### 

Rapport de Projet

# Répartition des tâches

La première étape fut de nous répartir les tâches afin de réaliser le projet sans disparités dans les charges de travail.

Voici la répartition choisie:

* Quentin se charge de programmer la carte, joueur et liste joueur en version classique puis objet et pour finir le diaporama.
* Xavier programme le plateau et le labyrinthe en version classique puis orientée objet.
* Maxime s’occupe de la programmation de la matrice, du joueur, de la liste des joueurs en version classique puis orientée objet et du rapport.

# Les méthodes de test adoptées

Lors de la programmation il était important de régulièrement tester nos fonctions. En effet, certaines étant dépendantes des fonctions précédentes, il est important que celles-ci ne soient pas fausses nous faire perdre du temps. Pour cela nous vérifions à l’aide d’exemples que chaque fonction retournait ce qu’elle devait retourner. Enfin les programmes complets étaient passés dans les test fournis avec le sujet, ce qui nous permettait de corriger les éventuels derniers défauts.

# Le choix des structures de données utilisées

Il était important, dès le départ, de choisir les bonnes structures de données afin de simplifier la mise en commun des programmes. Nous avons donc choisi de représenter les ***joueurs*** avec des dictionnaires dont les clés seraient: .Le Nom(str) .Le numéro(int) .Les trésors(liste)

Ces joueurs sont regroupés dans une liste afin d’obtenir une liste de joueurs.

Pour ***listeJoueurs***, il a été décidé de faire une liste regroupant ces joueurs afin de les manipuler représenter efficacement.

La structure pour représenter les ***cartes ainsi que le labyrinthe*** est un dictionnaire puisqu’il permet une manipulation intuitive des cardinalités en leur associant leur booléen ainsi que pour la liste de pions de la carte et du plateau avec la liste de joueurs pour le labyrinthe

# Principaux algorithmes et les extensions apportées

## *Les algorithmes*

Les principaux algorithmes implémentés sont ceux pour la création du plateau qui est bien plus complexe qu’il n’y paraît avec notamment la fonction pour vérifier l’accessibilité.

**Le plateau**

Pour ce faire, nous avons décidé d’utiliser un dictionnaire qui indiquait le nombre de chaque pièce restantes ainsi que la fonction pour coder cette pièce afin d’en tirer une au hasard qui n’était pas une case fixe.



|  |
| --- |
| **def** **choisi\_carte**(cpt\_murs):  """  choisi aléatoirement une parmi les cartes restantes  cpt\_murs est une dictionnaire qui associe à chaque type de carte la fonction pour la coder ainsi que la quantité restante  retourne la carte et modifie cpt\_murs avec un compteur -1 et si le cpt==0 alors on supprime la clé  """  type\_C = choice(list(cpt\_murs))  func\_code,cpt = cpt\_murs[type\_C]  code = func\_code()  carte = Carte(**False**,**False**,**False**,**False**)  decoderMurs(carte,code)  **if** cpt == 1:  **del** cpt\_murs[type\_C]  **else**:  cpt\_murs[type\_C][1] -= 1  **return** carte |

Le problème des cartes fixes a été réglé en observant que les coordonnées (i,j) de ces cartes étaient tous les deux pairs.

Ensuite, puis générer le code de la pièce tirée, on a décidé de tirer au sort un code qui code une de ces pièces de manière aléatoire (la rotation), ces algorithmes ont été créés grâce à la représentation binaire des cartes.

|  |
| --- |
| **def** **codeangle**():  """  renvoie le code d'un angle de manière aléatoire  """  **return** choice((1,2)) + choice((4,8))  **def** **tout\_droit**():  """  renvoie le code d'un tout droit  """  **return** sum(choice(((1,4),(2,8))))  **def** **code\_jonction**():  """  renvoie le code d'une jonction de manière aléatoire  """   **return** choice((1,2,4,8)) |

**Le test d’accessibilité**

Cette partie est particulièrement ardue, étant donné que le choix fait par notre groupe pour cette fonction c’est de faire un parcours en profondeur du labyrinthe, pour ce faire, nous avons créé une fonction auxiliaire qui vérifiait tous les voisins en vérifiant s'ils sont dans le labyrinthe et s’il existe un passage entre eux.

Pour cette fonction auxiliaire, le choix de la structure de données effectuée est celle d’une liste de tuples avec des tuples à l’intérieur contenant la cardinalité avec l’opération à effectuer pour l’avoir.

|  |
| --- |
| voisins = [('N',(-1,0)),('O',(0,-1)),('E',(0,1)),('S',(1,0))] |

Le positionnement des cardinalités n’est pas anodin, le complément de l’indice de celles-ci avec la longueur de la liste nous renvoie la cardinalité opposée, ce qui nous permet de vérifier s’il existe bel et bien un passage

Nous avons fait le choix de renvoyer un générateur pour la fonction auxiliaire qui test tous les chemins possibles afin de pouvoir nous assurer d’avoir le plus court chemin possible tout en nous assurant que le chemin est bel est bien possible (double vérification).

**listeJoueurs**

Pour listeJoueurs en orienté objet, le choix qui a été réalisé par notre équipe pour la détermination du joueur courant est particulière.

En effet, nous avons opté pour la création d’un générateur qui se répétait à l’infini (à l’aide de cycle de itertools) ce qui nous permettait de ne pas avoir à créer une deuxième liste ou même de modifier des éléments de notre liste pour connaître ce joueur. De plus, le générateur permet de manipuler une structure plutôt légère puisque les éléments ne sont justement générés que sur demande.

## *Les extensions*

Nous n’avons pas vraiment réalisé d’extensions, mis à part peut-être la surcharge d’opérateur pour les classes afin de les rendre plus agréable à l’utilisation, notamment pour les tests en affichant aisément les résultats attendus (pour le plateau c’était vraiment pratique)

# Les Bugs

* **Principaux problèmes rencontrés**

Parmis nos nombreuses erreurs, certaines se démarquent des autres.

D’abord, les problèmes de compréhension, comprendre ce que la fonction attend de nous n’est pas toujours aisé. En effet, certaines fois, nous les aurions faites autrement ou bien même des problèmes de compréhension totale qui se retrouvent à la toute fin lorsque l’on essaie de lancer le labyrinthe

Nous avons ensuite eu certains problèmes avec les fonctions déjà présentes pour la version objet de notre programme. C’est ainsi que nous avons opté pour une réécriture partielle à partir de leur version classique et non de celle qui nous était proposée.

Après avoir connu quelques problèmes dans la distribution des trésors de manière équitable,la matrice et les pointeurs, ainsi ainsi que le débogage du labyrinthe… L’entraide nous a permis d’atteindre une version avec un seul bug récurrent, il arrive de manière très occasionnelle que lors de la sortie du joueur du plateau, il aille sur une autre case que celle à l’opposée.