

Rapport d'Implémentation

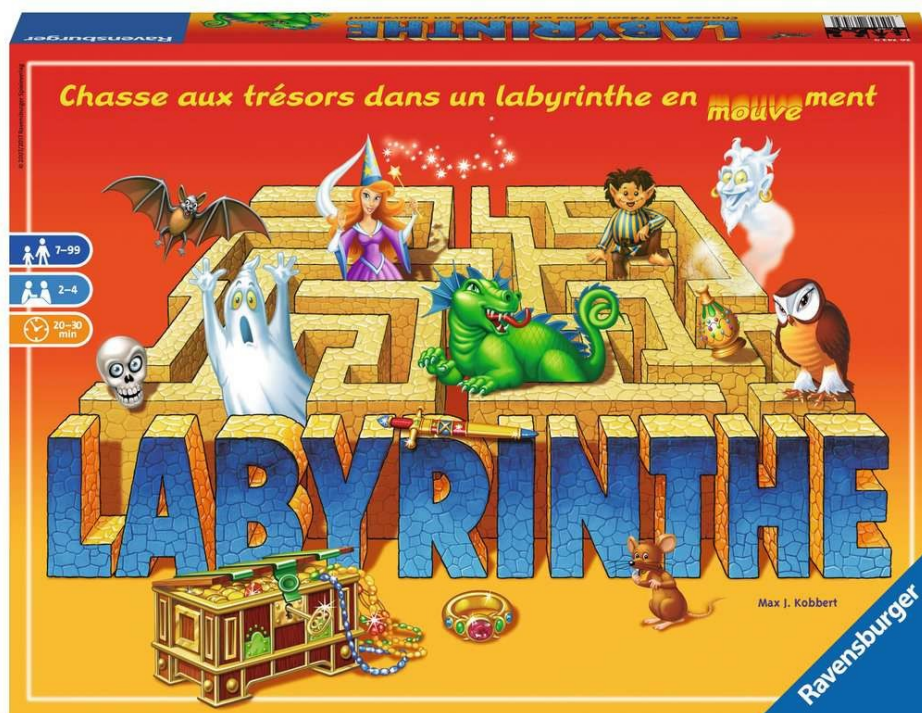
HERPOUX Mattéo

PLEUTIN Benjamin

TURIKUMWE Fabrice



Le jeu du Labyrinthe en C :



Sommaire

I. Introduction.....	2
II. Modifications par rapport à l'analyse.....	3
1. Fonctions.....	3
2. Structures.....	6
III. Choix de développement.....	6
1. Couleur.....	6
2. Affichage.....	7
3. IA.....	7
IV. Tests Unitaires.....	7
1. Module joueur.....	7
2. Module victoire.....	7
3. Module pousser.....	8
4. Module creer_plateau.....	8
5. Module distribuer.....	8
6. Module jouer_tour.....	8
V. Tests d'Intégration.....	8
VI. Tests de Validation.....	9
VII. Réalisation d'une partie :.....	9

I. Introduction

Ce document est le rapport d'analyse d'un projet de programmation en C dont le but est de développer un programme permettant de jouer à un jeu de société: le Labyrinthe.

On devra pouvoir faire des parties de Labyrinthe à partir d'un programme en C, en suivant les règles du Labyrinthe classique. Ici, on développera néanmoins une IA, dans le but de pouvoir jouer avec 2, 3, ou 4 joueurs même si l'on est seul.e à jouer.

Ce rapport comprend d'abord les modifications faites sur le projet final par rapport à ce qu'on avait prévu sur l'analyse du projet, les fonctions dont on a modifié l'utilité, celles qu'on a supprimé, et celles qu'on a ajouté. Ceci sera suivi des choix de développement qui ont entraînés ces modifications.

On présentera finalement les tests unitaires, les tests d'intégration et les tests de validation.

II. Modifications par rapport à l'analyse

1. Fonctions

-Couleur : Le premier changement concerne la couleur. En effet, on avait prévu plusieurs fonctions pour gérer l'affichage de couleur différentes dans le terminal, chose qu'on a décidé de laisser de côté pour des raisons qu'on détaillera plus précisément dans la partie 3. Nous n'utiliseront donc pas les fonctions `demandeur_couleur_joueurs` et `changer_couleur`.

`case_depart` : La fonction `case_depart` était censée retourner 1 si le joueur concerné a atteint tous ses trésors et est retourné à sa case de départ. Ici, elle retourne 1 dès que tous les trésors ont été atteints, pas besoin de retourner à la case de départ.

`Struct ligne` : On avait créé une structure ligne dans l'analyse. On a décidé d'abandonner cette structure qui ne faisait que compliquer les choses. Le plateau n'est donc plus un tableau de lignes mais un tableau 2d de structures Case (on a appelé la structure Case et pas `case` car le mot `case` est une fonction en C). Ce changement concerne beaucoup de choses : dans `creer_plateau`, on ne crée pas de tableau de 12 lignes, et on n'utilise pas la fonction `creer_lignes_mouvantes`. La fonction `est_elle_decalable` qui vérifie si la ligne qu'on veut pousser n'a pas été poussée dans le sens inverse juste avant prend désormais un int en paramètre qui est le numéro de la ligne (entre 1 et 12) . Pour cela on se sert de l'int `ligne_tmp` dans la structure plateau, qui est un int au lieu d'une ligne. La fonction `pousser_rangee` prends en paramètre un int au lieu d'une ligne, de même que `derniere_ligne` qui retourne également un int.

`assignation_indices_cases` : Contrairement à ce qu'on a dit dans la conception détaillée, les indices des cases ne vont pas aller de 0,0 à 7,7, mais bien de 1,1 pour la case en haut à gauche, et 7,7 pour celle en bas à droite. Ces indices sont stockés dans les int hauteur et largeur de la struct Case. Les indices de la case temporaire sont de 0,0. Ces indices sont sous la forme ligne,colonne. On en profite aussi pour assigner aux cases leur numéro sur le plateau.

`creer_tresors` : On utilisera finalement pas cette fonction, au profit de la fonction suivante.

`distribuer_tresors` : Cette fonction s'appelle maintenant `inittresors`, et a entraîné l'implémentation d'une autre fonction. Dans cette fonction, on ouvre le fichier `tresors.txt` qui est placé à la racine du projet, et on va créer le tableau de trésors en leur assignant un nom (pris dans le fichier) et un numéro pour faire les comparaisons. Cette fonction trie aléatoirement les 24 trésors en s'aidant de la fonction `int* tritabalea(int taille)` qui renvoie un tableau d'entier de taille `taille`, composé des numéros de 0 à `taille` dans un ordre aléatoire. Dans `inittresors`, le *i*ème trésor sera à la place `n°tab[i]` dans le tableau de trésors.

`placer_tresors` : Cette fonction prends en deuxième paramètre non plus un tableau de trésors mais un tableau de joueur, et on utilise le tableau de trésors `liste_tresors` de la struct `joueur`, qui a été remplie avec une fonction `trier_tresors` qu'on appelle dans le `main` (c'est une fonction qui n'était pas dans l'analyse, et qui prends en paramètre le tableau de joueur et qui le retourne. Elle utilise `init_tresors` et assigne équitablement la liste de trésors entre les joueurs). Pour placer les trésors sur le plateau aléatoirement, on utilise `tritabalea` avec une taille 49 pour les 49 cases, et on prends les 24 premières cases du tableau pour les positions des trésors (l'int `numcase` de la struct `case`), et on assigne les trésors. On peut avoir des positions qui correspondent à des coins, dans ce cas on prends la 25 ème position du tableau à la place, et ainsi de suite. Cependant, on a besoin d'une nouvelle fonction pour trouver les indices d'une cases a partir de son numéro. On a donc créé `void chercher_case_avec_num(plateau p, int num, int* pi, int* pj)` qui prends en paramètre le plateau, le numéro et deux pointeurs correspondants aux indices. Dans la fonctions, on parcourt chaque case du tableau et on compare les numéros jusqu'à ce qu'on tombe sur la bonne. Les trésors sont donc placés sur le plateau.

`tri_aleatoire_cases_mouvantes` : Dans la conception détaillée, nous parlions de trier aléatoirement un tableau de 34 cases puis de placer les cases en suivant les numéros aléatoires. Cependant, il y a 16 cases fixes dans le plateau qui ne changeront pas. On a donc implémenté une nouvelle fonction `int* transformer_tab(int* tab, plateau p)`, qui prends en paramètre le plateau et un tableau sorti de `tritabalea`. Le but de la fonction va être de transformer le tableau de 34 qui contient des numéros de 0 à 33 en un tableau de 34, mais qui va contenir les 34 numéros de 0 à 48 en omettant les 16 numéros des cases fixes. Une fois qu'on aura ce tableau qui contiendra tous les numéros de cases mouvantes aléatoirement, on suis la conception détaillée. Dedans on nous dit plusieurs fois qu'on prends une direction aléatoire. Pour cela on créé une nouvelle fonction `char orientation_alea()` qui renvoie un caractère aléatoire entre 'h', 'b', 'd', et 'g'. On utilise aussi la fonction `chercher_case_avec_num` comme dans `placer_tresors` pour placer les cases sur le plateau.

`assigner_valeurs_orientations` : Pour chaque case, on remplit les valeurs h, d, g, et b de la struct `Case`, avec 0 ou 1 en fonction de type de la case. Les valeurs correspondent à haut, droite, gauche et bas. Si un chemin part vers le haut sur la case, alors h=1, sinon, h=0. On fait pareil pour les 3 autres directions, pour les 49 cases du plateau, et pour la case temporaire. Cela nous servira pour déplacer le pion.

`tresor_atteignable` : On n'utilisera finalement pas cette fonction.

`est_on_sur_tresor` : Pour vérifier si le joueur est sur le trésor qu'il doit atteindre, on va inclure un nouvel entier dans la struct `joueur` : `nbr_trésors` qui sera le nombre de trésors restants à atteindre par le joueur. Si ce nombre est égal à 0, alors on a atteint tous les trésors et on renvoie 1, 0 sinon.

`tresor_suivant` : On n'utilisera finalement pas cette fonction.

`deplacer_pion` : Petite erreur ici : on ne passe pas le joueur en paramètre par adresse, donc si effectivement on peut déplacer le pion à l'endroit voulu, on renvoie juste 0 et on changera la case du joueur dans le `main`.

`case_compatible` : On utilise donc ici les paramètres h, g, d, et b de la struct `case` pour voir si il y a un chemin entre les deux cases passées en paramètre suivant l'orientation passée également en paramètre.

`peut_on_deplacer_ici` : On utilise ici une récursion. La fonction prends en paramètre 3 arguments, à savoir le plateau, la case où on est , et la case où l'on désire aller. On a donc supprimé le 4ème argument qui était un pointeur sur un tableau de cases. A la place, on écrit deux variables globales : un tableau de int `tabcases` de taille 49 qui contiendra les numéros des cases où l'on peut se déplacer ; et un int `tabcaseint` qui sera le nombre de cases où l'on pourra se déplacer. Au début de la fonction, `tabcaseint` est donc à 0, et on ajoute la case où on est dans la liste des cases où l'on peut se déplacer (`tabcases[tabcaseint]`), et on itère `tabcaseint`. Pour les quatres directions, on va donc vérifier, sous réserve de certaines conditions, comme « est-on au bord du plateau ? », où « à-t-on déjà ajouté ce numéro de case dans `tabcases` avant ? », si la case où on est possède un chemin pour aller à la case à coté (suivant la direction actuelle). Si oui, alors on appelle la fonction `peut_on_deplacer_ici` qui refera toute la manipulation à partir de la case d'à coté (qui deviendra le second argument). Quand on a fini de vérifier les 4 directions, on se retrouve avec un tableau de longueur `tabcaseint` qui contient tous les numéros de cases où l'on peut se déplacer. On vérifie donc si le numéro de la case où l'on veut aller se trouve dedans. Si oui on retourne 1, sinon, 0.

`pion_deplacable` : On ne regarde plus si le pion est entre quatre murs avant de déplacer, car c'est fait automatiquement dans `peut_on_deplacer_ici`, cependant, on réinitialise `tabcases` (toutes les cases à -1) et `tabcaseint` (à 0) dans les deux cas (si on peut déplacer et si on ne peut pas).

Affichage : On parlera des fonctions affichage dans la partie sur les choix de développement. On n'utilisera pas les fonctions `afficher_fleche`, `afficher_tresor_sur_case`, `afficher_murs`, et `afficher_pion`.

`est_elle_decalable` : Le paramètre de la struct `plateau` `ligne_tmp` est devenu un int.

`pousser_rangee` : La fonction prends maintenant en paramètres le numéro de la ligne à pousser, la plateau qu'elle renvoie également, et un pointeur sur le joueur actuel. On suis ensuite globalement la conception détaillée (on sépare la fonction en 4 cas, si on pousse vers la gauche, droite haut ou bas) à une exeption près : on change le paramètre `c` du pointeur sur joueur pour changer sa case. Si un poin se trouve sur la case éjectée du plateau, il se place sur la case opposée, contrairement à un trésor qui peut se trouver sur la case temporaire. Si on essaye de pousser la ligne opposée à celle que l'on a poussé avant, on a un message qui nous le dit et on ne peut plus pousser de ligne ce tour-ci.

`remplacer_case` : On n'utilisera finalement pas cette fonction.

`yatil_des_pions` : On n'utilisera finalement pas cette fonction.

`jouer_tour_ia` : Afin de permettre à l'IA de jouer son tour, deux fonctions assez denses ont été créées: `case_proche_tresor` (qui peut également être utilisée pour un humain pour une aide supplémentaire) se présente comme une série de tests et d'incrémentations/décrémentations faits pour que l'on trouve la case accessible au pion qui aura le moins de cases d'écart avec le trésor sur le plateau. Elle est écrite de manière à commencer les tests une fois que `i` et `j` ont les valeurs des coordonnées du trésor, et à pouvoir couvrir tout le plateau.

`analyse_coups_possibles_IA` se décompose en 4 parties:

- L'algorithme de la première partie permet, avec un double for ligne-sens, de tester tous les coups possibles en un tour (48 coups possibles) et savoir si le trésor est atteignable. Si le coup n'est pas retourné (si le trésor n'est pas atteignable), le programme continue vers la partie suivante.
- L'algorithme de la seconde partie permet, dans une quadruple itération (ligne, sens, tour 1, tour 2), de tester tous les coups possibles en deux tours (48x48 coups) pour savoir si le trésor sera atteignable au second tour. Si le coup n'est pas retourné, le programme continue vers la partie suivante.
- L'algorithme de la troisième partie permet, dans une triple itération(joueurs, ligne, sens), de tester tous les coups possibles en un tour de chaque autre joueur pour savoir s'ils pourront atteindre leur trésor, et de retourner le coup qui empêchera tous les autres joueurs d'accéder à leur trésor. Si le coup n'est pas retourné, le programme continue vers la partie suivante. Cet algorithme n'est finalement pas fonctionnel.
- L'algorithme de la dernière partie permet de jouer un coup au hasard, c'est l'algorithme par défaut.

2. Structures

Des structures ont donc changé :

tresor :

son nom char* nom
son numéro int numero (on a enlevé la case où il est)

joueur :

son nom char* pseudo (on a enlevé sa couleur)
si c'est une ia int ia
son numéro int numero
son nombre de trésors restants int nbrtresors
la case où iel se trouve Case c
la liste des trésors restants tresor* listetresors

plateau :

le tableau de cases Case tabcases[9][9] (on a enlevé les lignes)
la case temporaire Case case_tmp
le numéro de la ligne qu'on vient de bouger int ligne_tmp

Case :

le trésor qu'il y a dessus tresor tresor
son type char type[2] (on a supprimé le numéro de la ligne)
ses valeurs de directions int h,b,g,d
ses indices dans le plateau int hauteur,largeur
son numéro de case int numcase

On a aussi une nouvelle structure coupsIA :

la case de départ Case init
le numéro de la ligne que l'IA va bouger durant son tour int ligne
le sens de la case qu'elle va insérer char sens
la case de destination qui sera déterminée à l'aide de la fonction Case
dest
un int qui prendra la valeur retournée par est_on_sur_tresor int atteint

III. Choix de développement

1. Couleur

Nous devons à la base attribuer des couleurs à chaque joueurs, et nous avons prévus deux fonctions pour cela en lien avec la couleur ; ce ne sera au final pas fait principalement par manque de temps.

2. Affichage

Nous avons une multitude de manières différentes de gérer l'interaction entre les joueurs et joueuses et la machine. Nous avons au final choisi de ne pas représenter directement les pions et les trésors sur le plateau, et d'afficher, après chaque affichage du plateau, la position du joueur ou de la joueuse concerné.e et celle du trésor actuel à atteindre sous la forme d'indices. Par exemple, la personne qui joue en premier dans le jeu se trouve au départ sur la case d'indice 1,1 (ligne,colonne), ce qui correspond à la case en haut à gauche. Nous n'affichons pas les numéros de lignes et de colonnes à coté du plateau, car nous affichons déjà avec des flèches les rangée de cases que nous pouvons pousser avec leur numéro. Ce n'est pas la manière la plus agréable de jouer au jeu mais c'est assez lisible pour en faire abstraction.

IV. Tests Unitaires

Pour faire les tests unitaires nous avons utilisés la bibliothèque de test [Unity](#), qu'on a placé à la racine du projet. Nous avons créé un fichier de test par module, rangés dans le dossier tests, et un fichier principal test.c à la même place où on exécute tous les tests.

1. Module joueur

On demande à l'utilisateur de rentrer des joueurs tests (seule fois où l'utilisateur agit pendant les tests) et on a un premier test qui vérifie que le nombre de joueur est bien compris entre 2 et 4 compris.

On a ensuite un deuxième test qui vérifie que le nombre d'ia est strictement inférieur au nombre de joueurs.

Le troisième test vérifie que le bon numéro à bien été attribué à chaque joueur et joueuse.⁷

⁷ Herpoux Mattéo - Pleutin Benjamin - Turikumwe Fabrice

Le quatrième vérifie que le paramètre ia de joueur a bien été changé pour chaque ia.

On a enfin la fonction mainfjoueur qui lance tous les tests précédents.

2. Module victoire

On n'a qu'un seul test, qui vérifie que la fonction case_depart renvoie bien 0 sur un joueur non initialisé, et le mainfvictoire qui lance le test.

3. Module pousser

On n'a là aussi qu'un seul test, mais plus complexe : on crée un tableau de 2 joueur qu'on laisse non initialisé (car on ne s'en servira pas dans la fonction mais on en a besoin en paramètre), et un plateau qu'on initialise avec creer_plateau. On pousse la rangée numéro 1, et on vérifie pour toutes les cases de la rangée qu'elle ont bien bougé d'un cran. On répète ça pour les rangées 4, 7 et 10 en réinitialisant le plateau à chaque fois.

4. Module creer plateau

On a un premier test qui vérifie que la fonction creer_tresors ne renvoie pas rien.

On a ensuite un second test qui exécute la fonction placer_tresors, qui parcourt le plateau en comptant les tresors et qui vérifie qu'il y en a bien 24.

Le troisième test vérifie le fonctionnement de transformer_tab, elle regarde si il ne reste pas de numéro de case correspondant à une case qui ne peut pas bouger.

Le quatrième test vérifie que orientation_alea renvoie bien une des 4 orientation voulue à savoir h, g, b ou d. Le test s'exécute 50 fois pour être sûr.

Le dernier test vérifie que assignation_indices_cases assigne les bon indices pour chaque cases.

Enfin le mainfcreer_plateau qui lance ces 5 tests.

5. Module distribuer

Le premier test vérifie que tritabalea ne renvoie pas de pointeur NULL.

Le second test vérifie que inittresors ne renvoie pas de pointeur NULL.

Le mainfdistribuer lance les 2 tests.

6. Module jouer tour

Le premier test vérifie le fonctionnement de trouver_case_avec_tresor, il crée un plateau et le parcourt 24 fois pour trouver les 24 cases, et on teste si la fonction a bien trouvé 24 cases.

Le second test vérifie le bon fonctionnement de case_compatible, en prenant pour exemple la case 2,1 d'un plateau pour la direction droite (car on connaît le type de la case de droite).

Le mainfjouer_tourt lance ces fonctions.⁸

On a exécuté tous les main des fichiers tests dans le fichier test.c.

V. Tests d'Intégration

On va faire ici un parallèle avec l'analyse descendante réalisée pendant la phase d'analyse. Elle est trouvable sur le rapport d'analyse et sur [le projet framagit](#).

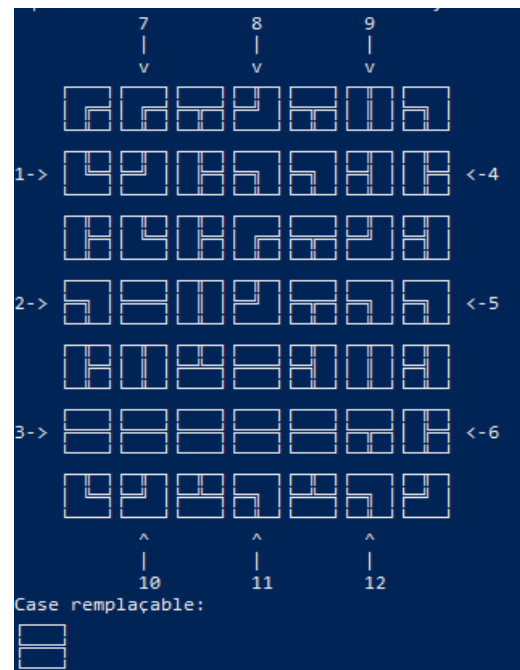
Dans le graphique de l'analyse descendante, nous avons divisé le projet en plusieurs parties : les joueurs, la victoire, la création du plateau, et la gestion du tour. Nous avons gardé les deux premières parties, mais nous avons divisé la partie sur le tour en trois autres parties, l'affichage, la gestion du décalage d'une rangée de cases, et le reste des fonctions pour jouer le tour, à savoir les fonctions pour déplacer le pion et les fonctions sur l'IA. Nous avons également extrait les fonctions pour distribuer les trésors de la création du plateau pour en faire une partie à part entière. Nous arrivons donc à un total de 7 parties principales, plus le fichier main, qui est relativement long (+150 lignes) .

Commençons par le module « afficher » :

1. Afficher

Le programme affiche comme prévu, les cases du plateau de façon lisible, avec des flèches et des numéros des quatres cotés du plateau pour représenter les 12 rangées décalables. La case temporaire est aussi affichée en dessous du plateau. En revanche, les pions et les trésors ne sont pas directement affichés sur leurs cases respectives.

Par conséquent, rien sur le plateau ne s'affiche en couleur. Tout s'affiche en blanc.



2. Pousser

Quand on choisit de pousser une rangée, on nous demande d'abord le numéro de la rangée que l'on veut pousser, et ensuite, on nous montre un exemple de ce que donnerai la case temporaire penchée dans chacune des quatre directions dans laquelle on a la possibilité de l'insérer dans le plateau. On nous demande donc de rentrer une direction sous la forme d'un caractère, puis la rangée se décale. La case qui sort du plateau devient la nouvelle case temporaire, et l'ancienne case temporaire est insérée dans le plateau selon l'orientation déterminée par le joueur ou la joueuse.

3. Joueur

Le programme demande tout d'abord le nombre de joueurs et de joueuses, puis le nombre d'IA qui joueront, et enfin leur pseudonyme.

4. Distribuer

Le programme récupère d'abord le nom des trésors dans le fichier `tresors.txt` placé à la racine du projet, puis les trie aléatoirement, et les distribue à part égale aux joueurs et joueuses, de telle sorte qu'à chaque personne puisse être associée une liste de trésors propre.

5. Créer plateau

Le programme commence par assigner les bons types de cases aux seize cases fixes présentes sur le plateau, qui ne changeront pas de type ni de place au cours de la partie. Puis il assigne aux 49 cases du plateau leurs indices hauteur et largeur (numéro de ligne et numéro de colonne), qui seront notamment utilisés pour communiquer avec les joueurs et joueuses, puisque les pions et les trésors ne s'afficheront pas directement sur le plateau, il faudra leur dire où ils sont. Troisièmement, le programme va répartir aléatoirement sur le plateau les 34 cases de type et de position aléatoire, puis leur assigner les valeurs correspondantes à leur type qui vont permettre au programme de savoir où un pion peut se déplacer sur le plateau. Enfin, le programme va disposer les 24 trésors aléatoirement sur le plateau, c'est-à-dire leur assigner une case à chacun, en évitant les coins.

6. Jouer tour

Contrairement aux autres modules, il n'existe pas de fonction « principale » nommée « jouer_tour ». Il y en a plutôt deux dans ce module, la fonction `deplacer_pion` et la fonction `jouer_tour_ia`. La première cartographie le plateau au moment où elle est utilisée pour connaître toutes les cases où il est possible de se déplacer, comme ça elle peut facilement savoir si il est possible de déplacer le pion à l'endroit voulu. Concernant la seconde : D'une part, la condition qui fait que l'IA n'insère une case et ne bouge une ligne que quand cela lui permet de se déplacer d'au moins une case, n'a pas été implémentée. Au lieu de cela, l'IA va, par défaut et souci de simplicité, bouger une ligne avant de bouger son pion. D'autre part, le test pour savoir si chaque coup testé bouche le chemin d'un ou plusieurs joueurs, n'a pas été implémenté non plus, car cela aurait rendu l'IA bien trop forte.

7. Victoire

Ce module est très court, il ne contient que trois petites fonctions, qui consistent à vérifier si le joueur ou la joueuse en question n'a plus de trésor à aller chercher, alors on stoppe la boucle du tour et la partie s'arrête. Un message de victoire s'affiche alors. C'est la différence par rapport à ce qu'on avait prévu : il n'y a pas besoin de retourner à son point de départ quand on a trouvé tous nos trésors.

VI. Tests de Validation

Au final, nous avons un Labyrinthe fonctionnel, qui remplit la plupart des besoins que l'on avait défini au début de l'analyse. En suivant ligne par ligne [les spécifications](#) nous avons ceci :

1. Identifier les joueurs et les joueuses

Les joueurs et les joueuses peuvent choisir leur pseudo, mais pas leur couleur, on ne peut donc pas changer le couleur des écritures sur le terminal.

L'IA n'est ni trop stupide, ni infaillible, elle vérifie si elle peut atteindre le trésor de là où elle est, elle bouge bien une ligne uniquement pour se déplacer, sinon elle va chercher à embêter les autres.

A chacun.e est associé un pion, c'est-à-dire un numéro, un pseudo, mais pas une couleur.

2. Créer le plateau

On place aléatoirement toutes les cases, on a défini tous les différents types de cases, et douze lignes uniquement peuvent se déplacer. Nous avons également créé les trésors et les avons placés sur le plateau.¹¹

11 Herpoux Mattéo - Pleutin Benjamin - Turikumwe Fabrice

3. Jouer un tour

On affiche bien le plateau, avec les flèches autour et leur numéros associés pour les lignes mouvantes. On affiche donc bien une case, mais on affiche pas de trésors sur la case, ni de pion, mais on voit clairement à l’affichage où sont les murs et vers où partent les chemins accessibles.

On affiche également la case temporaire en dessous du plateau.

On peut pousser une rangée, mais on a pas besoin de vérifier que c’est une des douze lignes décalables car les lignes non décalables n’ont finalement pas de numéros. On vérifie bien que la ligne n’a pas été décalée dans le sens inverse juste avant car on la garde en mémoire. On insère bien la case temporaire dans la plateau dans l’orientation de notre choix, et nla case qui sort du plateau devient la nouvelle case temporaire. Si des pions sont placés sur la case qui sort du plateau, ils sont automatiquement replacés sur la case opposé sur la plateau, celle qui vient de rentrer.

On a bien la possibilité de déplacer son pion sur le plateau, et quand on le fait, le programme lie bien toutes les cases possible pour le déplacement avant de vérifier que la case où l’on veut se déplacer soit présente dedans.

On affiche chaque manipulation du joueur ou de la joueuse actuelle, humain.e ou IA, les déplacement et les changements sur le plateau.

On regarde a chaque fin de déplacement si on est sur une case trésor, et si oui, on le dit et on passe bien au trésor suivant dans la liste, mais on a pas besoin de retourner à son point de départ pour gagner la partie.

4. Stopper la partie lorsque quelqu’un a gagné

A chaque fin de tour, on regarde bien si le joueur ou la joueuse actuel.le possède encore des trésors à aller chercher, et sinon, alors on stoppe le programme et on affiche le message de victoire en conséquence. Encore une fois, on ne vérifie pas si on est retourné sur notre case de départ.

VII. Réalisation d'une partie

Nous allons faire une partie de Labyrinthe avec une IA et un joueur :

On rentre les paramètres :

```
Combien de joueur.euse.s dans cette partie?
2
Combien il y a-t-il d'IA?
1
Quelle est le pseudo du joueur 1?
IA
Quelle est le pseudo du joueur 2?
Aifé
```

L'IA commence a jouer :

```

Début de la partie:
C'est à l'IA IA de jouer:
IA a pour objectif actuel d'atteindre le trésor Sir Alonne
qui se trouve sur la case d'indice 4,5 et IA se trouve actuellement sur la case d'indices 1,1.
      7      8      9
      |      |      |
      v      v      v
  1->  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817  818  819  820  821  
```

L'IA a vu qu'elle pouvait atteindre son trésors en un coup, elle a donc poussé la bonne ligne, et s'est déplacée sur le trésor.¹³

C'est maintenant à Aifé de jouer :

```

C'est à Aifé de jouer:
Aifé a pour objectif actuel d'atteindre le trésor Chaos Witch Quelaag
qui se trouve sur la case d'indice 3,4 et Aifé se trouve actuellement sur la case d'indices 1,7.

      7       8       9
      |       |       |
      v       v       v
  1->  [ ][ ][ ][ ][ ][ ][ ] <-4
      [ ][ ][ ][ ][ ][ ][ ]
  2->  [ ][ ][ ][ ][ ][ ][ ] <-5
      [ ][ ][ ][ ][ ][ ][ ]
  3->  [ ][ ][ ][ ][ ][ ][ ] <-6
      [ ][ ][ ][ ][ ][ ][ ]
      ^       ^       ^
      10      11      12
Case remplaçable:
[ ]
Aifé veut-il déplacer son pion(tapez 0) ou pousser une rangée(tapez 1)?
0
Ou voulez-vous déplacer votre pion? Rentrez les indices de la case.
2
6
Aifé est désormais sur la case d'indice 2,6, et le trésor qu'il doit atteindre sur la case d'indice 3,4.

```

(le plateau s'affiche ici)

```





Avez-vous terminé de déplacer votre pion? Oui->1, Non->0
1
Aifé veut-il pousser une ligne(tapez 0) ou non(tapez 1)?
0
Entrez le numéro de la ligne que vous voulez pousser:
4
Orientation n(nord):
Case remplaçable:
[ ]
Orientation e(est):
Case remplaçable:
[ ]
Orientation s(sud):
Case remplaçable:
[ ]
Orientation o(ouest):
Case remplaçable:
[ ]
Entrez l'orientation que vous voulez attribuer à la case temporaire:
0
Vous n'avez pas le droit de pousser la ligne inverse de celle qui a été poussée juste avant.

```

Ici, Aifé a fait une erreur et a tenté de pousser la ligne inverse de celle que l'IA a poussé juste avant. La ligne n'est donc pas poussée, et Aifé ne peut plus pousser de ligne pour ce tour. Un tour se poursuit sans que ni Aifé ni l'IA ne trouve son trésor.

```
Aifé veut-il déplacer son pion(tapez 0) ou pousser une rangée(tapez 1)?
0
Ou voulez-vous déplacer votre pion? Rentrez les indices de la case.
3
4
Le pion ne peut pas être déplacé ici.
Rentrez à nouveau les indices:
2
6
Aifé est désormais sur la case d'indice 2,6, et le trésor qu'il doit atteindre sur la case d'indice 3,4.
```

Aifé tente ici de déplacer son pion sur une case où il ne peut pas aller. Il est donc obligé de reentrer les indices d'une case disponible (la case où il est actuellement si il n'y en a aucune).

```
Avez-vous terminé de déplacer votre pion? Oui->1, Non->0
1
Aifé veut-il pousser une ligne(tapez 0) ou non(tapez 1)?
0
Entrez le numéro de la ligne que vous voulez pousser:
9
Orientation n(nord):
Case remplaçable:

Orientation e(est):
Case remplaçable:

Orientation s(sud):
Case remplaçable:

Orientation o(ouest):
Case remplaçable:

Entrez l'orientation que vous voulez attribuer à la case temporaire:
o
On pousse la rangée 9:
```

(Affichage de plateau)

```
Aifé est désormais sur la case d'indice 3,6, et le trésor qu'il doit atteindre sur la case d'indice 3,4.
Aifé veut-il déplacer son pion(tapez 0) ou terminer son tour?(tapez 1)?
0
Ou voulez-vous déplacer votre pion? Rentrez les indices de la case.
3
4
Aifé a trouvé le trésor Chaos Witch Quelaag
Aifé doit maintenant aller chercher le trésor Ornstein and Smough
qui se trouve sur la case d'indice 5 2
Il reste désormais à Aifé 11 trésors à trouver.
Aifé est désormais sur la case d'indice 3,4, et le trésor qu'il doit atteindre sur la case d'indice 5,2.
```

Ici, Aifé est tombé sur son trésor. Il lui en reste donc 11.

```
Avez-vous terminé de déplacer votre pion? Oui->1, Non->0
0
Ou voulez-vous déplacer votre pion? Rentrez les indices de la case.
5
3
Aifé est désormais sur la case d'indice 5,3, et le trésor qu'il doit atteindre sur la case d'indice 5,2.
```

Aifé a le droit de bouger son pion autant qu'il le veut avant de terminer son tour, avant ou après qu'il ait poussé la rangée.

```
L'IA IA a poussé la rangée 1, puis a déplacé son pion sur la case 3 6
IA a trouvé le trésor Velstadt, the Royal Aegis
IA doit maintenant aller chercher le trésor Pontiff Sulyvahn
```

Après quelques tours, l'IA a trouvé son premier trésor. (en beaucoup de temps car elle essayait de pousser la ligne où le trésor était alors que l'IA était aussi dessus).

Si on se trompe dans les input du programme, qu'on nous demande 0 ou 1 par exemple et qu'on rentre 2, on doit rerentrer un chiffre ; pareil pour l'orientation de la case temporaire qu'on insère dans le sens qu'on veut. On ne le voit pas sur les photos, mais le plateau s'affiche régulièrement pour qu'on voie bien ce que l'on fait.

La partie se poursuit ainsi jusqu'à ce que l'IA ou Aifé trouve tous ses trésors.