Rapport d’analyse

-HERPOUX Mattéo

-PLEUTIN Benjamin

Sommaire

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. Ce rapport comprend une analyse exhaustive des besoins représentée pas du texte et un graphique, et les solutions apportées sous forme de fonctions écrites en pseudo-code.

II. Spécifications

1. Analyse des besoins

Identifier les joueurs

- Qui sont-ils?
    - Donner la possibilité aux joueurs de choisir leur pseudo
    - Donner la possibilité aux joueurs de choisir leur couleur
        - Changer la couleur d'une écriture.
- Combien d'humains / Combien d'IA ?
    - Programmer une IA
        - Faire en sorte qu'elle puisse jouer comme un joueur humain, qu'elle ne soit ni stupide ni infaillible
            - Regarder si de là où elle est, l'IA peut atteindre son trésor

- Elle ne bougera une case que si ça lui ouvre un chemin pour déplacer son pion d'au moins une case, et si ce n'est pas possible, chercher à embêter les autres joueurs
    - Regarder si, quand on bouge une ligne à proximité de l'IA, ça lui libère un chemin
    - Boucher le chemin, si c'est possible d'un autre joueur
        - Regarder, si en bougeant une ligne, le chemin du joueur sera bouché
    - Un autre joueur a-t-il un chemin ouvert pour son trésor?(qui est connu on le rappelle)

- Assigner un pion à chaque joueur(pseudo, couleur...)
- Répartir équitablement et aléatoirement la liste des trésors entre chaque joueurs
    - Trier un tableau de façon aléatoire

Créer le plateau

- Placer aléatoirement les cases (chemin, trésor)
    - Définir tous les différents types de cases
        - Définir une case
        - Créer les trésors
- Permettre à 12 lignes uniquement de se déplacer

Jouer un tour

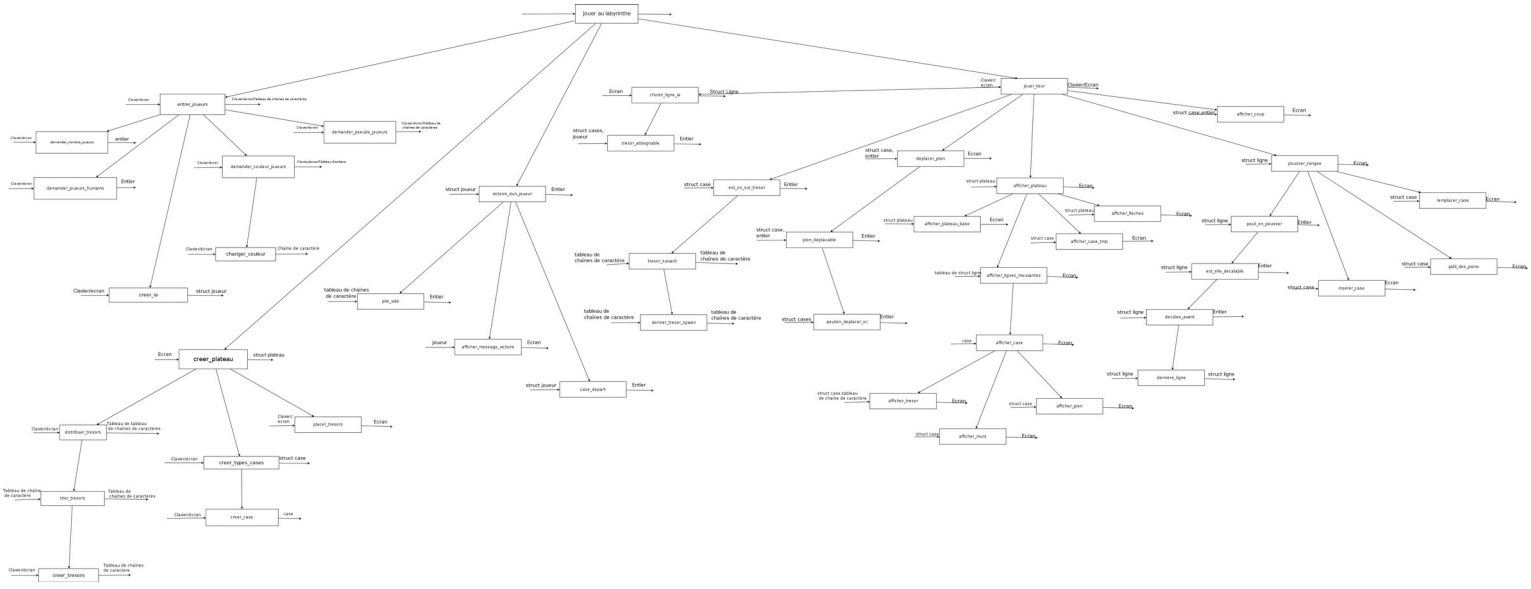
- Afficher le plateau
    - Afficher le plateau avec ses cases fixes
    - Afficher les flèches autour du plateau avec leur numéro
    - Afficher une ligne de cases mouvantes
        - Afficher une case
            - Afficher un trésor sur une case
            - Afficher une case de sorte à ce qu'on reconnaisse quel type de case c'est, où sont les murs
            - Afficher un pion sur une case
    - Afficher la case temporaire à coté du plateau
    - Si on peut, décaler la ligne d'un cran
        - Regarder si la ligne qu'on veut décaler est décalable
        - Regarder si la ligne qu'on veut décaler à été décalée juste avant dans le sens inverse
            - Garder en mémoire la dernière ligne décalée et dans quel sens elle l'a été
        - Insérer la case temporaire dans le plateau pour remplir la ligne
        - La case à la fin de la ligne devient la nouvelle case temporaire
        - Si un ou plusieurs pions sont sur la case qui sort, alors ils sont placés sur la case qui vient de rentrer

- Permettre ou non au pion de se déplacer
    - Le pion peut-il se déplacer à l'endroit voulu par le joueur?
        - Existe-il un chemin?
            - Lier toutes les cases compatibles pour créer un chemin
- Afficher la manipulation effectuée par le joueur({nom du joueur} a déplacé son pion en...)
- Regarder si on est sur une case trésor
- Passer au trésor suivant quand on atteint
    - Arrivé au dernier trésor, le nouvel objectif est le point de départ(un des 4 coins de l'écran)

Stopper la partie lorsqu'un joueur a gagné

- Regarder si un joueur a gagné à la fin du tour
    - Regarder si la pile de trésors d'un joueur est vide et si il est placé sur sa case de départ
- Afficher un message en conséquence
    - Regarder quel joueur a gagné

2. Analyse descendante



A télécharger en .png ici

III. Conception préliminaire

1. Fonctions

- joueur* entrer_joueurs()
- void demander_joueurs_humains(joueur *j)
- joueur* demander_nombre_joueurs()
- void demander_couleur_joueurs(joueurs *j)
- void demander_pseudo_joueur(joueur *j)
- void changer_couleur(joueur j)
- joueur creer_ia(joueur j)
- void victoire_dun_joueur(joueur j)
- void afficher_message_victoire(joueur j)
- int case_depart(joueur j)
- void jouer_tour_ia(joueur j, plateau p)
- ligne analyse_coups_possibles_IA() (En cours de conception)
- ligne choisir_ligne_ia(plateau p, joueur j)
- void deplacer_pion_ia()
- plateau creer_plateau()
- tresor* creer_tresors()
- void placer_tresors(plateau p, tresor* liste_tresors)
- void tri_aleatoire_cases_mouvantes(plateau p)
- void assignation_indices_cases(ligne*)
- ligne* creer_lignes_mouvantes()
- plateau creer_casesfixes()

- void assigner_valeurs_orientations(p)
- int tresor_atteignable(plateau p, joueur j)
- int est_on_sur_tresor(joueur j)
- void tresor_suivant(joueur j)
- void deplacer_pion(joueur j, plateau p, case c2)
- int pion_deplacable(plateau p, case c1, case c2)
- int peut_on_deplacer_ici(plateau p, case c1, case c2, case* tabcase)
- int case_compatible(case c1, case c2, char orientation(où est la case 1 par rapport a la 2))
- void afficher_plateau(plateau p)
- void afficher_fleche(char c)
- void afficher_case_tmp(case c)
- void afficher_lignes(plateau p)
- void afficher_ligne(ligne l)
- void afficher_case(case c)
- void afficher_liste_tresors(joueur j)
- void afficher_tresor_sur_case(case c)
- void afficher_murs(case c)
- void afficher_pion(case c)
- void afficher_coup(case c1, case c2, ligne l, joueur j)
- int est_elle_decalable(ligne l)
- void pousser_rangee(ligne l)
- ligne derniere_ligne(ligne l)
- void inserer_case(case c)
- void remplacer_case(case c)
- int yatil_des_pions(case c)

2. Structures :

- plateau
- tresor
- joueur
- ligne
- case

IV. Conception détaillée

1. Fonctions:

- joueur* entrer_joueurs()
    - joueur *j= demander_nombre_joueurs()
    - demander_joueurs_humains(j)
    - demander_couleur_joueurs(j)
    - demander_pseudo_joueur(j)

- void demander_joueurs_humains(joueur *j)
    - afficher "Combien d'IA?"
    - si nombre d'IA supérieur a nbrdejoueur-1
        - afficher "Nombre d'IA pas cohérent avec le nombre de joueurs"
    - sinon
        - pour i de 1 a nombre d'IA
            - creer_ia(j[i])

- joueur* demander_nombre_joueurs()
    - afficher "Combien de joueurs sur cette partie?"(nbr)
    - si nbr égal à 2, 3 ou 4
        - créer des struct joueurs en conséquence
    - sinon
        - afficher "Il n'y a pas un nombre adéquat de joueur"

- void demander_couleur_joueurs(joueurs *j)
    - pour chaque joueur
        - afficher "Quelle est la couleur du joueur j.i?"(afficher les couleurs 1:Rouge, 2:Jaune...)
        - j.couleur= le scanf (un int)

- void demander_pseudo_joueur(joueur *j)
    - pour chaque joueur
        - afficher "Quel est le pseudo du joueur j.i?"
        - j.nom= le scanf(%s)

- void changer_couleur(joueur j)
    - voir https://stackoverflow.com/questions/3219393/stdlib-and-colored-output-in-c

- joueur creer_ia(joueur j)
    - j.ia=1(fonction courte mais plus parlante ds le code que juste "j.ia=1")

- void victoire_dun_joueur(joueur j)
    - si case_depart(j)==1
        - afficher_message_victoire(j)
        - on sort de la boucle du main avec les jouer_tour

- void afficher_message_victoire(joueur j)
    - afficher "Le joueur j.nom a remporté la partie!"

- int case_depart(joueur j)
    - si j.tresors[0]='\0'
        - retourne 1
    - sinon
        - retourne 0

- void jouer_tour_ia(joueur j, plateau p)
    - analyse_coups_possibles_IA()
    - pousser_rangee(choisir_ligne_ia(p, j))
    - deplacer_pion_ia()
    - afficher_coup()

- void analyse_coups_possibles_IA() (En cours de conception)
-

- ligne choisir_ligne_ia(plateau p, joueur j)
    - pour les 12 lignes:
        - si, quand on bouge la ligne, le prochain tresor est accessible
            - on retourne cette ligne

- void deplacer_pion_ia()
    - si le prochain tresor est atteignable
        - deplacer_pion() sur le tresor
        - tresor_suivant()
    - sinon on reste ou on est

- plateau creer_plateau()
    - plateau p=creer_cases_fixes()
    - on cree un tableau de 12 lignes=creer_lignes_mouvantes()
    - assignation_indices_cases(le tableau)
    - tri_aleatoire_cases_mouvantes(p)
    - tresor* liste_tresors=creer_tresors()
    - placer_tresors(p, liste_tresors)

- tresor* creer_tresors()
    - on crée un tableau de 24 trésors
    - pour i de 0 a 23:
        - tresor[i].numero=i+1
        - tresor[i].nom="le nom"(peut etre dans un fichier)
    - retourne le tableau

- void placer_tresors(plateau p, tresor* liste_tresors)
    - creer un tableau de int de taille 34, le remplir avec 0,1,2,3,4...33
    - le trier aléatoirement
    - int j=0
    - pour i de 0 a 24
        - si la case n'est pas un coin
            - on associe a la case qui a le numéro tab[i] le tresor numero i de la liste
        - sinon
            - j++ et on associe le tresor numero i a la case[24+j]

- ligne* creer_lignes_mouvantes()
    - on crée un tableau de 12 lignes
    - pour i de 1 a 12
        - tab[i].numéro=i
        - on créé un tableau de 8 cases et on l'assigne a la ligne
        - pour chaque case de la ligne
            - case.numligne=i
            - on renvoie le tableau

- void tri_aleatoire_cases_mouvantes(plateau p)
    - creer un tableau de int de taille 34, le remplir avec 0,1,2,3,4...33
    - trier aléatoirement ce tableau
    - pour i de 0 a 11
        - si le num de la case est 0(la case tmp), case_tmp.type='a', sinon:
            - pour la case dont le case.numero est = a tab[i]: case.type[0]='a'
            - pour la case dont le case.numero est = a tab[i]: case.type[1]= une orientation aléatoire entre les 4(h,b,d ou g)
    - pour i de 12 a 17
        - si le num de la case est 0(la case tmp), case_tmp.type='c', sinon:
            - pour la case dont le case.numero est = a tab[i]: case.type[0]='c'
            - pour la case dont le case.numero est = a tab[i]: case.type[1]= une orientation aléatoire entre les 4(h,b,d ou g)
    - pour i de 18 a 33
        - si le num de la case est 0(la case tmp), case_tmp.type='b', sinon:
            - pour la case dont le case.numero est = a tab[i]: case.type[0]='b'
            - pour la case dont le case.numero est = a tab[i]: case.type[1]= une orientation aléatoire entre les 4(h,b,d ou g)

- void assignation_indices_cases(ligne*)
    - pour i de 2 a 6 (de 2 en 2)
        - pour toutes les 8 cases de la ligne[i/2] (j de 1 a 8):
            - case.numero_ligne[0]=i/2 et case.numero_ligne[1]=(i/2)+6 et quand j=i: case.numero_ligne[2]=i
            - indice i de la case =i
            - indice j de la case=j
            - case.numero=(i/2)*3+((i/2)-1)*7+j
    - pour i de 8 a 12 (de 2 en 2)
        - pour toutes les 8 cases de la ligne[i/2] (j de 1 a 8):
            - case.numero_ligne[0]=i/2 et case.numero_ligne[1]=(i/2)+6 et quand j=i: case.numero_ligne[2]=i
            - indice j de la case=i
            - indice i de la case=j
    - pour i de 1 a 7 (par 2)
        - pour j de 2 a 6 (par 2)
            - case[i][j].numero=((i-i%2)*5+j

- plateau creer_casesfixes()
    - on crée un plateau
    - case1,1.type=be
    - case1,3.type=ce
    - case1,5.type=ce
    - case1,7.type=bs
    - case3,1.type=cn
    - case3,3.type=cn
    - case3,5.type=ce
    - case3,7.type=cs
    - case5,1.type=cn
    - case5,3.type=co
    - case5,5.type=cs
    - case5,7.type=cs
    - case7,1.type=bn
    - case7,3.type=co
    - case7,5.type=co
    - case7,7.type=bo
    - retourne le plateau

- void assigner_valeurs_orientations(p)
    - pour chaque case du plateau:
        - si case an alors h=b=1 et g=d=0
        - si case ae alors h=b=0 et d=g=1
        - si case as alors h=b=1 et d=g=0
        - si case ao alors h=b=0 et d=g=1
        - si case bn alors h=d=1 et b=g=0
        - si case be alors h=g=0 et b=d=1
        - si case bs alors g=b=1 et h=d=0
        - si case bo alors h=g=1 et d=b=0
        - si case cn alors g=0 et h=d=b=1
        - si case ce alors h=0 et d=g=b=1
        - si case cs alors d=0 et h=g=b=1
        - si case co alors b=0 et h=g=d=1

- int tresor_atteignable(plateau p, joueur j)
    - si peut_on_deplacer_ici(p, j.case, j.tresor[0].case)==1
        - retourne 1
    - sinon
        - retourne 0

- int est_on_sur_tresor(joueur j)
    - si j.case.tresor==j.tresorsrestants[0]
        - retourne 1
    - sinon
        - retourne 0

- void tresor_suivant(joueur j)
    - pour tous les tresors restants -1(j.tresorsrestants)
        - tresors[i]=tresors[i+1]
    - dernier tresor(qui est ducoup en double)='\0'

- void deplacer_pion(joueur j, plateau p, case c2)
    - si pion_deplacable==1
        - j.case=c2
    - sinon
        - afficher "Le pion ne peux pas être déplacé ici"

- int pion_deplacable(plateau p, case c1, case c2)
    - si le pion n'est pas entre 4 murs(utiliser case_compatible) ET si peut_on_deplacer_ici(p, c1, c2)==1
        - retourne 1
    - sinon
        - retourne 0

static int tabcaseint=0

- int peut_on_deplacer_ici(plateau p, case c1, case c2, case* tabcase)
    - tabcase[tabcaseint]=c1
    - tabcaseint++
    - (cette ligne est un for)si tabcaseint=0, pour les 4 orientation, sinon, pour seulement 3(pas celle d'ou l'on vient):
        - si case_compatible(p, c1, (la case a coté de c1(suivant l'orientation)), tabcase)==1
            - peut_on_deplacer_ici(p, la case a coté, c2, tabcase)
    - si c2 présente dans tabcase
        - retourne 1
    - sinon
        - retourne 0

- int case_compatible(case c1, case c2, char orientation(où est la case 1 par rapport a la 2))
    - si c1.orientation=1 et c2.orientation opposée=1
        - retourne 1
    - sinon
        - retourne 0

- void afficher_plateau(plateau p)
    - pour i de 0 a 7
        - afficher_fleche(s1)
    - pour i de 0 a 7
        - afficher_fleche(s2)
    - afficher_lignes(p)
    - pour i de 0 a 7
        - afficher_fleche(n1)
    - pour i de 0 a 7
        - afficher_fleche(n2)

- void afficher_fleche(char c)
    - si c='s1', afficher "|"
    - si c='s2', afficher "v"
    - si c='e', afficher "->"
    - si c='o', afficher "<-"
    - si c='n1', afficher "^"
    - si c='n2', afficher "|"

- void afficher_case_tmp(case c)
    - afficher "Case pour pousser:"
    - afficher_case(c)

- void afficher_lignes(plateau p)
    - pour i de 1 a 7 * afficher_ligne(p.l[i])

- void afficher_ligne(ligne l)
    - afficher_fleche(e)
    - pour i de 1 a 7
        - afficher_case(l.c[i])
    - afficher_fleche(o\n)

- void afficher_case(case c)
    - afficher_murs(c)
    - afficher_pion(c)
    - afficher_tresor_sur_case(c)

- void afficher_liste_tresors(joueur j)
    - pour tous les trésors restants
        - afficher ""numéro du trésor". "nom du trésor"\n"

- void afficher_tresor_sur_case(case c)
    - afficher "c.tresor"

- void afficher_murs(case c)
    - afficher les murs en fonction du type de c avec des "|" et "-"
- void afficher_pion(case c)
    - changer la couleur du pion en fonction de c.joueur
    - afficher "I" de la bonne couleur

- void afficher_coup(case c1, case c2, ligne l, joueur j)
    - afficher "Le joueur j.nom a poussé la ligne l et a déplacé son pion de la case c1 a la case c2"

- int est_elle_decalable(ligne l)
    - Si l=ligne_tmp
        - on retourne 1
    - Sinon on retourne 0

- void pousser_rangee(ligne l)
    - Si est_elle_decalable(l)==1
        - Pour i de 0 a 7
            - l.case[i+1]=l.case[i]
        - l.case[0]=case_tmp
        - case_tmp=case[8]

- ligne derniere_ligne(ligne l)
    - ligne_tmp=l

- void inserer_case(case c)
    - c prend la valeur de case_tmp

- void remplacer_case(case c)
    - case_tmp prend la valeur de c

- int yatil_des_pions(case c)
    - Si il y a 0 joueurs sur la case:
        - on renvoit 0
    - Sinon:
        - on renvoit 1

2. Structures

- plateau
    - les 12 lignes
    - la case temporaire
- joueur
    - le nom
    - la couleur
    - si c'est une ia(un int)
    - la liste des tresors qu'il lui reste a atteindre
- tresor
    - le nom
    - son numero
    - la case où il est
- ligne
    - le numéro de la ligne
    - le tableau de cases de la ligne
- case
    - le trésor sur la case(rien si il n'y en a pas)
    - le types de case et son orientation(chaine(an, be...))
    - si il y a des murs au 4 orientations(4 entiers(booléens) h, b, d, g)
    - le pion sur la case (PERSONNE si il n'y en a pas)
    - numero_ligne le numéro de la ligne auquelle la case appartient (tableau de 3 int, -1, -1, -1 si case non mouvante)
    - numero le numéro de case/34
    - les indices de la case dans le plateau