int > enemiesAtLocation;
```

You also need a "visited" array, this map will be used to store the amount of ammo Leon has when he arrives at a location, if another path in the future is found to this location but Leon has less ammo in this time, then you do not revisit the node, only revisit if the amount of ammo Leon has is more than the last time he visited this node, the structure is defined below

```
std::unordered_map< std::string, int > ammoAtLocation;
```

And of course in order to output the path Leon takes, you will need a predecessor array

```
std::unordered_map< std::string, std::string > predecessor;
```

Input

You will be given a map file and an enemies at a location file. In the maps file, each line contains a "from" location to a "to" location separated by a whitespace and each line is terminated with an end of line character, thus you populate the adjacency list by adding a node into "from"'s neighbors linked list using the code below

```
std::string from, to;

ifstreamVar >> from >> to;

adjList[from].push_back(to);
```

You can iterate through all the neighbors of any location using the following techniques

```
for (auto neighbor = adjList[location].begin();
    neighbor != adjList[location].end(); neighbor++)
{
  *neighbor; //retrieves the neighbor node name
}

for (auto neighbor : adjList[location])
{
  neighbor; //retrieves the neighbor node name
}
```

The enemies file that contains the amount of enemies at a location contains a location name and an integer separated by a whitespace and each line is terminated with an end of line character, you insert each amount in the enemiesAtLocation map using the location name as the key and the enemy amount as the value.

Main

You will need to implement the following DFS type recursive function defined below

```
bool saveAshley(std::string leon, int ammo,
  std::unordered_map< std::string, std::list<std::string> > map,
  std::unordered_map<std::string, int> enemiesAtLocation,
  std::unordered_map<std::string, int>& ammoAtLocation,
  std::unordered_map<std::string, std::string>& predecessor);
```

Each parameter is used for the following

- `std::string leon` - denotes Leon's current location
- `int ammo` - the amount of ammo when Leon arrives at the current location
- `std::unordered_map< std::string, std::list<std::string> > map` - adjacency list
- `std::unordered_map<std::string, int> enemiesAtLocation` - a map that maps a location to the amount of enemies at the location
- `std::unordered_map<std::string, int>& ammoAtLocation` - maps a location to the max amount of ammo when Leon arrived at any location at any point in time
- `std::unordered_map<std::string, std::string>& predecessor` - predecessor array

This function is called recursively until `leon == "Church"` which implies a path is then found (you would return `true` in this case), the `ammoAtLocation` is your "visited" array but this is not a boolean array/map thus a node can be visited multiple times if any later visits occur that results in more ammo at arrival. A `false` is returned whenever you reach a dead end or no paths from the current node lead to a success etc. Initially when the function is called in main, you pass `"Village"` to the `leon` parameter, `10` into the `ammo` parameter and then the rest of the maps you would declare in main. Once the function is finished, you use the `predecessor` array to construct the path (you can use a `std::vector<std::string>` object to perform `push_back` using the elements stored in `predecessor` array and then traverse the vector from the last element to the first element when outputting the path).

Specifications

- You must a write a recursive DFS style function to traverse the graph
- No global variables
- Document your code (block header comment at the top of your program) and document the DFS function
- Do not modify the order of the neighbors list, just read the edges from the input and insert by pushing back to each node's neighbors list

Sample Run

```
% g++ main.cpp
% ./a.out

LEON!!!

Enter map file: map01.txt

Enter enemies file: enemies01.txt
```

```
I'm coming Ashley!
Village -> GrandHall -> Mines -> Clocktower -> Ballroom -> Church

% ./a.out

LEON!!!

Enter map file: map02.txt

Enter enemies file: enemies03.txt

I'm coming Ashley!
Village -> Mines -> Castle -> Church

% ./a.out

LEON!!!

Enter map file: map03.txt

Enter enemies file: enemies03.txt

I'm coming Ashley!
Village -> Lake -> Church
```

Submission

Submit the source file to code grade by the deadline

References

- Supplemental Video https://youtu.be/phHCejm3lOY
- Link to the top image can be found at https://www.gamesindustry.biz/resident-evil-4-remake-critical-consensus