# Weekly report n°2

Secretary: Fauste LEBOEUF

October 11, 2024

## 1 Data structure

We decided to represent the maze by a graph. The graph is an array of vertices whose structure is defined as follow:

```
struct vertex
{
        unsigned int id;

        // Pointers to the neighbours of the vertex.
        struct vertex *top;
        struct vertex *left;
        struct vertex *bottom;
        struct vertex *right;

        // Boolean to know if the vertex is the
        // entry point of the graph.
        unsigned int is_start;

        // Boolean to know if the vertex is the
        // exit of the graph.
        unsigned int is_end;
};
```

## 2 Specification

We will use six functions to generate the maze according to the algorithm described in the assignment:

```c
/*
 * Preconditions: None.
 * Postconditions: None.
 *
 * Returns a pointer to the first element of the graph,
 * or NULL on error.
 */
struct vertex *init_graph(int const width, int const height,
        int id_start);

/*
 * Preconditions: - width must be equal to the width
 *                  of graph.
 *                - size must be equal to the size
 *                  of graph.
 * Postconditions: If a wall has been added, graph
 *                 has lost an edge.
 *                 Otherwise, graph is not modified,
 *                 or NULL.
 */
void add_wall(struct vertex *const graph,
        int const i, int const j,
        int const width, int const size);

/*
 * Preconditions: - width must be equal to the width
 *                  of graph.
 *                - height must be equal to the height
 *                  of graph.
 * Postconditions: graph represents a maze randomly
 *                 generated, or NULL.
 */
void generate_maze(struct vertex *const graph,
        int const width, int const height);
```

```
/*
 * Preconditions: - graph must be initialized.
 *                - width must be equal to the width
 *                  of the orginal graph.
 *                - height must be equal to the height
 *                  of the original graph.
 *                - first_vertex_id must be the first
 *                  vertex id of graph.
 *                - cur_width must be the width of graph.
 *                - cur_height must be the height of graph.
 * Postconditions: graph represents a maze randomly generated.
 */
void generate_maze_aux(struct vertex *const graph,
        int const width, int const height,
        int const first_vertex_id,
        int const cur_width, int const cur_height);

/*
 * Preconditions: None.
 * Postconditions: The memory allocated for graph by
 * init_graph() is freed, or does nothing if graph is NULL.
 */
void free_graph(struct vertex *graph);

/*
 * Preconditions: - step must be a positive integer.
 * Postconditions: None.
 *
 * Returns an integer n equals to min + k * step (k >= 0)
 * such that n is between min and max included.
 */
int random_int(int min, int max, int const step);
```

## 3 Tests

To test our data structure and maze generation algorithm, each team member will be focusing on specific tasks:

- Djakhar:
  - Verify that each vertex's neighbour is indeed the neighbour of the current vertex (i.e. a cell does not have a non-adjacent cell has a neighbour).
- King:
  - Verify that the vertices at the edges of the maze do not have neighbours outside of the maze.
- Mensanh:
  - Test the functions with limit values.
  - Test the maze generation algorithm.
  - Verify the behaviour of the maze generation algorithm for extrem dimensions.
- Edgar:
  - Test the maze generation algorithm with invalid data (wrong types, negative values).
- Fatima:
  - Verify that the entry point exists.
  - Verify that the exit is at the bottom right of the maze.
  - Verify that the exit is unique.
- Kian:
  - Test the maze generation algorithm for different dimensions.
  - Verify that the graph has the correct number of vertices and edges to represent the maze.
- Fauste:
  - Verify that a $1 \times 1$ graph has a unique vertex and the maze generation algorithm does not create any wall.
  - Verify that each wall has a door.