# PROJET 3 : Help MacGyver to escape from the Red Keep !

Document texte expliquant ma démarche et comprenant le lien vers mon code source (sur Github). Développez notamment le choix de l'algorithme. Expliquez également les difficultés rencontrées et les solutions trouvées. Le document doit être en format pdf et ne pas excéder 2 pages A4.

Lien Github : https://github.com/Maximedu13/Help-MacGyver-to-escape-from-the-Red-Keep-

**Les règles du jeu.**

Emprisonné par la reine Cersei Lannister, MacGyver doit s'échapper du Donjon Rouge en réussissant à collecter 3 objets :

- une aiguille
- de l'éther
- un tube en plastique

Ceux-ci permettront la fabrication d'une seringue, visant à endormir la gardienne. Il doit se présenter à celle-ci avec ces 3 objets. Dans le cas contraire, il reste emprisonné et meurt.

**Les fichiers**

- home.py : c'est le menu du jeu ainsi que son point de départ. L'utilisateur lance le terminal dans le dossier du jeu et écrit home.py puis appuie sur entrée. Une interface apparait alors. L'utilisateur peut lancer le jeu en appuyant sur F1, le quitter en appuyant sur F2, et même couper le son avec la touche F3.
- main.py : c'est le fichier central du jeu. Il permet d'afficher les personnages, les items, le labyrinthe, le sac, de collecter les items etc.
- constants.py : il s'agit des constantes du jeu (texte, couleurs, polices etc.)
- maze.py : génère le labyrinthe ainsi que les objets/items collectables.
- character.py : il permet à Macgyver de se déplacer et lance la méthode victoire ou défaite.
- defeat.py et victory.py : conditionnent la victoire et la défaite.
- map : la carte du jeu sous forme de symboles.
    - ○ est un sprite qui affiche la case de départ.
    - ■ est un sprite qui affiche un mur.
    - – est un sprite qui affiche un chemin
    - ▫ est un sprite qui affiche la case d'arrivée.

  Les fichiers main, home, victory, et defeat sont organisés de façon bilatérale : une méthode qui charge les éléments et une méthode qui les affiche. Les fichiers maze et character définissent différentes classes et méthodes.

**Choix des algorithmes**

Je développerai dans cette partie certains algorithmes.

**L'algorithme de déplacement de Macgyver :** Les méthodes move_to_the_top, move_to_the_bottom, move_to_the_left, et move_to_the_right conditionnent le déplacement du héros suivant les flèches directionnelles du clavier. Pour ces 4 méthodes, on procède de la même façon : on vérifie par l'intermédiaire d'une condition if selon la direction choisie si celle-ci ne dépasse pas le cadre du jeu (coordonnées x et y comprises entre 0 et 14). A l'intérieur même de cette condition if, une autre condition if va vérifier si le point de destination n'est pas un mur, car macgyver ne peut pas en franchir. On incrémente ou décrémente alors les coordonnées de

macgyver suivant l'axe des abscisses (x) et des ordonnées (y). La position réelle en pixels est obtenue en multipliant le résultat par la taille d'une case.

**L'algorithme de position aléatoire des objets** : La méthode Define_position permet de définir une position aléatoire unique des items collectables sans que ceux-ci puissent se superposer. Pour cela, j'ai crée dans un premier temps une liste vide list_random qui va pouvoir stocker les positions x et y des items. Puis par l'intermédiaire d'une boucle while, qui vérifie la structure ou sprite d'arrivée de l'objet, j'ai créé les positions x et y de l'item via la fonction random.randint(). Elle est donc comprise entre 0 et 14 pour qu'il ne se retrouve pas hors-jeu. obj_sprite_x et obj_sprite_calculent en pixels réels les positions de l'item. Puis on ajoute les positions de l'item dans list_random. Enfin pour éviter les collisions d'items, on retourne la fonction si jamais les positions de litem 1, 2, ou 3 sont égales.

**L'algorithme de chargement et de collecte des items :** La méthode load_items permet ceux-ci. Elle crée une liste avec les 3 items collectables, ether, aiguille et tube. Elle lance la methode define_position décrite précédemment et lance également la methode display_items via une boucle for pour afficher tous les items sur le labyrinthe. Pour collecter un item, on vérifie par une condition if que les coordonnes du héros soient égales aux coordonnées de l'item en question. Si celle-ci- est vérifiée, une petit son annonce la collecte d'un item, incrémente le sac de macgyver, déplace l'item collecté dans le sac de Macgyver (interface prévue à cet effet) et efface les points d'interrogation ( = sac vide).

**L'algorithme de fin de jeu :** endgame : conditionne la victoire ou de la défaite du joueur/Héros. On va vérifier dans les deux cas avec une condition if si le héros se présente devant la gardienne donc que le sprite d'arrivée de macgyver soit la ligne d'arrivée (sprite ▫). La seconde condition combinée a la première sépare la défaite de la victoire. Il s'agit du nombre d'objets présents dans le sac de macgyver. S'il est égal à 3, une variable d_b appelée par un return lance le fichier victory.py. S'il n'est pas égal a 3, alors une variable d_b appelée par un return lance le fichier defeat.py. Dans les deux conditions, on lance un main.__init__() pour que le joueur puisse rejouer.

**Difficultés rencontrées et solutions trouvées.**

- La première difficulté que j'ai rencontrée est l'appels de fonctions externes.
- La seconde est comment créer une fonction permettant de générer un labyrinthe a part d'un fichier externe (ici map).
- La troisième difficulté a laquelle j'ai été confronté a été définir une position aléatoire pour un objet sur le labyrinthe.
- Refactoring de code : comme je n'ai pas eu de soutenance pendant 2 semaines, mon mentor m'a aidé à comprendre le principe de refactoring de code, bien séparer le code en différentes fonctions pour le rendre plus lisible.
- Respecter les normes de la PEP8.

**Améliorations possibles.**

- Changer les sprites et donner un relief moins « 2D » au jeu.
- Enrichir le jeu (élargir le labyrinthe en dimension, rajouter des objets, éventuellement ajouter des personnages qui pourraient interagir avec le héros etc.)