# Homework Assignment – Graphs: Using the graph visualization API

This exercise sheet is relative to the Graph Viewer Application Programming Interface (API). For more information, please consult the official documentation at: https://graphviewercpp.readthedocs.io/en/latest/

## Instructions

- Download the zipped file **cal_fp07_graphviewer_CLion.zip** from the course's Moodle area and unzip it (it contains the folder lib which includes the GraphViewerCpp project, and the folder TP07_graphviewer with required files implementing main(), and the file CMakeLists.txt). **IMPORTANT**: this exercise does not use unit tests!
- In the Clion IDE, open the project that has been used for the CAL course's lab classes.
- Copy the folder **lib/GraphViewerCpp** to folder **lib** of the project.
- If you're using a Linux distribution, install SFML from your package manager; for Ubuntu/Debian, use **`sudo apt-get install libsfml-dev`**. If you're using Microsoft Windows, then all dependencies come already bundled with GraphViewerCpp.
- Copy **TP7_graphviewer** to the root of the project, to the same level of the other lab classes' folders.
- Replace the project's **CMakeLists.txt** file with the new file provided for this lab class.
- Do "*Load CMake Project*" over the file **CMakeLists.txt** in order to load the run configurations for the TP.
- Run the project (Run).
- **Important note**: to read text files in I/O mode, you may need to tell CLion where such files are, by redefining the IDE environment variable "Working Directory" for the TP3 configuration, through menu Run > Edit Configurations... > Working Directory.

## Exercises

**Ex.1:** Base structure of a graph

a. Configure your development environment
   i. Create a GraphViewer instance and create a window (all other changes to the graph should be made after setting the graph center, and before creating the window). Note: to create a window you should use the following code:

```
// Instantiate GraphViewer
GraphViewer gv;

// Set coordinates of window center
gv.setCenter(sf::Vector2f(300, 300));

// Create window
gv.createWindow(600, 600);

// Join viewer thread (blocks till window closed)
gv.join();
```

b. Create a vertex
   i. Create a blue vertex with ID 0 at (200, 300).

Note: to create a vertex with those properties you should use the following code:

```
Node &node0 = gv.addNode(0, sf::Vector2f(200, 300)); // Create node
```

```
node0.setColor(GraphViewer::BLUE);                    // Change color
```

ii. Create a blue vertex with ID 1 at (400, 300).

iii. Create a black edge between the two previously created vertices.

Note: to create edges, use the following code:

```
// for bidirectional edges
Edge &edge1 =
gv.addEdge(idEdge,idSource,idDestination,GraphViewer::EdgeType::UNDIRECTED);
// for directed edges
Edge &edge1 =
gv.addEdge(idEdge,idSource,idDestination,GraphViewer::EdgeType::DIRECTED);
```

iv. Remove vertex 1

Note: to remove a vertex, run the following method:

```
gv.removeNode(1);
```

v. Add a new vertex with ID 2 at (500, 300).

vi. Add a black edge between vertices 0 and 2.

vii. Add a label to vertex 2 with a text of your choosing

Note: to add a label, use the following code:

```
node2.setLabel("This is a vertex");
```

viii. Add a label to an edge with a text of your choosing

Note: to add a label to an edge use the following code:

```
edge2.setLabel("This is an edge");
```

ix. Make vertex 2 green

Note: to configure a vertex's color, use the following code:

```
node2.setColor(GraphViewer::GREEN);
```

x. Make the edges yellow

Note: to configure all edges' color use the following code:

```
for(Edge *edge: gv.getEdges())
    edge->setColor(GraphViewer::YELLOW);
```

xi. Make the "background.png" image the background

Note: to configure the background image use the following code

```
gv.setBackground("../TP7_graphviewer/resources/background.png");
```

**Ex.2:** Graph animations simulation.

a. Add vertices with the following attributes:

```
id:  0, x: 300, y:  50
id:  1, x: 318, y:  58
id:  2, x: 325, y:  75
id:  3, x: 318, y:  93
id:  4, x: 300, y: 100
id:  5, x: 282, y:  93
id:  6, x: 275, y:  75
id:  7, x: 282, y:  58
id:  8, x: 150, y: 200
id:  9, x: 300, y: 200
id: 10, x: 450, y: 200
id: 11, x: 300, y: 400
id: 12, x: 200, y: 550
id: 13, x: 400, y: 550
```

b. Add edges with the following attributes

```
id:  0, idSourceVertex:  0, idDestinationVertex:  1
id:  1, idSourceVertex:  1, idDestinationVertex:  2
id:  2, idSourceVertex:  2, idDestinationVertex:  3
id:  3, idSourceVertex:  3, idDestinationVertex:  4
id:  4, idSourceVertex:  4, idDestinationVertex:  5
id:  5, idSourceVertex:  5, idDestinationVertex:  6
id:  6, idSourceVertex:  6, idDestinationVertex:  7
id:  7, idSourceVertex:  7, idDestinationVertex:  0
id:  8, idSourceVertex:  4, idDestinationVertex:  9
id:  9, idSourceVertex:  9, idDestinationVertex:  8
id: 10, idSourceVertex:  9, idDestinationVertex: 10
id: 11, idSourceVertex:  9, idDestinationVertex: 11
id: 12, idSourceVertex: 11, idDestinationVertex: 12
id: 13, idSourceVertex: 11, idDestinationVertex: 13
```

c. Animation

Note: displaying is only allowed if the GraphViewer instance can obtain the lock on its drawing mutex. If you want to change properties of a node/edge after having called **createWindow()**, you must **lock()** the GraphViewer instance, and **unlock()** it after you're done. It may also work if you do not lock/unlock, but it is not guaranteed. So that the animation is perceptible, you can pause the execution between re-draws (sleep(numSeconds) in Linux and Sleep(numMiliSeconds) in Windows).

i. Make nodes 12 and 13 alternate between their original positions and

```
id: 12, x: 250, y: 550
id: 13, x: 350, y: 550
```

every 1 second, so the stick-figure is moving its legs.

Note: to change the position of a node use the following code:

```
Node &node12 = gv.getNode(12);              // Get reference to node
node12.setPosition(sf::Vector2f(250, 550)); // Set position
```

**Ex.3:** Load a graph from a file.

a. Read the **nodes.txt** and **edges.txt** files in folder **resources/map1** to load the graph represented in them.

The files use the following format:

- nodes.txt:
  - Line 1 contains the number of nodes N
  - Line 1+i (0≤i<N) describes the i-th node in format **<id> <x> <y>**
- edges.txt:
  - Line 1 contains the number of edges E
  - Line 1+i (0≤i<E) describes the i-th edge in format **<id> <fromNode> <toNode>**

**Ex.4:** Performance *[optional]*

a. Run the code in **exercise4()**. You should see a graph of the roads of the Metropolitan Area of Porto (Portugal) over satellite imagery of that area. If you try to interact with it, you'll see it is a bit slow/unresponsive; this can be verified by pressing the key 'D', which shows the number of frames per second in the lower left corner (you should see something around 2 FPS).

b. Add the following lines after the background is set but before the window is created:

```
gv.setEnabledNodes(false);      // Disable node drawing
gv.setEnabledEdgesText(false);  // Disable edge text drawing
```

Check the frame rate again; it should have improved to about 10-15 FPS, and the window should be much more responsive now. (Set the nodes' outline thickness and size to 0.0 to remove the empty space between edges connected to the same node).

c. Add the following line just before creating the window (do not erase the changes you made previously):

```
gv.setZipEdges(true);
```

You should see a massive framerate improvement, to about 250-300 FPS. This is because all edges are being condensed into a single object to improve performance. If you enable edge zipping, you should call **gv.setZipEdges(true)** again once you're done making all updates, so the zipped edges object is updated accordingly.