PROJET D'AP2
LABYRINTHE

RÉALISÉ PAR:

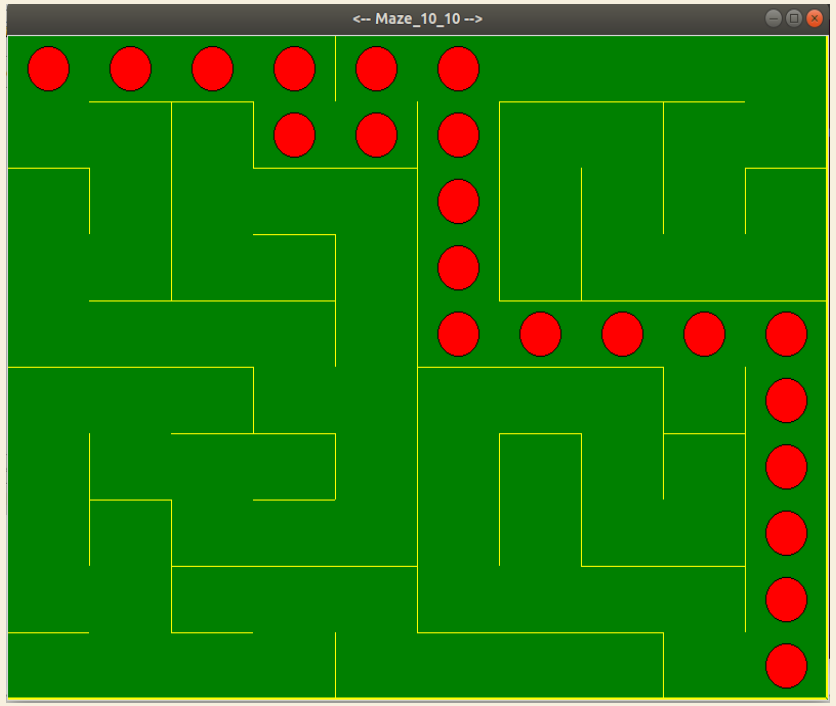
DIALLO MAMADOU, COULIBALY IVETTE &
SAIDI MAHREZ

LES POINTS A ABORDER

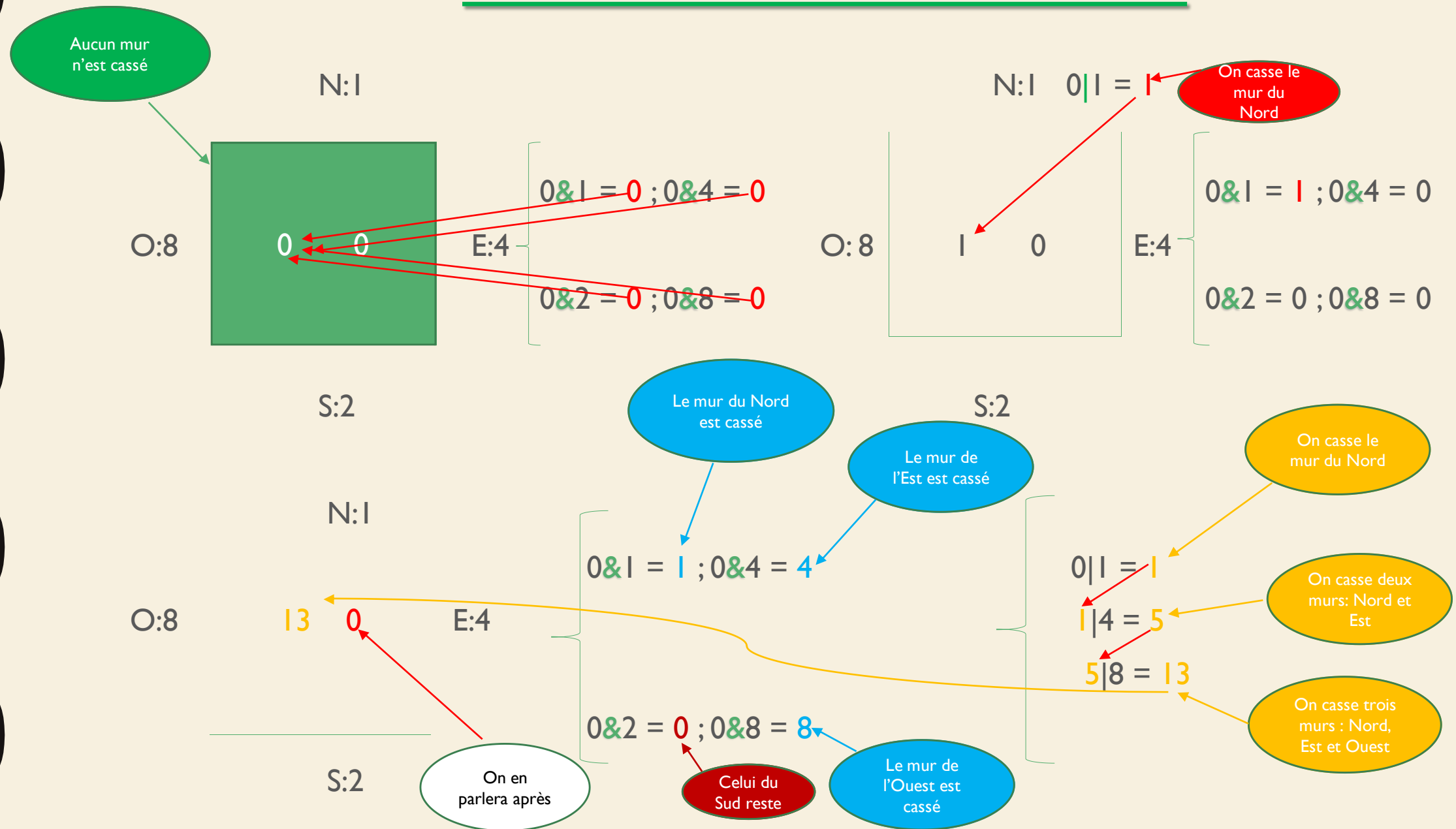
- La description du projet
- Explication des principes utilisés
- Détails des modules et classes effectués
- Les fonctionnalités du programme réalisé
- Une exécution du programme

DESCRIPTION

- Le projet consistait à réaliser un programme permettant à un utilisateur de créer un labyrinthe parfait de son choix (dont les chemins sont produits aléatoirement) aussi à effectuer une recherche de chemin entre deux points du labyrinthe.



PRINCIPES UTILISÉS



LA CLASSE CELL

. **__init__():** Qui permet de créer un objet de type « Cell ».

. **get_direction(),get_direction_situation(), get_c():** qui font respectivement les travaux qui suivent récupérer les quatre directions, l'état des directions, et un autre état de la cellule qui dit si elle fait partie d'un chemin recherché ou non.

. **change_c():** Change la valeur de « c ».

. **break_wall():** Celle-ci casse un mur de la cellule.

. **visit(), unvisit():** Qui respectivement visite et remet une cellule à non visitée.

. **is_visited() :** Un prédicat qui dit si une cellule est visitée ou non.

UNE FONCTION AUXILIAIRE

- La fonction « neighborhood » comme son nom l'indique elle nous donne les cellules voisines d'une cellule du labyrinthe dont les coordonnées lui sont passées en paramètre. Selon la position de la cellule dans le labyrinthe, une cellule a au plus quatre cellules voisines et au moins deux.

LA CLASSE MAZE

Les différentes méthodes essentielles utilisées:

. **__init__():** Qui permet de créer un objet de type « Maze » avec des dimensions données.

. **get_width(),get_height(), get_all_cells(), get_cell():** qui font respectivement les travaux qui suivent récupérer la largeur, la hauteur, toutes les cellules et une cellules du labyrinthe.

.**break_walls():** Celle-ci casse le mur séparant deux cellules adjacentes

. **paths():** Avec l'algorithme de fusion des cellules à partir d'une cellule choisie aléatoirement on obtient des labyrinthes aléatoires et parfaits.

. **list_lines():** A partir du principe utilisé dans la class cell et des modifications faites sur les cellules celle-ci représente toutes les lignes du labyrinthe dans une liste.

.**__str__():** Elle transforme l'objet « Maze » en chaîne de caractères.

.**make_maze_file():** Ouvre un canal vers un fichier et recopie le labyrinthe dans ce dernier.

.**point_to_point_path():** Parcours les cellules du labyrinthe à partir du point de départ pour retrouver le point d'arrivé.

MODE D'EMPLOI

Dans un interpréteur de commandes :

La commande : main1.py permet de créer un labyrinthe par défaut de taille 5 5.

La commande : main1.py width height crée un labyrinthe de taille width et height.

Pour rechercher un chemin en même temps qu'on crée un labyrinthe il suffit d'ajouter les coordonnées des deux points

La commande : main1.py width height x1 y1 x2 y2 réalise cette tâche.

Et pour utiliser l'interface graphique il suffit juste dans toutes les commandes remplacer main1.py par main2.py.

MERCI POUR VOTRE ATTENTION