TP C#9
PathFinding

Assignment

Archive

You must submit a zip file with the following architecture:

```
rendu-tp-firstname.lastname.zip
    |-- rendu-tp-firstname.lastname/
        |-- AUTHORS
        |-- README
        |-- PathFinding/
            |-- PathFinding.sln
            |-- PathFinding/
                |-- Program.cs
                |-- Map.cs
                |-- Room.cs
                |-- everything except bin/ and obj/
```

- Replace *firstname.lastname* with your name.
- The code must compile (There should not be any warning).

AUTHORS

This file should contain the following : a star (*), a space and your login.
Here is an example:

```
* firstname.lastname
```

Please note that the filename is AUTHORS with NO extension.

README

This file should contain information about your TP: what did you manage to do, where did you have a hard time, **what bonii have you done**.
Please note that the filename is README with NO extension.

1 Introduction

1.1 Objectives

During this TP, you will learn new notions such as:

- Pathfinding algorithm (Dijkstra)
- Generic stack: `Stack<T>`
- Tuples: `Tuple<T, ...>`

And review important notions such as:

- Classes
- Exceptions
- Generic list: `List<T>`

2 Course

2.1 Pathfinding

According to Wikipedia: "Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points."
You've got it, this week we are going to talk about finding a way from a point A to a point B. For human beings, it's easy to find the shortest way between these two points. Just think about when you use the map of the metro in Paris. You can find it, but for a computer, it's a lot harder. You have to make it understand what two train stations are, what a distance is, what the connections are between these two places. With two points, it is easy to find the shortest path (because there is only one way) but when you add more stations, 3 for example, how can it find it ? You have eyes, you have a brain (at least some of you have one) so you can compute which way is better than the others but how can a computer think ? You are not going to make a computer like HAL 9000 this week but you are going to make it a little smarter.
Pathfinding is about complexity as well. Our goal is to build an algorithm that quickly computes a path between 2 points in a map. For this TP, we could put all the paths from the source point to the destination point in a matrix and compute (very slowly) the shortest path. But this technique is not scalable and we will see an approach to pathfinding which works with all the maps we want in a reasonable time.

2.2 Graph

Today, we are going to study a simplified version of graphs. A graph is a set of nodes which are linked with edges. You have already seen what a binary tree is during your lessons. A binary tree is a kind of a graph. In this TP, we will use a non-oriented graph and the cost of going through an edge can be positive or negative. Translation : you can go from node A to node B and from node B to node A with a cost of 1 but the cost can be -5. There are different types of graphs but you will learn them next year.

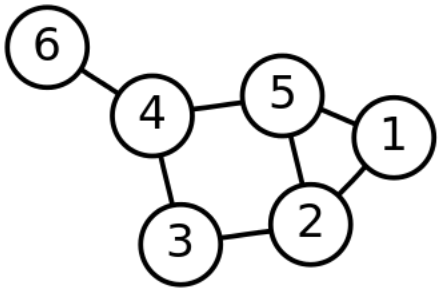

Figure 1: An example of a simple graph

2.3 Dijkstra

The Dijkstra algorithm is a common path-finding algorithm named after Edsger W. Dijkstra a famous computer scientist. He was one of the most influential members of computer science and is known for the path-finding algorithm we study this week but also for many other things that have made computer science what it is today [1]. This algorithm is mostly used with graphs with costs but it still works on the kind of graph we are using today.
The principle of the Dijkstra is quite simple: Each node contains a boolean to check whether it has been visited, its best parent, and a distance representing the distance from the start node to this node. These are the main steps of Dijkstra algorithm:

- You set all the nodes to unvisited and their distances to infinite.
- Set the current node to be the start node. Its distance is 0. It has no parent and has not been visited yet.
- Update the distance of its sons: for each son, you add the distance of the current node (in this case 0) with the distance of the son. For example, if the son is at 5 distances, the son's distance from the current node will be $0 + 5 = 5$. If this computed distance is less than the son's distance (in this case, 5 is a lot less than infinite) you update the son's distance to the computed distance (5 in our case) and you set its best parent to be the current node. Read and reread this point until you understand it.
- Now that you've done that, check the current node as visited and set the current node to point to nowhere.
- Then you look at all the non-visited nodes. You search the node with the shortest distance which **is not visited**. This node becomes your current node.
- Update its sons' distances as you did in the third point of this list (do not forget to set their best parent if needed!) and check the current node as visited.
- Do the last two points until you've done all the paths of the graph. Sometimes you do not need to check all the paths of the graph to find the shortest one but we won't explain that here. If you want to know more, google it[2].

[1] You can find more on Wikipedia, it's really huge.
[2] https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra#Sp.C3.A9cialisation_de_l.27algorithme

- Have you done all the ways ? Great. Now, the easy part. Take a stack, put the final node in it and then go on its best parent. Put it in the stack. Then go on its best parent, and put it in the stack. Then... Do it until you get to the start node. Put the start node in the stack and voilà, you're done.
- To know the shortest path from the start node to the final node, just take all the nodes you've put in the stack one by one.

We know that it can be hard to understand but read this part several times, draw a graph and do the algorithm on it, check it on Wikipedia, etc. Take your time to know how Dijkstra works - it is really important.
[3]

Don't worry, you will be guided throughout the TP. It's easier than it may look!

[3] Slide with examples in french: http://licence-math.univ-lyon1.fr/lib/exe/fetch.php?media=gla:dijkstra.pdf

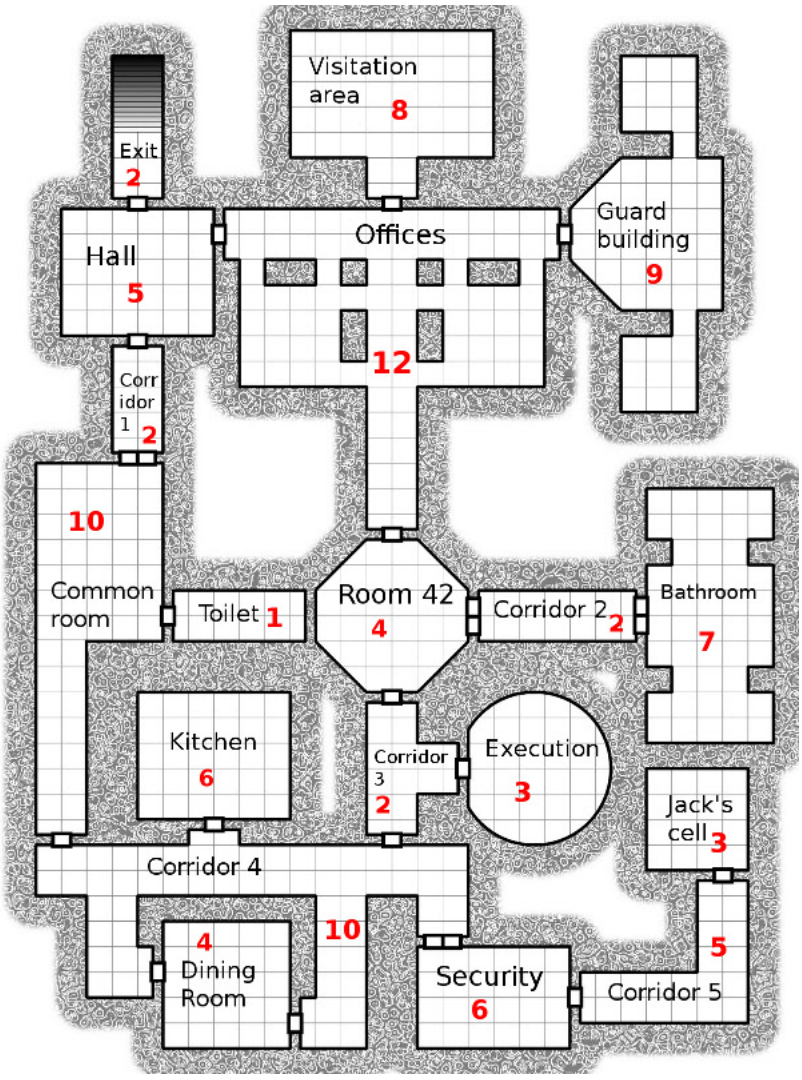

Figure 2: The jail map

3 Exercise: Get Jack the fuck outta here!

Captain Barbossa has captured Jack Sparrow and put him in the biggest jail in the Caribbean so that he can't escape and will have to stay there for eternity. According to the pirate's code, we have no choice, we must deliver Jack as soon as possible.

3.1 Before starting

In this exercise, your goal is to code a pathfinding algorithm to find the shortest path from Jack's cell to the exit.

You are going to use 3 files: **Room.cs**, **Map.cs** and **Program.cs**. Room.cs and Map.cs contain the `Room` and `Map` classes respectively. The Program.cs file contains only the `Main` method.

As we are going to use generic collections like `List<T>` and `Stack<T>`, do not forget to add `using System.Collections.Generic;` at the beginning of each file.

In terms of object design, our graph will be the entire jail and each room will be a node of it.

3.2 Room.cs

Each node of our graph can be seen as a room of the jail. So we need a `Room` class!

Room class

The first class you have to design is the `Room` class. A `Room` has a `name`, a `size` and `neighbours`.
The `neighbours` of a `Room` X are all the other `Room`s connected to X with a door.
The 2 attributes are `private`.
Note that the `size` attribute will allow your Dijkstra to find the shortest path by considering the size of all the rooms. The different sizes and names are given in Figure 2: The jail map.

```
1  public class Room
2  {
3          // Attributes
4          private string name;
5          private int size;
6          private List<Room> neighbours;
7
8          // Methods
9          // FIXME
10 }
```

Room constructor

The `Room` constructor initializes the `name` and the `size` with arguments and the `neighbours` with an empty `List<Room>`.

```
public Room(string name, int size);
```

All the following methods are obviously coded in the `Room` class.

Name getter

As the `name` attribute is `private`, we need a getter to it.

```
public string GetName();
```

Size getter

As the `size` attribute is `private`, we need a getter to it.

```
public int GetSize();
```

Add door with another room

This method adds the room in argument to the `neighbours` list attribute. Do not forget that you should also add the current room in the `neighbours` of the room in argument! (If there is a door to pass from a room A to a room B then you can pass from B to A too)

```
public void AddDoorWith(Room room);
```

Check if a room is connected to another

This methods checks if the actual room has the room in argument in its `neighbours`. It returns `true` if the room in argument is in its `neighbours` and `false` otherwise.

```
public bool HasDoorWith(Room room);
```

3.3 Map.cs

If the nodes are the rooms, you have guessed that the map is the graph. Let's build a `Map` class!

Map class

Here is the main part of the subject!
The `Map` class has one `private` attribute which is a list of rooms (`List<Room>`) called `rooms`.

```
1 public class Map
2 {
3         // Attributes
4         private List<Room> rooms;
5
6         // Methods
7         // FIXME
8 }
```

Map constructor

The `Map` constructor initializes the `rooms` attribute with an array of `Tuple<string, int>`. Remember that the `Room` constructor takes a string.
Hint1: To access the 2 first elements of a `Tuple` use respectively the `Item1` and `Item2` fields.
Hint2: To add an element to a list, just use the `Add` native method of `List<T>`. Something like `this.rooms.Add(new Room(...))` is legitimate in our case.

```
public Map(Tuple<string, int>[] rooms);
```

All the following methods are obviously coded in the `Map` class.

Add a door between 2 rooms

The following method adds a door between 2 `Rooms`.
Do you want to find a specific element in a `List<Room>` with only its name?
`this.rooms.Find(room => room.GetName() == room1)` returns the room named `room1` if it exists in `this.rooms` and `null` otherwise. Check MSDN for more info.

```
public void AddDoorBetween(string room1, string room2);
```

Saving Private Jack

Here it is! The most interesting part of that TP!

You have to code the `FindShortestPath` method which finds the shortest path from `src` to `dest`; `src` and `dest` are room names.
This method returns a `Stack<Room>`[4] because when you've filled your parent/father vector, you will have to push all your rooms, from the destination room to the source room, in the stack. Then, to get the shortest path, from source to destination, just pop them one by one.
This method throws an exception, with a well formatted message, if the `src` room or the `dest` room doesn't exist in the map.

```
public Stack<Room> FindShortestPath(string src, string dest);
```

Don't hesitate to read again the Dijkstra course part or to ask your ACDCs for help if you are completely lost. **Google is your friend as well.**

You can of course help each other but, be careful. As the Dijkstra algorithm is pretty difficult to implement, we will not tolerate any cheating! If 5+ students have the same algorithm we will see it and your mark will be divided by the number of cheaters.

Remember the way

You still have to do something to help the Captain: give him a path to follow ! He will forget the way as soon as he get through the door, go write the path you found in the previous method in a file. Use the method `WritePath` to help him, return false if something bad has happened, true otherwise.

```
public bool WritePath(string filename, Stack<Room> path);
```

The output should be like this:

```
--- Path to follow ---
Room 1
Room 2
...
Room n
--------------------
```

[4] Stack<T> documentation: https://msdn.microsoft.com/fr-fr/library/3278tedw(v=vs.110).aspx

```
Total cost: 24

// If an error occurs, it should display on the console
Error during the opening of [FILENAME]
[ERROR MESSAGE]
```

3.4 Program.cs

In this file we are just going to put our `Main` method to test the pathfinding. As a reminder, don't try to trick us with a hard coded shortest path algorithm. We will not just try your Dijkstra with this prison map but with many other maps, to check if your algorithm is really working well.

```
1  public static void Main(string[] args)
2  {
3          // Create a new map with pairs name/size for each room
4          Map map = new Map(new[] {
5                  new Tuple<string, int>("Corridor 1", 2),
6                  new Tuple<string, int>("Corridor 2", 2),
7                  new Tuple<string, int>("Corridor 3", 2),
8                  new Tuple<string, int>("Corridor 4", 10),
9                  new Tuple<string, int>("Corridor 5", 5),
10                 new Tuple<string, int>("Exit", 2),
11                 new Tuple<string, int>("Hall", 5),
12                 new Tuple<string, int>("Visitation Area", 8),
13                 new Tuple<string, int>("Offices", 12),
14                 new Tuple<string, int>("Guard Building", 9),
15                 new Tuple<string, int>("Room 42", 4),
16                 new Tuple<string, int>("Common Room", 10),
17                 new Tuple<string, int>("Toilet", 1),
18                 new Tuple<string, int>("Kitchen", 6),
19                 new Tuple<string, int>("Bathroom", 7),
20                 new Tuple<string, int>("Dining Room", 4),
21                 new Tuple<string, int>("Security", 6),
22                 new Tuple<string, int>("Jack's Cell", 3),
23                 new Tuple<string, int>("Execution", 3)
24         });
25
26         // Add doors between rooms to get a concrete map
27         map.AddDoorBetween ("Jack's Cell", "Corridor 5");
28         map.AddDoorBetween ("Security", "Corridor 5");
29         map.AddDoorBetween ("Security", "Corridor 4");
30         map.AddDoorBetween ("Dining Room", "Corridor 4");
31         map.AddDoorBetween ("Kitchen", "Corridor 4");
```

```
32          map.AddDoorBetween ("Common Room", "Corridor 4");
33          map.AddDoorBetween ("Common Room", "Toilet");
34          map.AddDoorBetween ("Common Room", "Corridor 1");
35          map.AddDoorBetween ("Hall", "Corridor 1");
36          map.AddDoorBetween ("Hall", "Exit");
37          map.AddDoorBetween ("Hall", "Offices");
38          map.AddDoorBetween ("Visitation Area", "Offices");
39          map.AddDoorBetween ("Guard Building", "Offices");
40          map.AddDoorBetween ("Room 42", "Offices");
41          map.AddDoorBetween ("Corridor 3", "Corridor 4");
42          map.AddDoorBetween ("Corridor 3", "Execution");
43          map.AddDoorBetween ("Corridor 3", "Room 42");
44          map.AddDoorBetween ("Corridor 2", "Room 42");
45          map.AddDoorBetween ("Corridor 2", "Bathroom");
46
47           try
48          {
49                  // Get the path using our Dijkstra algorithm
50                  Stack<Room> path = map.FindShortestPath("Jack's Cell", "Exit");
51
52                  // FIXME: Display a formatted message on stdout
53                  // with the path from the Jack's Cell to the Exit
54          // Do not forget to get and check the predictions !
55          }
56          catch (Exception e)
57          {
58                  // FIXME: Print the exception
59          }
60  }
```

Have you got a path connectring Jack's cell to the exit? Then stand up pirate and scream...

Remember this day as the day you ALMOST caught captain Jack Sparrow!

Oh wait !

Did you think it was over? Nah, you're not that stupid are you? We're talking about Captain Jack Sparrow here! You know how he is, he likes to play games so here is one: he will try to guess the correct path before you give him the paper with it. But because of his rank, you have to write his predictions and compare with the path you got. To do so, code the following methods:

```
public List<string> GetPredictions();
public void ComparePredictions(List<string> predictions, Stack<Room> path);
```

The output should be like this:

```
// The predictions are wrong (the number of rooms are not the same)
Ohoh, you got it wrong !

// Predictions are wrong too
Ok, you're wrong buddy. You got [NUMBER] on [NUMBER] good answers

// Predictions are right
You are a wizard buddy, you were right!
```

4 Bonii

Here is a list of bonii for this TP. It is advised to do them in this order. As they are based on the whole TP, you have to have finished the first part before starting the bonii.

- You can display the correct path in the console before (or after) putting it in the file. But if you do that, do not forget to add some colours.
- The jail is often dirty, so sometimes some rooms are closed in order to be cleaned. Add an option where the user can choose which rooms are closed.
- Add any bonii you want, as long as you have done the other bonii. Be creative, have fun!

Don't forget to tell us what you've done in your README.

The code is the law.