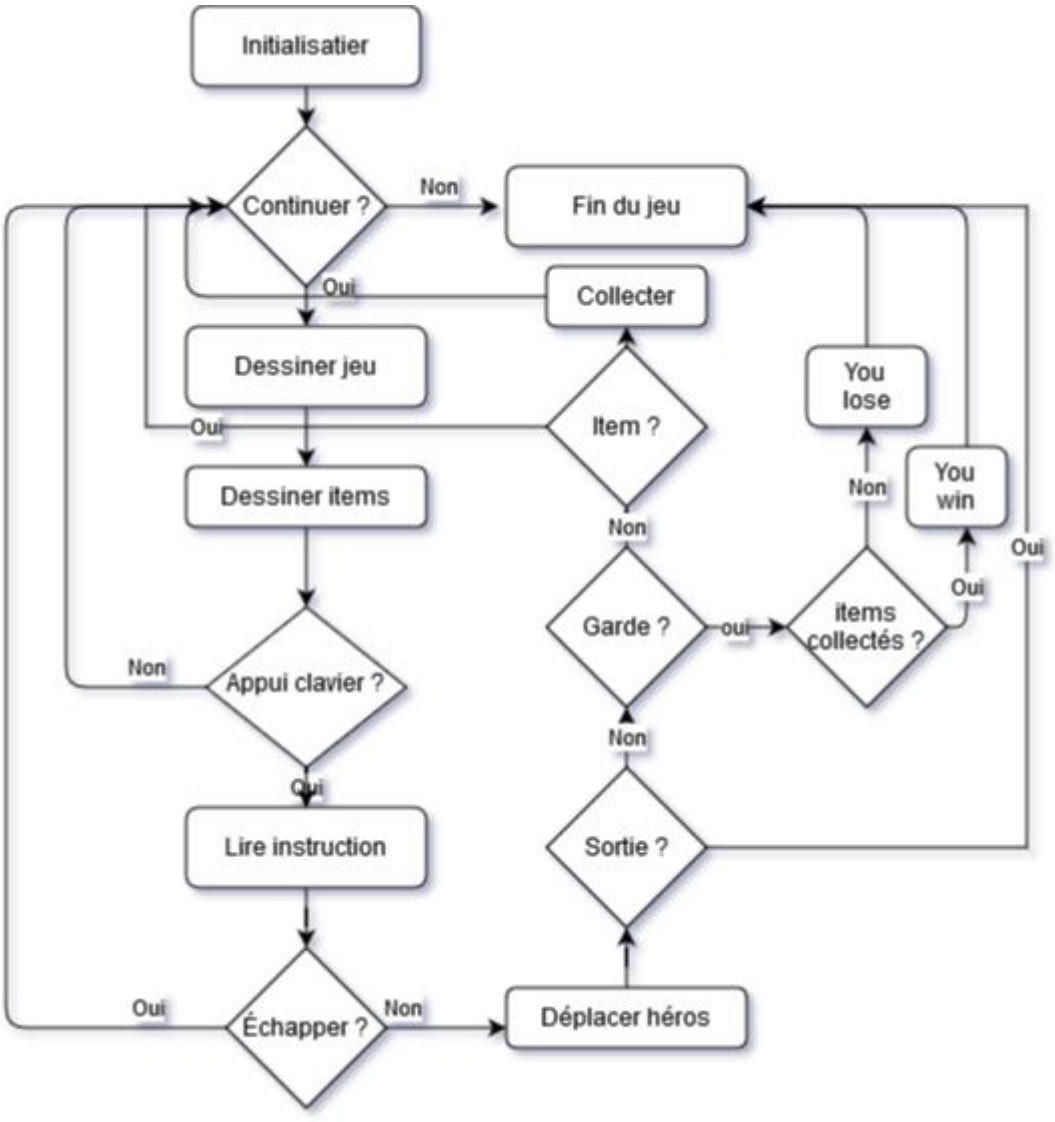
Demarche

Lien du projet sur Github:
https://github.com/jean-charles-gibier/macgyver#macgyver

Choix de l'algorithme central :
Il s'agit d'une boucle standard basée sur l'attente d'un appui clavier.
(Event pour pygame / getch pour la version tetxe)



La conception est basée sur le principe de la représentation du jeu par fichier texte. Dans un fichier de 15 lignes X 15 caractères, chaque caractère représente un "item" rattaché à un "sprite" (excepté les éléments indiquant l'entrée et la sortie du jeu qui ne sont pas affichés dans cette version de jeu).
Un "item" peut être un personnage, un objet ou un emplacement (entrée et sortie du jeu).
Les murs, l'entrée et la sortie sont inscrits dans le plan décrit dans le fichier.

Le héros est placé sur la case entrée en début de partie.
Les autres éléments sont placés sur le jeu par la fonction random.

Éléments/items	Signes caractères	Explications
Le héros	M	Mc Gyver, seul personnage mouvant.
Le garde	G	Le Garde est placé en fonction des autres items
Le tube	T	Ces éléments sont disposés au hasard par le programme lors de l'initialisation.
La bouteille	E	
L'aiguille	N	
Le mur	#	Éléments disposés en fonction du fichier «map».
Entrée	S	S for start.
Sortie	E	E for end.

Afin de faciliter la détection des déplacements possibles, le programme va répertorier les tuples des positions accessibles (ie :toutes sauf les murs). Les déplacement seront exclus si les positions résultantes ne font pas partie de cette liste. Cette méthode est plus pratique que celle constituant à tester le contenu de la case (x+1,y), (x,y-1) etc. à chaque déplacement.

Exemple de problèmes rencontrés / résolus :

La recherche de chemin pour placer les items «dans le bon sens» :

Afin ne pas bloquer le déroulement du jeu, il faut positionner le garde de manière à ce que tous les objets puissent être récupérés avant de passer devant le garde.

Solution : définir au moins un chemin praticable (entre McGyver et la sortie) et disposer les items / perso dans un ordre aléatoire en disposant le garde en dernier sur le chemin choisi.
La découverte du chemin se fera par appels récursifs d'une méthode explorant les voisins immédiats à chaque itération. Il suffira de trouver le 1er chemin (tous le chemins ne sont pas investigables) praticable pour générer une configuration jouable. La fonction retourne un erreur si le chemin n'est pas praticable (par exemple si la sortie n'est pas accessible car murée ou si un des éléments entrée ou sortie est manquant dans le plan).

Au pire la complexité est de l'ordre factorielle N mais avec N limité à 225 (15 x 15 emplacements) et la contrainte des murs limitant la recherche, cette lourdeur algorithmique reste acceptable.

Le découplage présentation / contrôle :

Pour démontrer que le programme respecte la séparation des logiques model/vue/controle, j'ai pris la peine d'implémenter un mode de présentation textuelle du jeu en plus de la version graphique.
Le choix se fait selon le paramètre 'interface' de la ligne de commande. (Cf explication dans le lien ci dessus)
La librairie pygame n'est chargée que si l'implémentation graphique est appelée (en fait pas tout à fait: mais j'expliquerais pourquoi dans la soutenance).

Solution :
Création d'une classe "interface"visant à abstraire les méthodes de présentation et chargement de modules dynamiques en fonction des différents modes sélectionnés.

- l'"implémentation en mode texte, plus compliquée qu'il n'y parait car il faut émuler un getchar()
(saisie des instructions caractère par caractère sans entrée au clavier) valable pour Windows et Unix

- la hiérarchie des classes item/perso (perso est un item qui se déplace => méthode deplacer)