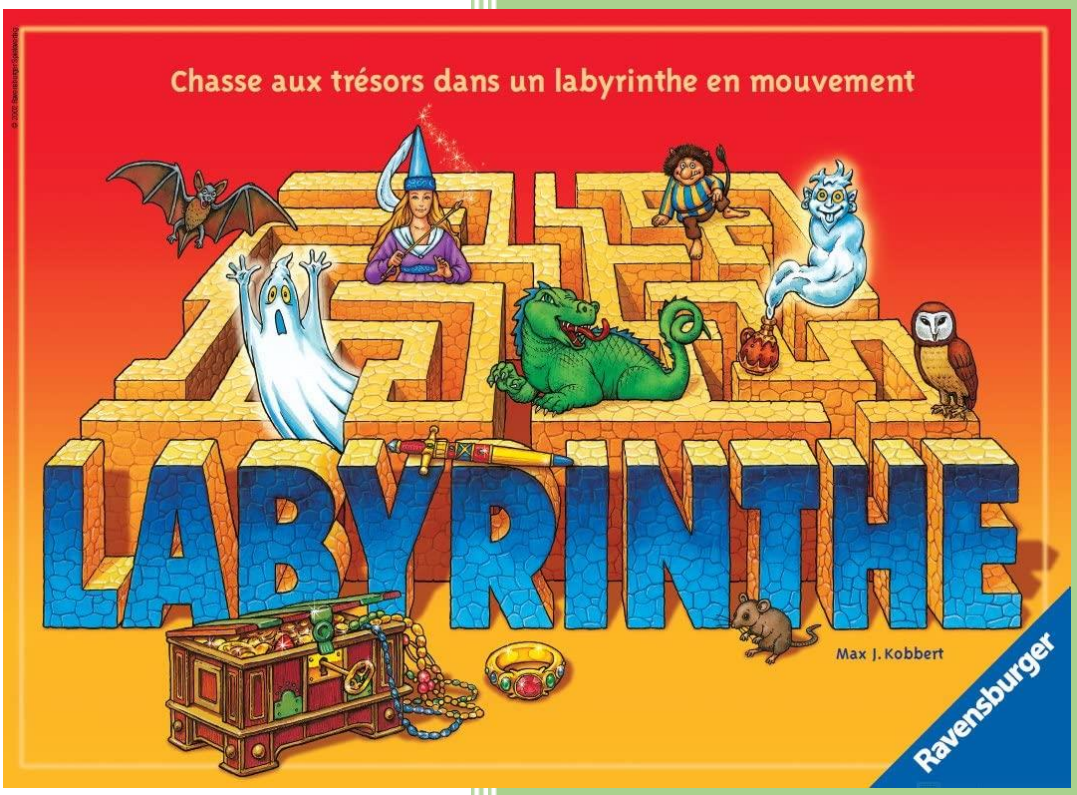
UNIVERSITÉ PERPIGNAN VIA DOMITIA

2021

Rapport d'analyse : Labyrinthe Ravensburger



Yu-Gi-Oh

HAMICHE Mohamed

ELLOUZATI Mohamed

YAYA Ahmed

04/03/2021

Sommaire :

I. Analyse des besoins :

A. Présentation du jeu :

Introduction :

Le labyrinthe de Ravensburger est un jeu de société qui consiste à aller chercher des trésors dans un labyrinthe en mouvement, pour avoir une idée claire du jeu, et pour bien le comprendre, nous avons décidé de l'acheter, et après une lecture attentive de la notice, et le déroulement de plusieurs parties entre nous, nous avons a pu conclure le suivant :

Règles du jeu :

La partie se déroule sur un plateau carré 7 x 7, ce plateau est composé de :

- 33 tuiles non fixes : 12 tuiles "ligne droite", 16 "tuiles tournant", 5 tuiles "ligne droite" + "tournant".
- 16 tuiles fixes : 0 tuiles "ligne droite", 4 tuiles "tournant", 12 tuiles "tournant" + ligne droite".

Dans un premier temps, il faut mettre en place le plateau de jeu. Pour ce faire, il vous faut mélanger les tuiles qui représentent les couloirs du labyrinthe vu du dessus. Ensuite, il faut les placer sur les emplacements libres du plateau de jeu de manière à créer un labyrinthe aléatoire. Une fois le labyrinthe créé il doit vous rester une tuile. Cette tuile servira à déplacer les couloirs en les faisant coulisser.

Avant de commencer la partie, les joueurs doivent se distribuer toutes les cartes trésors, face cachée, de façon à ce qu'ils aient tous le même nombre de cartes. Chaque joueur doit empiler ses cartes devant lui sans les regarder. Chaque joueur choisit son pion et le place sur la case correspondante à la couleur de celui-ci (dans les 4 coins du plateau de jeu).

C'est le joueur le plus jeune qui commence à jouer et la partie se poursuit dans le sens des aiguilles d'une montre. Chaque joueur regarde la carte trésor située en haut de sa pile sans la dévoiler aux autres. Il faut savoir que le tour d'un joueur se déroule toujours de la même manière.

En effet, le joueur doit toujours déplacer une rangée ou une colonne en premier, en introduisant la tuile supplémentaire. Ensuite il peut déplacer son pion pour essayer d'atteindre son objectif « **trésor** ». Notez qu'un joueur est toujours obligé de modifier le labyrinthe avant de pouvoir déplacer son pion et ce même si son « **trésor** » est accessible directement.

Pour remporter une **partie au Labyrinthe,** le joueur doit avoir découvert tous ses trésors et doit revenir à son point de départ. Une fois de retour à son point de départ il est déclaré vainqueur et la partie est terminée.

B. Reformulation du problème :

Jouer au labyrinthe Ravensburger : (les étapes)

- Définir le mode de jeu (1 vs 1 ou 1 vs machine).
  Mode 1 vs 1 :
- Poser le plateau :
  1. On remarque qu'il y a des tuiles fixes.
  2. Remplir le plateau avec les tuiles couloir au hasard.
  3. Il reste une tuile en dehors du plateau.
- Attribuer un pion à chaque joueur.
- Poser les pions sur les coins respectifs des joueurs.
- Distribuer les cartes trésor : chaque joueur a une pile de cartes face cachée.
- Qui commence la partie ?
  L'humain si le mode de jeu est 1 vs machine, si 1 vs 1, attribuer au hasard.
- Le joueur 1 à la tuile à jouer en main.
- Le joueur 1 regarde la tête de sa pile.
- Le joueur 1 doit choisir parmi les 12 couloirs où insérer la tuile qu'il a en main.
  Le couloir choisi ne doit pas être l'opposé du coup précédent.
- Une autre tuile sort de l'autre cote du couloir.
- Cette tuile passe à la main du joueur suivant (joueur 2).
- Le joueur 1 choisit de déplacer son pion ou pas.

- Vérifier si le coup est valable : si le joueur n'a pas franchi un mur.
- Si joueur 1 a atteint son trésor :
  1. Il dépile la tête de ses cartes trésor.
  2. Il regarde son prochain objectif.
- C'est au tour du joueur 2.
- Si un joueur a atteint tous ses trésors : (sa pile est vide)
  Son objectif n'est plus d'atteindre un trésor, il doit ramener son pion au point de départ pour gagner la partie.
- La partie est terminée si un des joueurs a dépilé toutes ses cartes et est revenu à son point de départ.

II. Spécifications :

Maintenant qu'on a reformulé le problème, on va le décomposer en sous-problèmes et proposer des solutions avec un langage assez simpliste, ce qui va nous permettre de réaliser une analyse descendante.

1- Mode de jeu :
   - Solutions :
     - Humain Vs Humain
     - Humain Vs Machine
   - Sous-problèmes :
     - Choisir des pseudos.
     - Attribution des pions

2- Poser le plateau :
   - Initialisation du plateau, avec les tuiles fixes.
   - Distribution aléatoire des tuiles couloirs.
   - Mémoriser la tuile restante.

3- Distribution des cartes :
   - Attribuer aléatoirement 12 cartes trésor à chacun des 2 joueurs.
   - Le premier élément de la pile pointe vers la case de départ, suivi des 12 cartes trésor.

4- Le prochain trésor à trouver :
   - Afficher la première carte de la pile.

5- À qui le tour ?
   - Si la partie n'a pas encore commencé, donner tour au hasard.
   - Sinon, alterner :

   Pour passer d'un joueur à l'autre, il faut que le coup précédent de l'autre joueur soit valide.

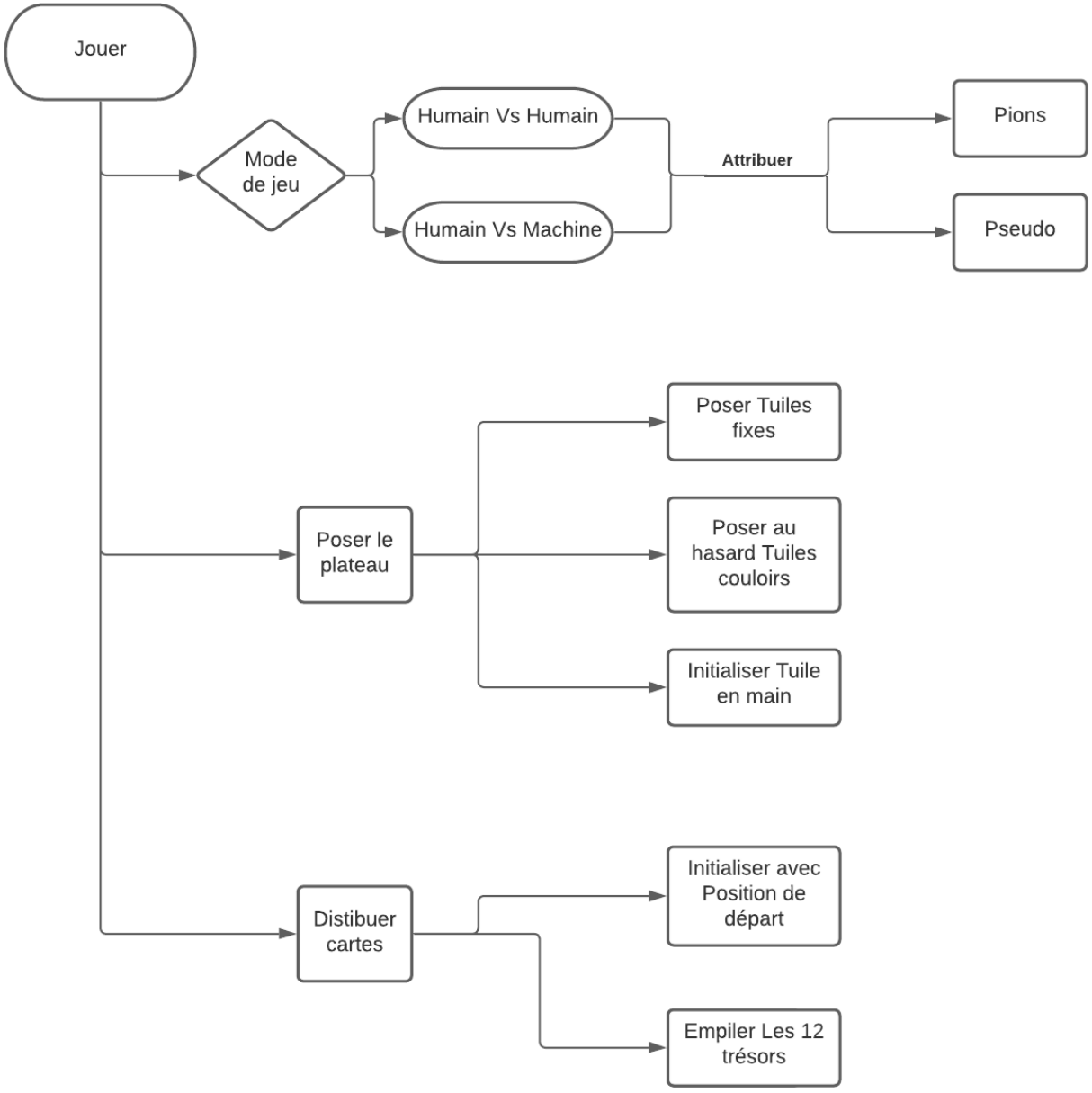
6- Insertion de la tuile :
   - Choix du couloir (Impossible de l'endroit où la tuile est sortie)
   - Choix de l'orientation (de la tuile : horizontalement ou verticalement etc.…)
   - Décalage d'une case du couloir dans le sens de l'insertion
   - Mettre à jour la tuile sortante.

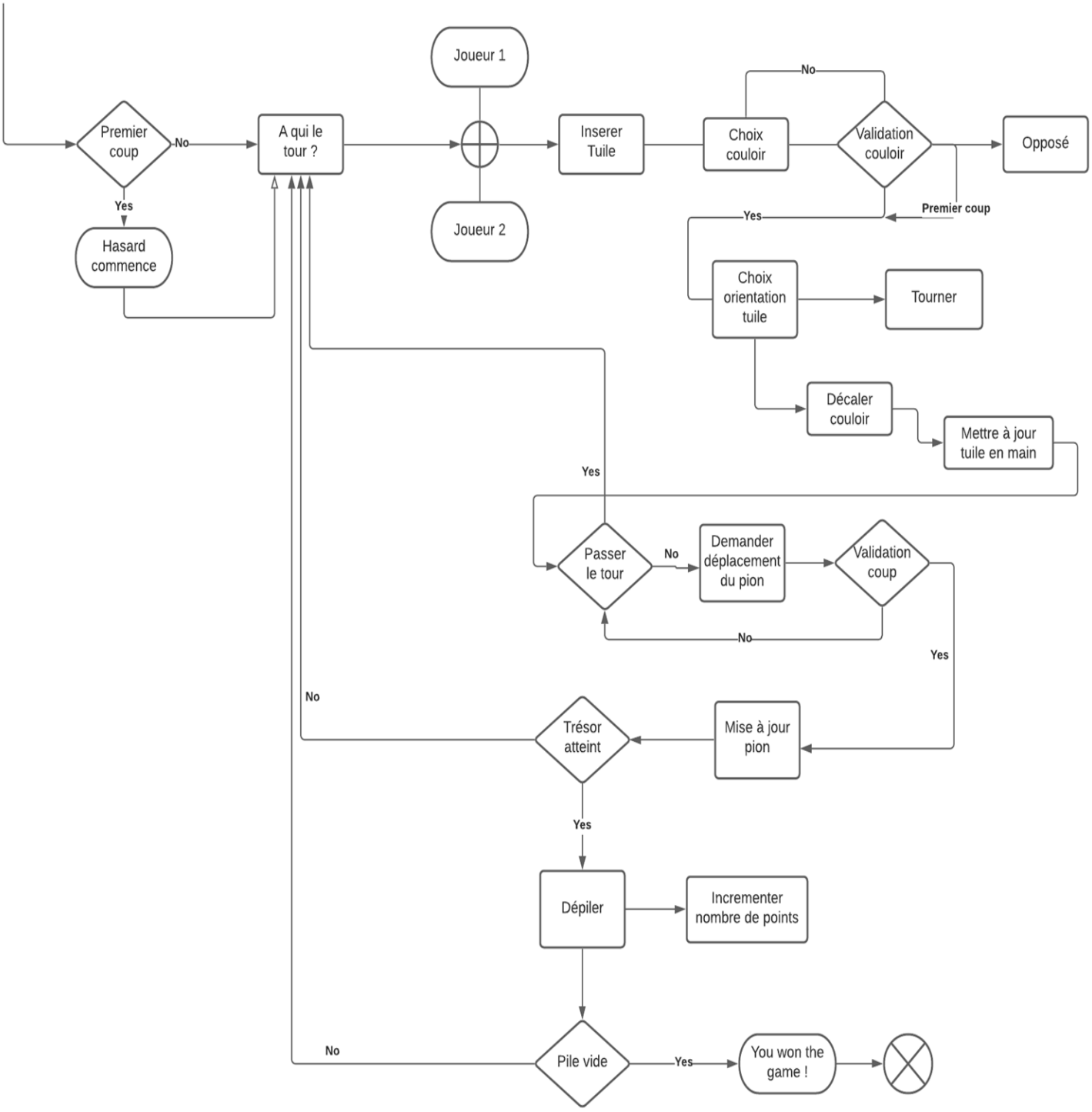
7- Déplacement du pion :
   - Choix :
     - Passer le tour : ne pas déplacer, valider le coup.
     - Déplacer : donner les coordonnées du déplacement.
   - Validation du coup : existence d'un chemin valide entre la position initiale du joueur, et la position où il souhaite aller, et si ce chemin n'existe pas, lui demander de réessayer.

8- Vérification si le trésor atteint :
   - Si le trésor est atteint dépiler le trésor de la pile du joueur correspondant.

9- Si ma pile est *vide* j'ai **gagné**.

A. Analyse descendante :




Premier coup
No
Yes
Hasard commence
A qui le tour ?
Joueur 1
Joueur 2
Inserer Tuile
Choix couloir
No
Validation couloir
Opposé
Premier coup
Yes
Choix orientation tuile
Tourner
Décaler couloir
Mettre à jour tuile en main
Yes
Passer le tour
No
Demander déplacement du pion
Validation coup
No
Yes
No
Trésor atteint
Mise à jour pion
Yes
Dépiler
Incrementer nombre de points
No
Pile vide
Yes
You won the game !

III. Conception préliminaire :

Pour ce projet on a opté pour la programmation structurée (organisation hiérarchique simple des fonctions, qui ne dépassent pas un certain nombre de lignes et sont donc assez courtes).

L'objectif est de lister les fonctions qu'on va rappeler dans notre algorithme.

A. Les structures :

```
struct coordonnees
  entier x
  entier y

struct Tuile
  booléen fixe // 1 fixe 0 couloir
  entier trésor //0 si pas de trésor, n sinon avec n entre 1 et 24
  booléen gauche, droite, haut, bas //0 si mur, 1 si ouverture
  booléen posée //si la tuile est posée ou pas sur le plateau

Tuile Plateau [7][7] // Matrice de tuiles

struct Joueur
  string pseudo
  coordonneés postion_actuelle
  struct pile pile_trésor
  entier nombre_de_points
-------------------------------------------------
```

B. Liste des fonctions :

- Entier modeDeJeu()
- String lire_pseudo()
- initPosition(Joueur P1, Joueur P2)
- distribuerCartes(Joueur P1, Joueur P2)
- Joueur quiCommence (Joueur P1, Joueur P2)
- initPlateau(Tuile Plateau[7][7], Tuile tuile_en_main)
    - poserTuilesFixes(Tuile Plateau[7][7])
    - poserTuilesCouloir(Tuile Plateau[7][7],tuile tuile_en_main)
- alterner(Joueur p1, Joueur p2,Joueur joueurActuel)
- insererTuile (Tuile plateau[7][7],Tuile tuile_en_main,coordonnées)
    - coordonnées choixCouloir()
    - booléen validationCouloir(coordonnées choix_actuel,coordonnées choix_precedent)
        - booléen opposé(coordonnées a, coordonnées choix_precedent)
    - choixOrientationTuile(Tuile tuile_en_main)
        - touner(Tuile tuile_en_main, Entier nombre_de_tours, char direction)
    - decalerCouloir(Tuile Plateau, coordonnees choixCouloir, Tuile tuile_en_main)
- choixDeplacement(joueur P1, Tuile plateau[7][7])
    - booléen validationCoup(Tuile case,coordonnees a,coordonnees choix,Entier compt)
- tresorAtteint(Joueur P1, Tuile Plateau[7][7])

IV. Conception détaillée :

Dans cette étape nous allons développer la conception préliminaire sur le plan algorithmique et structures de données en allant le plus loin possible dans les détails, c'est-à-dire écrire les pseudocodes des fonctions et relier entre elles avec des algorithmes, pour cela on va utiliser des instructions de bases et des structures conditionnelles (Si, Sinon, Boucles Pour …).

A. Pseudo codes des fonctions :

```
Entier modeDeJeu()
//demande au joueur de jouer contre joueur ou de jouer contre machine
DEBUT
  Entier choix
  Affiche(Quel mode voulez vous ? 1- JvsJ ou 2-JvsMachine)
  Lire(choix)
  si choix = 1
    retourner 1
  fin si
  sinon
    retourner 2
FIN
----------------------
String lire_pseudo()
DEBUT
  afficher(donner votre pseudo)
  lire(pseudo)
  retourne pseudo
FIN
----------------------
initPosition(Joueur P1, Joueur P2)
DEBUT
  P1.x <- 0
  P1.y <- 0
  P2.x <- 6
  P2.y <- 6
FIN
```

```
----------------------
distribuerCartes(Joueur P1, Joueur P2)
DEBUT
  Entier i <- 0
  Entier j <- 0
  Tant que i < 24
  Debut
    tableau[i] <- remplir aléatoirement avec un entier entre 1 et 24
    j <- 0
    Tant que j < i:
// vérifie si la valeur entrée n'est pas égale à une valeur déjà entré
//auparavant dans le tableau
//de sorte à ce que la pile ne contienne pas les mêmes trésors.
      si(t[i] =/= t[j])
         j <- j+1
      finSi
    si(i=j)
      i <- i+1
    finSi
  fin tant que

  //initialiser la pile des deux joueurs à leur position de départ
  empiler(P1, 25)
  empiler(P2, 26)

  //remplir la pile des deux joueurs avec le tableau[24] de valeur aléatoire
comprises entre 1 et 24.
  Pour i allant de 1 à 12 faire
    empiler(P1, tab[i])
  finPour
  Pour i allant de 13 à 24 faire
    empiler(P2, tab[i])
  finPour
FIN
```

```
-----------------
Joueur quiCommence (Joueur P1, Joueur P2)
DEBUT
  Entier x <- random entre 1 et 2
  si x=1
    retourner P1
  sinon
    retourner P2
  finSi
FIN
-----------------
initPlateau(Tuile Plateau[7][7], Tuile tuile_en_main)
DEBUT
  Pour i allant de 0 à 6 faire
    Pour(j allant de 0 à 6 faire)
      Plateau[i][j].posee <- 0
    finPour
  finPour

  poserTuilesFixes(Plateau)
  poserTuilesCouloir(Plateau, tuile_en_main)

FIN
-----------------
poserTuilesFixes(Tuile Plateau[7][7])
DEBUT
  Plateau[0][0].fixe <- 1
  Plateau[0][0].tresor <- 25
  Plateau[0][0].g <- 0
  Plateau[0][0].d <- 1
  Plateau[0][0].h <- 0
  Plateau[0][0].b <- 1
  Plateau[0][0].posee <- 1
  .
  . // 16 fixes au total
  .
  Plateau[6][6].fixe <- 1
  Plateau[6][6].tresor <- 26
  Plateau[6][6].g <- 1
  Plateau[6][6].d <- 0
  Plateau[6][6].h <- 1
  Plateau[6][6].b <- 0
  Plateau[6][6].posee <- 1
FIN
```

```
-----------------------------
poserTuilesCouloir(Tuile Plateau[7][7],tuile tuile_en_main)
DEBUT
  Tuile tuilesCouloir[34]
  ...
  // A remplir à la main
  coordonnées pos_aleatoire
  Entier i <- 0
  Tant que (!plateauRempl ET i<33)
  pos_aleatoire.x <- valeur aléatoire entre 0 et 6
  pos_aleatoire.y <- valeur aléatoire entre 0 et 6

  si (Plateau[pos_aleatoire.x][pos_aleatoire.y].posee=0)
    Plateau[pos_aleatoire.x][pos_aleatoire.y] <- tuilesCouloir[i]
    i <- i+1
  fin Si
  tuile_en_main <- tuilesCouloir[34]
  Fin Tant que
FIN
-----------------------------
alterner(Joueur p1, Joueur p2,Joueur joueurActuel)
DEBUT
  si joueurActuel = p1
    joueurActuel <- p2
  fin si
  sinon
    joueurActuel <- p1
FIN
-----------------------------
insererTuile (Tuile plateau[7][7],Tuile tuile_en_main,coordonnées
choix_precedent)
DEBUT
  coordonnées entré <- choixCouloir()
  tant que(!validationCouloir(entré,choixprécédent))
    entré <- choixCouloir()
  Fin tant que
  choixOrientationTuile(Tuile tuile_en_main)
  (choix_precedent) <- entré
  decalerCouloir(plateau, entré, tuile_en_main)
FIN
```

```
-----------------------------

booléen opposé(coordonnées a, coordonnées choix_precedent)
// on veut savoir si a est choix sont opposés
DEBUT fonction
  coordonnees tmp // pour trouver l'opposé de a
  si(a.x=0 et a.y=1)
    tmp.x <- 6
    tmp.y <- 1
  fin si

 sinon:
  si(a.x=0 et a.y=3)
      tmp.x <- 6;
      tmp.y <- 3;
  fin si
 .
 .
 .
  sinon si (a.x=6 et a.y= 1)
    tmp.x <- 0
    tmp.y <- 1

  sinon
    return 0;
  si // l'opposé de a (tmp) est le choix_actuel alors OUI
    (tmp.x=choix_precedent.x et tmp.y = choix_precedent.y)
    retourner 1;
  fin si

  sinon
    retourner 0;
FIN
-----------------------------
coordonnées choixCouloir()
DEBUT
  afficher(donnez coordonnées du couloir ou inserer)
  coordonnées a
  lire(a.x)
  lire(a.y)
  retourne a
FIN
-----------------------------
booléen validationCouloir(coordonnées choix_actuel,coordonnées choix_precedent)
DEBUT
  si (opposé(choix_actuel,(choix_precedent))
    (choix_precedent) <- choix_actuel
     retourne 1
  fin Si
  sinon
    retourne 0
FIN
```

```
-----------------------------
choixOrientationTuile(Tuile tuile_en_main)
DEBUT
  Entier choix
  afficher(1.inserer la tuile telle qu'elle est)
  afficher(2.tourner la tuile)
  lire(choix)
  si (choix=2)
    afficher('g' pour gauche 'd' pour droite)
    char direction
    lire(direction)
    Entier nbTours
    afficher(combien de tours)
    lire(nbTours)
    tourner(tuile_en_main,nbTours,direction)
  fin Si
FIN

-----------------------------
touner(Tuile tuile_en_main, Entier nombre_de_tours, char direction)
  // tuile_en_main: adresse de la tuile en main
  // nombre_de_tours: Entier entre 0 et 3
  // direction: g pour gauche, d pour droite
DEBUT
  Entier nbTours <- nombre_de_tours mod 4
  si (direction = 'g')
    Pour (Entier i allant de 0 à nbTours)
      tuile_en_main.g <- tuile_en_main.h
      tuile_en_main.h <- tuile_en_main.d
      tuile_en_main.d <- tuile_en_main.b
      tuile_en_main.b <- tuile_en_main.g
    fin Pour

  sinon si (direction = 'd')
        Pour (Entier i allant de 0 à nbTours)
          tuile_en_main.g <- tuile_en_main.b
          tuile_en_main.h <- tuile_en_main.g
          tuile_en_main.d <- tuile_en_main.h
          tuile_en_main.b <- tuile_en_main.d
        fin Pour
  fin Si
FIN
```

```
-----------------------------
decalerCouloir(Tuile Plateau, coordonnees choixCouloir, Tuile tuile_en_main)
  // Principe de la fonction: 4 orientations de décalage d'un couloir, par
exemple si on insert coté 'Ouest' il y aura un décalage du couloir de
  //...gauche vers la droite ainsi on modifie la ligne du plateau correspondant.
  // Autre exemple: Si on insert coté Nord il y aura un décalage du haut vers le
bas donc la fonction va modifier la colonne en partant du haut ...
DEBUT
  Tuile tmp
  Tuile dec
  Entier i <- 0
  si choixCouloir.x = 1 OU choixCouloir.x = 7:
    si choixCouloir.x = 1
      tmp <- plateau[choixCouloir.x][choixCouloir.y]
      plateau[choixCouloir.x][choixCouloir.y] <- tuile_en_main
      Pour i allant de 1 à 5:
        dec <- plateau[i+1][choixCouloir.y]
        plateau[i+1][choixCouloir.y] <- tmp
        tmp <- dec
      fin pour
    fin si

    si choixCouloir.x = 7
      tmp <- plateau[choixCouloir.x][choixCouloir.y]
      plateau[choixCouloir.y][choixCouloir.y] <- tuile_en_main
      Pour i allant de 5 à 1:
        dec <- plateau[i-1][choixCouloir.y]
       plateau[i-1][choixCouloir.y] <- tmp
        tmp <- dec
      fin pour
    fin si

  fin si

    sinon si choixCouloir.y = 1 OU choixCouloir.y = 7:
      si choixCouloir.y = 1
        tmp <- plateau[choixCouloir.x][choixCouloir.y]
        plateau[choixCouloir.x][choixCouloir.y] <- tuile_en_main
      Pour i allant de 1 à 5:
        dec <- plateau[choixCouloir.x][i+1]
        plateau[choixCouloir.x][i+1] <- tmp
        tmp <- dec
      fin pour

      si choixCouloir.y = 7
        tmp <- plateau[choixCouloir.x][choixCouloir.y]
        plateau[choixCouloir.x][choixCouloir.y] <- tuile_en_main
        Pour i allant de 5 à 1:
          dec <- plateau[choixCouloir.x][i-1]
          plateau[choixCouloir.x][i-1] <- tmp
          tmp <- dec
        fin pour
       fin si
    fin sinon
```

```
  tuile_en_main <- tmp
FIN
-----------------------------
booléen validationCoup(Tuile case,coordonnees a,coordonnees choix,Entier compt)
  // Tuile case : adresse de la case précedente appelé dans l'appel précedent
  // coordonnees a : position de actuel de l'appel de la fonction
  // coordonnees choix : La case à atteindre
  // Entier compt : nombre actuel des appels récursif de la fonction
DEBUT
  si(a.x = choix.x ET a.y = choix.y)
    retourne 1
  Fin si

  si(compt = 300) // Le plus long chemin possible pour être certain que la
fonction parcours tous les chemins possible à partir de la coordonnees et pour
que ça s'arrête.
    retourne 0
  fin si

  si ((tab[a.x][a.y].d = 1 ET tab[a.x][a.y = a.y+1].g = 1) ET tab[a.x][a.y = a.y
+ 1] =/= case)
 // Si c'est "ouvert" à droite et si à droite c'est "ouvert" à
//..gauche et que ce n'est pas la case précédente alors déplacer a (a.y
//incrémenté -> vers la droite)
    a.y = a.y+1                    // On déplace la coordonnees vers la
"droite" ...
    res =validationCoup(tab[a.x][a.y = a.y-1],a,choix,compt = compt +1) // Appel
récursif
  fin si

  si ((tab[a.x][a.y].h = 1 ET tab[a.x = a.x-1][a.y].b = 1) ET tab[a.x = a.x-
1][a.y] =/= case) //...
    a.x = a.x-1                                  //...
    res= validationCoup(tab[a.x = a.x+1][a.y],a,choix,comp = compt+1)
//...
  fin si

  si ((tab[a.x][a.y].g = 1 ET tab[a.x][a.y = a.y-1].d = 1) ET tab[a.x][a.y =
a.y-1] =/= case)
    a.y = a.y-1
    res= validationCoup(tab[a.x][a.y = a.y+1],a,choix,compt = compt+1)
  fin si

  si ((tab[a.x][a.y].b = 1 ET tab[a.x = a.x+1][a.y].h = 1) ET tab[a.x =
a.x+1][a.y] =/= case)
    a.x = a.x-1
    res= validationCoup(tab[a.x = a.x-1][a.y],a,choix,compt = compt+1)
  fin si
  retourne res // affectation = au lieu de <- pour la lisibilité
FIN
```

```
------------------------------
choixDeplacement(joueur P1, Tuile plateau[7][7])
DEBUT
  booleen deplacement
  coordonnees choix

  Afficher (Entrer 1 pour déplacer, 0 pour rester à la case actuelle)
  Lire (deplacement)
  si deplacement = 0
    retourne 1
  fin si
  sinon
    Lire (choix.x)
    Lire (choix.y)
  Tant
que(!validationCoup(plateau[P1.positionActuelle.x][P1.positionActuelle.y],P1.pos
itionActuelle,choix,0))
    choixDeplacement(p1, plateau)
  fin Tant que

  retourne 1
FIN
------------------------------
tresorAtteint(Joueur P1, Tuile Plateau[7][7])
DEBUT
  coordonnees a = P1.positionActuelle
  si(plateau[a.x][a.y].tresor = P1.pileTresor)
    retourne 1
  fin si

  sinon
    retourne 0
FIN
```

B. Programme principal :

```
Programme principal :
DEBUT
  modeDeJeu()
  Joueur p1,p2
  P1.pseudo = lire_pseudo()
  P2.pseudo = lire_pseudo()
  initPosition(p1,p2)
  distribuerCartes(p1,p2)
  Tuile plateau [7][7] // tableau 2D de tuiles
  Tuile tuile_en_main
  initPlateau(plateau, tuile_en_main)
  Joueur joueurActuel = quiCommence(p1,p2)
  coordonnees choix_precedent=(-1,-1)
  Tantque (!pileVide(p1.pile_tresor) ET !pileVide(p2.pile_tresor))
faire
  Debut
    affichageJeu()
    insererTuile(plateau,tuile_en_main,choix_precedent);
      //choix_precedent pour valider le choix actuel
    affichageJeu()
    choixDeplacement(joueurActuel)
    affichageJeu()
    si tresorAtteint(joueurActuel)
      depilerTresor(joueurActuel)
    fin si
    alterner(p1,p2,joueurActuel)
  finTantque
  affichageGagnant(p1,p2)
FIN
```

V. Conclusion :

À présent, on a assez d'éléments pour commencer l'implémentation du code en C, la plupart des éventuelles erreurs à résoudre seront plus reliés au langage plutôt qu'à l'algorithme.

La rédaction de ce rapport nous a permis de pousser notre réflexion le plus loin possible dans l'analyse des problématiques et la formalisation des solutions.
Avec les contributions de chacun des membres, on a appris l'organisation du travail en groupe (divisions des taches, utilisation des outils collaboratifs…) pour assurer le bon déroulement du projet.