Processus de développement

Zayed Herma

1 Analyse du besoin

— On souhaite trouver une solution pour le problème défini au **test5** qui consiste à trouver, s'il existe, un chemin entre le point de départ "S" et un point "E" en respectant les contraintes de l'énoncé.

— La solution a été implementée en language **python**. Le code du script se trouve dans le dossier **test5**.

— Interfaces :

  — Le programme prend un fichier texte en argument, et peut être appelé en ligne de commande.

  — Le programme résout le problème décrit précédemment.

  — S'il y a une solution au problème, le programme écrit une solution dans STDOUT et termine avec le exit code 0.

  — S'il n'y a pas de solution, le programme termine avec le exit code 1.

  — Pour savoir lequel des deux exits s'agit-il, il suffit de lancer la commande echo $? dans le terminal à la fin de l'exécution.

2 Architecture

2.1 Stratégie de développement

— Dans l'implémentation de la solution, on suppose que l'on dispose du point de départ S.

— La stratégie choisie est de résoudre le problème par l'algorithme de parcours en longueur, qui permet de trouver une solution si elle existe.

— Le choix particulier de cet algorithme répose sur la simplicité de l'implémentation en termes de code et de structure de données.

— Cependant, cet algorithme de parcours en longueur ne donne pas toujours le chemin le plus court entre "S" et "E". Cette exigence n'étant pas exigée par le client.

2.2 Architecture fonctionnelle

2.2.1 Vue logique

— Le code de la solution générale du problème **test5.py** est divisé en plusieurs fonctions concises dont chacune joue un rôle important dans le bon fonctionnement du code.

2.2.2 Vue de processus

— La fonction **solutionGrid** appelle la fonction récursive **findPath** qui appelle à son tour les autres sous-fonctions qui permetent de dérouler correctement l'algorithme.

2.3 Architecture technique

2.3.1 Vue de réalisation

— Le code **test5.py** peut être exécuté sur macOs , Windows, Lunix...

— Le code **test5.py** ne dépend pas de sources externes. Il peut être utilisé pour servir dans la résolution d'autres problèmes similaires.

2.3.2 Contraintes de réalisation

— L'algorithme de parcours en longueur est universel, et peut être implementé dans plusieurs languages.

— Tant que les entrées de l'algorithme ne sont pas changés et que les modules utilisés (**json,sys**) fonctionnent bien, il n'y a pas d'impact sur l'exécution du code.

— Le code est implementé en **Python**, par conéquent le respect des règles de codage est toujours possible.

3 Plan de validation

— L'objectif est d'implémenter un algorithme permettant de trouver un chemin entre un point de départ "S" et un point "E" en respectant les contraintes de l'énoncé (sans brûler).

— Ce qui doit être validé, est l'exactitude de l'algorithme en vérifiant manuellemnt sa sortie suite à plusieurs tests d'entrées différentes.

— Le critère de succès de l'agorithme correspond à trouver un chemin valide en sortie : un chemin reliant les points "S" et "E" sans passer par une case voisine du feu. Le critère d'échec correspond au cas contraire.

— Pour s'assurer du plan de validation, l'algorithme a été exécuté sur différents tests et les résulats obtenus ont été enegistrés dans le fichier **Captures des solutions**.