https://github.com/SoCrespo/macgyver

Conception de la structure du labyrinthe

Après avoir dessiné le labyrinthe, j'ai d'abord voulu le traduire sous forme d'une liste à 2 dimensions de 0 et de 1, selon qu'il s'agissait d'un couloir ou d'un mur. Mon mentor m'a alors précisé que le labyrinthe devait être un fichier texte facilement modifiable, et qu'il fallait aussi y faire figurer la position de départ de Mac et la sortie. J'ai donc conçu ce fichier texte puis décidé de le traduire en un dictionnaire contenant, en clé, des tuples (x, y) avec les coordonnées de chaque case, et en valeur, le contenu de la case, sous forme d'un caractère.

J'ai ainsi pu extraire facilement les positions du joueur et du garde, ainsi que les coordonnées des couloirs (le test "Mac peut-il se déplacer sur cette case" était donc simple). Une liste 2D aurait aussi été utilisable mais aurait nécessité le recours à des indices, plus lourd.

Pour la sortie par exemple, s'il est très simple de visualiser où elle est sur un dessin, il est plus difficile de traduire dans une liste le concept de "trou dans un mur d'enceinte". Le fait de la faire figurer explicitement dans le labyrinthe à l'aide d'un caractère dédié (":") simplifiait grandement son extraction (qui serait cependant possible en testant l'existence d'un unique trou dans les 1ères et dernières lignes et colonnes).

J'ai prévu le test de la présence d'une et une seule sortie : une amélioration possible serait de vérifier qu'elle ne se trouve pas dans un angle du labyrinthe, c'est-à-dire que ses coordonnées ne sont pas dans [(0, 0), (0, laby.height), (laby.width, 0), (laby.width, laby.height)]. On pourrait également tester que la forme du labyrinthe est bien un rectangle (toutes les lignes de même longueur que la 1ère), un carré (rectangle + hauteur == largeur), etc.

Conception de l'algorithme

J'ai commis l'erreur de vouloir formaliser trop tôt les classes à utiliser, alors que je n'avais pas détaillé assez finement les actions à effectuer. Par exemple, une action aussi simple que "déplacer le joueur" en nécessite en fait plusieurs :

- au plan conceptuel : vérifier que le déplacement est possible (Mac ne peut pas passer à travers les murs), modifier la position de Mac, transformer la case où il était en case vide; s'il y a un objet, le supprimer du labyrinthe et le stocker quelque part...
- au plan graphique : récupérer la touche pressée par le joueur, actualiser l'image du jeu pour refléter la nouvelle position du personnage, et aussi celle des objets…

Faute d'avoir bien défini toutes ces fonctions, il m'était difficile de les répartir dans la bonne classe. J'ai ainsi buté sur la rédaction de l'algorithme global. J'ai fini par rédiger un algorithme procédural unique, au départ sans les objets à ramasser, et qui fonctionnait en

demandant à l'utilisateur de taper U (pour "up"), D (pour "down"), etc. J'ai ainsi pu construire la boucle principale et la condition de victoire.

Partant de cet algorithme procédural, j'ai ensuite détaillé et scindé les fonctions, jusqu'à aboutir à des fonctions simples, faisant généralement 1 seule chose, et dont le rattachement à telle ou telle classe devenait ainsi plus clair.

Toutefois ce rattachement n'a pas toujours été évident. J'ai ainsi longtemps hésité sur le meilleur endroit où mettre les méthodes de classe concernant les objets à ramasser. Valait-il mieux les mettre dans la classe qui les instancie (Tool), ou dans une classe à part ? Cette réflexion m'a fait comprendre qu'il y a très souvent plusieurs façons de coder pour arriver au même résultat. Un constat un peu déconcertant pour un débutant qui voudrait qu'il n'y ait qu'une bonne réponse !

Utilisation de Pygame

Une fois la première version de l'algorithme créée (sans les objets), j'ai mis en place l'interface graphique afin de pouvoir le tester plus facilement. J'ai éprouvé au départ des difficultés à comprendre l'interface et sa logique telle que présentée dans la documentation. Ce sont les tutoriels et exemples trouvés lors de mes recherches sur différents sites qui m'ont éclairée.

L'analyse plus poussée de la fonction pygame.event.get() (je détaillerai ce point en soutenance) m'a permis de saisir l'importance d'une lecture attentive de la documentation : c'est notamment parce que j'avais voulu lire directement la doc de cette fonction, sans passer par la présentation générale de la classe Event, que je n'ai pas bien compris au début comment elle fonctionnait.

Conclusion

Ce premier vrai contact avec le code a été source d'innombrables erreurs et moments de frustration, tous très formateurs.L'un des meilleurs conseils qui m'a été donné a été celui de procéder à la transformation de mon unique algorithme procédural en ensemble de modules et classes, par très petites étapes :

- changer le nom d'une variable, créer une classe, y transférer une méthode…
- vérifier que le programme était toujours fonctionnel suite à ce changement
- et commiter.

Dès que je me suis écartée de cette ligne de conduite, j'ai rencontré des difficultés. Avancer à très petits pas a été le seul moyen de mener à bien le projet. Cette démarche demande de se discipliner, mais le résultat en vaut la peine.