Allainé Hugo
Angonnet Léo

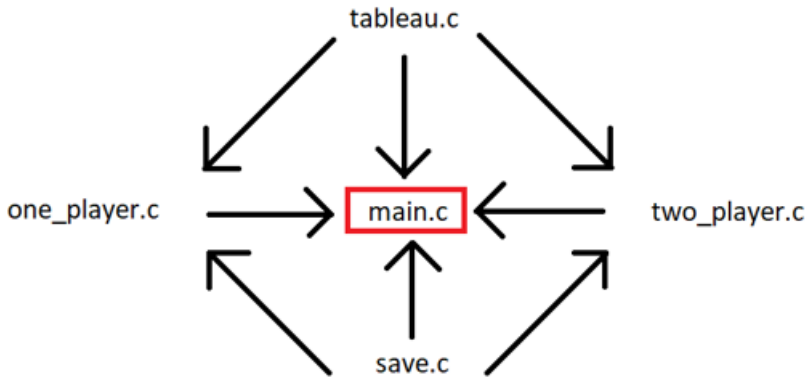
Groupe Z

IF2B – Livrable Projet Mefuhva – Automne 2021

Sommaire

I. Structure générale du code (Algorithme)

1. Schéma explicatif des liens entre les différents programmes



2. Programme principal du jeu Mefuhva (main.c)

Commandes préprocesseurs :

Modules importés : <stdio.h>, <stdlib.h>, <ctype.h>, "save.h", "tableau.h", "one_player.h", "two_player.h"

Si la compilation est effectuée depuis un OS Windows

Importation du module <windows.h>

Définition de la macro "START" (cette macro prend ici des commandes du module <windows.h> pour modifier la plage de caractère utilisé par l'invité de commande Windows en plage UTF-8 et ainsi résoudre tout problème d'affichage de caractères accentués.)

Définition de la macro "CLOSE" (cette macro prend ici une commande permettant le rétablissement de la table de caractère utilisé par l'invité de commande Windows avant la compilation du programme.)

Sinon

Définition de la macro "START" vide.

Définition de la macro "CLOSE" vide.

Fin Si

Variables utilisées :

Entiers : answer ← 0, type_of_game, nb_tour_2j ← 1

Caractère : choix

Tableau : plateau _de_jeu[4][6]

Début

Macro START. (Explication préprocesseur)

Ecrire ("Bienvenue dans le jeu du Mefuhva ! © Hugo Allainé - Léo Angonnet")

Tant que answer est égale à 0

Le programme demande à l'utilisateur le mode de jeu qu'il souhaite lancer, si l'utilisateur n'entre pas une lettre correspondante à un mode de jeu proposé, le programme demande à nouveau la même question à l'utilisateur.

Début Condition multiple (switch(choix))

Si la réponse de l'utilisateur est "a" (Mode 1 joueur) :

Ecrire ("Vous avez choisis le mode 1 Joueur.")

Ecrire les consignes et les règles du jeu à l'utilisateur.

Remplir le plateau de jeu avec une configuration de départ (remplir_tableau)

Afficher le plateau de jeu à l'utilisateur (afficher_tableau)

Lance la partie en mode 1 joueur (one_player)

Si la réponse de l'utilisateur est "b" (Mode 2 joueurs) :

Ecrire ("Vous avez choisis le mode 2 Joueurs.")

Ecrire les consignes et les règles du jeu à l'utilisateur.

Remplir le plateau de jeu avec une configuration de départ (remplir_tableau)

Afficher le plateau de jeu à l'utilisateur (afficher_tableau)

Lance la partie en mode 2 joueur (two_player)

Si la réponse de l'utilisateur est "c" (Reprendre la partie) :

Ecrire ("Vous avez choisis de reprendre une partie.")

Attribution des différents paramètres de jeu (type_of_game, nb_tour_2j, plateau_de_jeu) à l'aide du fichier de sauvegarde (load)

SI type_of_game est égale à 1

Afficher le plateau de jeu à l'utilisateur (afficher_tableau)

Lance la partie en mode 1 joueur (one_player)

Fin Si

Si type_of_game est égale à 2

Afficher le plateau de jeu à l'utilisateur (afficher_tableau)

Lance la partie en mode 2 joueur (two_player)

Fin Si

Si la réponse de l'utilisateur est "d" (Quitter le jeu) :

Ecrire ("A très bientôt !")

answer ← 1

Sinon

Ecrire ("Erreur, vous n'avez pas compris. Entrez la lettre correspondante à l'option souhaitée.")

Fin Condition multiple (switch)

Fin Tant que

Macro CLOSE. (Explication préprocesseur)

Fin

3. Fonctions de gestion du plateau de jeu (tableau.c)

Commandes préprocesseurs :

Modules importés : "tableau.h", <stdio.h>, <stdlib.h>

Début

Structure Coordonnees

Entier ligne, colonne

Fin Coordonnees

Typedef structure Coordonnees Coord

// Afin de résoudre des problèmes entre les fichiers, cette structure a été ensuite définie dans tableau.h directement

Fonction remplir_tableau(tableau[4][6])

Entiers : i ← 0, j ← 0

```
        // Fonction de remplissage du plateau de jeu (configuration de base d'une partie)
        Pour i<4
                Pour j<6
                        tableau[i][j] ← 2
                Fin Pour
        Fin Pour
        tableau[1][4] ← 1, tableau[1][5] ← 0
        tableau[2][0] ← 0, tableau[2][1] ← 1
Fin remplir_tableau

Fonction afficher_tableau(tableau[4][6])
        Entiers : i ← 0, j ← 0
        // Fonction affichant le plateau de jeu
        Ecrire ("----- Joueur 1 -----")
        Pour i<4
                Si i est égale à 2
                        Ecrire ("--------------------")
                Fin Si
                Pour j<6
                        Ecrire (" [tableau[i][j]] ")
                Fin Pour
                Ecrire ("\n")
        Fin Pour
        Ecrire ("----- Joueur 2 -----")
Fin afficher_tableau

Fonction case_suivante(Coord c)
        // Fonction retournant les coordonnées de la case suivante pour les déplacements du jeu
        Si c.ligne est égale à 0
                Si c.colonne est égale à 0
                        c.ligne ← 1
```

c.colonne ← 0

Retourner c

Sinon

c.ligne ← 0

c.colonne ← c.colonne - 1

Retourner c

Fin Si

Fin Si

Si c.ligne est égale à 1

Si c.colonne est égale à 5

c.ligne ← 0

c.colonne ← 5

Retourner c

Sinon

c.ligne ← 1

c.colonne ← c.colonne + 1

Retourner c

Fin Si

Fin Si

Si c.ligne est égale à 2

Si c.colonne est égale à 0

c.ligne ← 3

c.colonne ← 0

Retourner c

Sinon

c.ligne ← 2

c.colonne ← c.colonne - 1

Retourner c

Fin Si

Fin Si

Si c.ligne est égale à 3

```
            Si c.colonne est égale à 5
                c.ligne ← 2
                c.colonne ← 5
                Retourner c
            Sinon
                c.ligne ← 3
                c.colonne ← c.colonne + 1
                Retourner c
            Fin Si
        Fin Si
Fin case_suivante

Fonction dans_tableau(Coord c, Entier joueur, tab[4][6])
        // Fonction indiquant si le joueur a rentré des coordonnées se trouvant dans sa zone de jeu
        Si c.colonne est comprise entre 0 et 5 inclus
            Si joueur est égale à 1 et c.ligne est comprise entre 0 et 1 inclus et tab[c.ligne]
            [c.colonne] n'est pas égale à 0
                Retourner 1
            Fin Si
            Si joueur est égale à 2 et c.ligne est comprise entre 2 et 3 inclus et tab[c.ligne]
            [c.colonne] n'est pas égale à 0
                Retourner 1
            Sinon
                Retourner 0
            Fin Si
        Sinon
            Retourner 0
        Fin Si
Fin dans_tableau

Fonction deplacement_graine(Coord c, tab[4][6])
        Entiers : nb_graine ← tab[c.ligne][c.colonne], j ← 0
```

```
        // Fonction permettant de déplacer ces graines une à une
        Tant que nb_graine n'est pas égale à 0
                tab[c.ligne][c.colonne] ← 0
                Pour j<nb_graine
                        c ← case_suivante(c)
                        tab[c.ligne][c.colonne] ← tab[c.ligne][c.colonne] + 1
                Fin Pour
                Si tab[c.ligne][c.colonne] > 1
                        nb_graine ← tab[c.ligne][c.colonne]
                Sinon
                        nb_graine ← 0
                Fin Si
        Fin Tant que
        Retourner c
Fin deplacement_graine


Fonction capture_possible(Coord c, Entier joueur, tab[4][6])
        // Fonction indiquant si la dernière graine se trouve en bordure extérieur ou intérieur (donc
                si la capture est possible ou non)
        Si c.colonne est comprise entre 0 et 5 inclus
                Si joueur est égale à 1 et c.ligne est égale à 1 et tab[c.ligne+1][c.colonne] n'est pas
                                égale à 0
                        Retourner 1
                Fin Si
                Si joueur est égale à 2 et c.ligne est égale à 2 et tab[c.ligne-1][c.colonne] n'est pas
                                égale à 0
                        Retourner 1
                Sinon
                        Retourner 0
                Fin Si
        Sinon
                Retourner 0
```

```
        Fin Si
Fin capture_possible

Fonction capture_graine(Coord c, Entier joueur, tab[4][6])
        Entiers : recolte ← 0
        // Fonction permettant de capturer les graines de l'adversaire
        Si joueur est égale à 1
                recolte ← tab[c.ligne+1][c.colonne] + tab[c.ligne+2][c.colonne]
                tab[c.ligne+1][c.colonne] ← 0
                tab[c.ligne+2][c.colonne] ← 0
        Fin Si
        Si joueur est égale à 2
                recolte ← tab[c.ligne-1][c.colonne] + tab[c.ligne-2][c.colonne]
                tab[c.ligne-1][c.colonne] ← 0
                tab[c.ligne-2][c.colonne] ← 0
        Fin Si
        tab[c.ligne][c.colonne] ← tab[c.ligne][c.colonne] + recolte
Fin capture_graine

Fonction is_finish(tab[4][6])
        Entiers : compteur_j1 ← 0, compteur_j2 ← 0
        // Fonction indiquant s'il reste encore des graines sur les camps des joueurs
        Pour i<4
                Pour j<6
                        Si i est égale à 0 ou 1
                                Si tab[i][j] n'est pas égale à 0
                                        i ← i+1
                                        compteur_j1 = 1
                                        break
                                Fin Si
                        Fin Si
```

```
                    Si i est égale à 2 ou 3
                        Si tab[i][j] n'est pas égale à 0
                            i ← i+1
                            compteur_j2 = 1
                            break
                        Fin Si
                    Fin Si
                Fin Pour
            Fin Pour
            Si compteur_j1 est égale à 1 et compteur_j2 est égale à 1
                Retourner 0
            Sinon
                Si compteur_j1 est égale à 0 et compteur_j2 est égale à 1
                    Retourner 1
                Fin Si
                Si compteur_j1 est égale à 1 et compteur_j2 est égale à 0
                    Retourner 2
                Sinon
                    Ecrire ("[DEBUG][is_finish] Bug")
                Fin SI
            Fin Si
Fin is_finish
```

Fin

4. Fonctions de gestion de la sauvegarde (save.c)

Commandes préprocesseurs :

Modules importés : "save.h", <stdio.h>, <stdlib.h>, <time.h>

Début

```
Fonction save(Entier type_of_game, Entier nb_tour, tab[4][6])
        Entiers : h, m, s, jour, mois, an, i ← 0, j ← 0
        Fichier : f ← "save.txt" ouvert en écriture
        Time_t : now
        Struct tm : *local
        // Fonction permettant de sauvegarder le plateau de jeu actuel dans un fichier txt
        Ecrire ("Sauvegarde de la partie en cours...")
        Si f n'est pas égale à NULL (c'est à dire existe)
                h ← local->tm_hour
                m ← local->tm_min
                s ← local->tm_sec
                jour ← local->tm_mday
                mois ← local->tm_mon
                an ← local->tm_year
                Ecrire dans le fichier f ("Sauvegarde du jour/mois/an à h:m:s\n\n")
                Ecrire dans le fichier f ("type_of_game\n")
                Ecrire dans le fichier ("nb_tour\n")
                Pour i<4
                        Pour j<4
                                Ecrire dans le fichier ("tab[i][j] ")
                        Fin Pour
                        Ecrire dans le fichier ("\n")
                Fin Pour
                Ecrire ("Sauvegarde effectuée avec succès !\n\n")
        Sinon
```

```
            Ecrire ("[DEBUG][save] Erreur, fichier non trouvé.\n")
        Fin Si
        Fermer le fichier f
Fin save


Fonction load(Entier Pointeur type_of_game, Entier Pointeur nb_tour, tab[4][6])
        Caractère buffer[256]
        Entier t, nb, i ← 0
        Fichier f ← "save.txt" ouvert en lecture
        // Fonction permettant de charger la sauvegarde du plateau de jeu depuis un fichier txt
        Ecrire ("Chargement de la partie en cours...\n")
        Si f n'est pas égale à NULL (c'est à dire existe)
            Pour k<2
                Lire une ligne du fichier et la stocker dans buffer
            Fin Pour
            Lire une ligne du fichier ("%d") et la stocker dans t
            *type_of_game ← t
            Lire une ligne du fichier ("%d") et la stocker dans nb
            *nb_tour ← nb
            Pour i<4
                Lire une ligne du fichier ("%d %d %d %d %d ") et la stocker dans tab[i][0],
                tab[i][1], tab[i][2], tab[i][3], tab[i][4], tab[i][5]
            Fin Pour
            Ecrire ("Chargement effectué avec succès !\n\n")
        Sinon
            Ecrire ("[DEBUG][load] Erreur, fichier non trouvé.\n")
        Fin Si
        Fermer le fichier f
Fin load
```

Fin

5. Fonction du mode 1 Joueur (one_player.c)

Commandes préprocesseurs :

Modules importés : "one_player.h", <stdio.h>, <stdlib.h>, <time.h>, <ctype.h>, <unistd.h>, "tableau.h", "save.h"

Début

Fonction one_player(plateau_de_jeu[4][6], Entier nb_tour_2j)

Entiers : partie_en_cours ← 1, code_fin ← 0, nb_aleatoire

Coord : coord_j1, coord_j2

Caractère : continuer

// Fonction de jeu contre l'ordinateur

Tant que partie_en_cours est égale à 1

Ecrire ("Tour n°nb_tour_2j\n")

Ecrire ("C'est au tour de Joueur 1 !\n")

Faire

Ecrire ("Joueur 1, sélectionne la ligne (0,1) : ")

Lire (coord_j1.ligne)

Vider le buffer d'entrée

Ecrire ("Joueur 1, sélectionne la colonne (0,5) : ")

Lire (coord_j1.colonne)

Vider le buffer d'entrée

Tant que dans_tableau(coord_j1,1,plateau_de_jeu) renvoie 0

Ecrire ("Déplacement de vos graines en cours…\n")

coord_j1 ← deplacement_graine(coord_j1,plateau_de_jeu)

Ecrire ("Jouer 1, vous vous arrêtez en coordonnée (coord_j1.ligne,coord_j1.colonne).\n")

Afficher le plateau de jeu

Attendre 3 secondes

Si capture_possible(coord_j1,1,plateau_de_jeu) renvoie 1

Ecrire ("Joueur 1, tu as capturé des graines de ton adversaire\n")

capture_graine(coord_j1,1,plateau_de_jeu)

Afficher le plateau_de_jeu

Sinon

Ecrire ("Joueur 1, tu n'as pas pu capturer des graines de ton adversaire\n")

Fin Si

code_fin ← is_finish(plateau_de_jeu)

Si code_fin est égale à 1

Ecrire ("Joueur 1 n'a plus de graine sur son plateau, l'ordinateur gagne la partie.\n")

partie_en_cours ← 0

break

Fin Si

Si code_fin est égale à 2

Ecrire ("L'ordinateur n'a plus de graine sur son plateau, Joueur 1 gagne la partie.\n")

partie_en_cours ← 0

break

Fin Si

Attendre 3 secondes

Ecrire ("C'est au tour de l'ordinateur.\n")

Définir la variable d'aléatoire

Faire

Coord_j2.ligne = 2 + rand() %2

coord_j2.ligne ← 2 + nombre aléatoire entre 0 et 1 inclus

coord_j2.colonne ← nombre aléatoire entre 0 et 5 inclus

Tant que dans_tableau(coord_j2,2,plateau_de_jeu) renvoie 0

Ecrire ("Déplacement de vos graines en cours...\n")

coord_j2 ← deplacement_graine(coord_j2,plateau_de_jeu)

Ecrire ("Ordinateur, vous vous arrêtez en coordonnée (coord_j2.ligne,coord_j2.colonne).\n")

Afficher le plateau de jeu

Attendre 3 secondes

Si capture_possible(coord_j2,2,plateau_de_jeu) renvoie 1

Ecrire ("Ordinateur, tu as capturé des graines de ton adversaire\n")

capture_graine(coord_j2,2,plateau_de_jeu)

Afficher le plateau_de_jeu

Sinon

Ecrire ("Ordinateur, tu n'as pas pu capturer des graines de ton adversaire\n")

Fin Si

code_fin ← is_finish(plateau_de_jeu)

Si code_fin est égale à 1

Ecrire ("Joueur 1 n'a plus de graine sur son plateau, l'ordinateur gagne la partie.\n")

partie_en_cours ← 0

break

Fin Si

Si code_fin est égale à 2

Ecrire ("L'ordinateur n'a plus de graine sur son plateau, Joueur 1 gagne la partie.\n")

partie_en_cours ← 0

break

Fin Si

Ecrire ("Fin du tour, si vous souhaitez arrêter et sauvegarder, entrez *, sinon appuyez sur Entrée : \n")

Lire (continuer)

Vider le buffer d'entrée

Si continuer correspond au caractère *

Sauvegarder la partie (save(1,nb_tour_2j,plateau_de_jeu))

partie_en_cours ← 0

break

Sinon

Ecrire ("Très bien, continuons.\n")

Fin Si

nb_tour_2j ← nb_tour_2j+1

Fin Tant que

Attendre 3 secondes

Fin one_player

Fin

6. Fonction du mode 2 Joueurs (two_player.c)

Commandes préprocesseurs :

Modules importés : "two_player.h", <stdio.h>, <stdlib.h>, <time.h>, <ctype.h>, <unistd.h>, "tableau.h", "save.h"

Début

Fonction two_player(plateau_de_jeu[4][6], Entier nb_tour_2j)

Entiers : partie_en_cours ← 1, code_fin ← 0

Coord : coord_j1, coord_j2

Caractère : continuer

// Fonction de jeu contre l'ordinateur

Tant que partie_en_cours est égale à 1

Ecrire ("Tour n°nb_tour_2j\n")

Ecrire ("C'est au tour de Joueur 1 !\n")

Faire

Ecrire ("Joueur 1, sélectionne la ligne (0,1) : ")

Lire (coord_j1.ligne)

Vider le buffer d'entrée

Ecrire ("Joueur 1, sélectionne la colonne (0,5) : ")

Lire (coord_j1.colonne)

Vider le buffer d'entrée

Tant que dans_tableau(coord_j1,1,plateau_de_jeu) renvoie 0

Ecrire ("Déplacement de vos graines en cours…\n")

coord_j1 ← deplacement_graine(coord_j1,plateau_de_jeu)

Ecrire ("Jouer 1, vous vous arrêtez en coordonnée (coord_j1.ligne,coord_j1.colonne).\n")

Afficher le plateau de jeu

Attendre 3 secondes

Si capture_possible(coord_j1,1,plateau_de_jeu) renvoie 1

    Ecrire ("Joueur 1, tu as capturé des graines de ton adversaire\n")

    capture_graine(coord_j1,1,plateau_de_jeu)

    Afficher le plateau_de_jeu

Sinon

    Ecrire ("Joueur 1, tu n'as pas pu capturer des graines de ton adversaire\n")

Fin Si

code_fin ← is_finish(plateau_de_jeu)

Si code_fin est égale à 1

    Ecrire ("Joueur 1 n'a plus de graine sur son plateau, Joueur 2 gagne la partie.\n")

    partie_en_cours ← 0

    break

Fin Si

Si code_fin est égale à 2

    Ecrire ("Joueur 2 n'a plus de graine sur son plateau, Joueur 1 gagne la partie.\n")

    partie_en_cours ← 0

    break

Fin Si

Attendre 3 secondes

Ecrire ("C'est au tour du Joueur 2.\n")

Faire

    Ecrire ("Joueur 2, sélectionne la ligne (2,3) : ")

    Lire (coord_j2.ligne)

    Vider le buffer d'entrée

    Ecrire ("Joueur 2, sélectionne la colonne (0,5) : ")

    Lire (coord_j2.colonne)

    Vider le buffer d'entrée

Tant que dans_tableau(coord_j2,2,plateau_de_jeu) renvoie 0

Ecrire ("Déplacement de vos graines en cours...\n")

coord_j2 ← deplacement_graine(coord_j2,plateau_de_jeu)

Ecrire ("Joueur 2, vous vous arrêtez en coordonnée (coord_j2.ligne,coord_j2.colonne).\n")

Afficher le plateau de jeu

Attendre 3 secondes

Si capture_possible(coord_j2,2,plateau_de_jeu) renvoie 1

    Ecrire ("Joueur 2, tu as capturé des graines de ton adversaire\n")

    capture_graine(coord_j2,2,plateau_de_jeu)

    Afficher le plateau_de_jeu

Sinon

    Ecrire ("Joueur 2, tu n'as pas pu capturer des graines de ton adversaire\n")

Fin Si

code_fin ← is_finish(plateau_de_jeu)

Si code_fin est égale à 1

    Ecrire ("Joueur 1 n'a plus de graine sur son plateau, Joueur 2 gagne la partie.\n")

    partie_en_cours ← 0

    break

Fin Si

Si code_fin est égale à 2

    Ecrire ("Joueur 2 n'a plus de graine sur son plateau, Joueur 1 gagne la partie.\n")

    partie_en_cours ← 0

    break

Fin Si

Ecrire ("Fin du tour, si vous souhaitez arrêter et sauvegarder, entrez *, sinon appuyez sur Entrée : \n")

Lire (continuer)

Vider le buffer d'entrée

Si continuer correspond au caractère *

    Sauvegarder la partie (save(2,nb_tour_2j,plateau_de_jeu))

    partie_en_cours ← 0

```
                    break
                Sinon
                    Ecrire ("Très bien, continuons.\n")
                Fin Si
                nb_tour_2j ← nb_tour_2j+1
            Fin Tant que
            Attendre 3 secondes
Fin two_player
```

Fin

II. Les choix réalisés pour le développement du jeu

1. Les problèmes rencontrés et les choix réalisés

Dans un premier temps, nous avons créé 5 fichiers sources. Mais vu que le programme n'était pas complet, nous n'arrivions pas à le compiler. Nous avons donc tout regroupé sur un seul fichier, pour faire différents essais. Une fois le programme fonctionnel, nous l'avons refragmenté dans les 5 fichiers prévus à la base.

Dans la fonction « case_suivante » dans le fichier « tableau.c ». Il y avait deux variables « ligne » et « colonne » et la fonction devait renvoyer ces deux variables. Le langage C ne le permettant pas nous avons mis en place une variable de type structure, ce qui nous a permis de retourner nos deux variables insérées dans la variable structure.

Nous avons eu plusieurs problèmes au début du développement à propos du compilateur. Nous voulions initialement compiler nos fichiers ensembles mais comme nos fichiers n'étaient pas regroupé dans un projet CodeBlocks, un seul fichier était compilé au lancement. Nous avons pallié cela en compilant à la main au début jusqu'à comprendre que CodeBlocks peut compiler l'ensemble des fichiers en mode projet.

Lors de la réalisation des différents programmes modulaires, nous avons choisis de faire un fichier header par fichier source afin de bien séparer les modules alors que dans le cadre de ce projet nous aurions pu regrouper tous les fichiers sources en un fichier header.

Pour optimiser la vérification de fin de partie, nous avons fait le choix de vérifier s'il reste au moins une graine dans chaque coté du plateau de jeu, c'est-à-dire que lorsque qu'un graine est détecté dans le terrain du joueur 1, le programme passe directement au joueur 2 sans finir de parcourir le reste du terrain du joueur 1.

Nous avons fait le choix d'ajouter un temps de pause de 3 secondes entre l'affichage d'événement majeur comme le plateau après un déplacement et après la capture afin d'ajouter de la lisibilité dans le jeu. De plus, pendant le développement de l'ordinateur, nous nous sommes rendu compte que l'aléatoire rendait l'expérience de jeu assez complexe car même si l'ordinateur se déplace aléatoirement, la situation peut s'inverser rapidement dans ce jeu et donc prolonger le temps de partie.

Afin de simplifier notre travail, nous avons créé deux programmes en langage shell, un pour Windows (.bat) et un pour Linux/macOS (.sh), qui compilent le programme rapidement en utilisant le compilateur GCC depuis la console du système d'exploitation. Pour maximiser la compatibilité de ces programmes, nous avons donné des instructions de compilation au début de notre programme principal. Ces instructions permettent de changer la table de caractère de l'invité de commande Windows, passant en UTF-8, pour avoir un affichage correct des caractères accentués et spéciaux, puis de rétablir la table de caractère présente avant l'exécution du programme lorsque le programme s'arrête. Lorsque la compilation est faite sur Linux ou macOS, ces instructions ne sont pas données car le terminal utilisé sur ces systèmes d'exploitations est déjà en UTF-8.

III. Comment compiler le programme ?

1. Sur Code::Blocks

Pour compiler le programme sur CodeBlocks, double-cliquez sur Projet.cbp ou ouvrez manuellement CodeBlocks puis dans Ouvrir, cliquez sur Projet.cbp. Vous pouvez ensuite cliquer sur Build and Run et le programme se compilera et se lancera.

2. Sur L'invité de commande Windows / Terminal Linux

Pour compiler le programme depuis l'invité de commande Windows, vous pouvez lancer le fichier compiler.bat inclus ou taper manuellement dans une nouvelle fenêtre pointant vers le dossier où se trouve les fichiers sources la commande suivante :

*gcc *c -o projet.exe*

Vous devez avoir le compilateur GCC inclus dans vos variables d'environnement pour compiler avec l'invité de commande.

Pour compiler le programme depuis le terminal sur une distribution Linux, lancer une fenêtre du terminal à l'emplacement où se trouve compiler.sh et lancer ce fichier avec la commande suivante :

./compiler.sh

Vous pouvez aussi compiler avec la méthode manuelle expliquée précédemment.

IV. Résultat final / Bilan

1. Ce qui a été réussis

Le jeu fonctionne avec les règles indiquées donc la cahier des charges a bien été respecté. La sauvegarde est aussi fonctionnelle, La sauvegarde est aussi fonctionnelle, le programme peut charger une ancienne partie depuis un fichier txt.

Les éléments bonus non indiqué dans le cahier des charges sont aussi fonctionnels.

2. Les points d'amélioration éventuels

L'intelligence du bot quand nous jouons seul pourrait être améliorée, car pour le moment l'ordinateur se contente de choisir aléatoirement une case.

Quelques vérifications d'entrée utilisateur pourraient être ajoutées notamment à la demande des coordonnées de la case à jouer.

3. Conclusion

Pour conclure sur ce projet, il nous a permis de découvrir de façon plus avancé la programmation en langage C ainsi que plus largement le fonctionnement d'un compilateur et des instructions données à celui-ci. Cet exercice nous a aussi appris la rédaction des différents algorithmes ainsi que la pratique de la programmation modulaire et l'application de convention de développement.