Project Laby

Groupe E-F

Sommaire

Cahier des charges :

Le but de ce projet est de réaliser une arène où un maximum de 30 IA pourront s'affronter. La partie devra durer au maximum 4 minutes et se déroulera sur un maximum de 120 tours. Pour l'aspect graphique, l'arène devra être visible quand elle sera projetée sur un écran d'amphithéâtre.

Le concept de l'arène :

Il s'agit d'un labyrinthe de taille variable selon le nombre de joueurs (20*20 pour 4 joueurs par défaut) dans lequel des murs seront générés aléatoirement. Le but du jeu sera d'éliminer les autres joueurs dans le labyrinthe. La partie se déroule en 120 tours et à chaque tour, chaque joueur aura le choix parmi deux actions. Soit il pourra avancer pour essayer de tuer un autre joueur, soit il pourra casser un mur du labyrinthe. Lorsqu'un joueur détruit un mur, il ne peut pas se déplacer ,de même que lorsqu'il se déplace, il ne peut pas casser de mur.

Stratégie :

Chaque joueur devra réfléchir à chaque tour à la meilleure solution possible pour ne pas se faire éliminer. Si le joueur décide de partir directement vers un bonus, il peut se faire tuer s'il se rapproche trop mais qu'il ne l'atteint pas. Le joueur peut aussi décider d'esquiver tous les autres joueurs afin de terminer le plus haut possible dans le classement (nous ne serons pas responsables du "camp" dans le jeu). Enfin, pour ne pas terminer sur une égalité, nous voulons que le labyrinthe rétrécisse à partir d'un certain nombre de tour sous la forme d'une spirale (comme dans Bomberman). Pour le moment le classement se fait selon la position du joueur mais nous prévoyons de mettre en place un système de points qui seront attribués en fonction des éliminations et de la position finale.

Déroulement d'une partie :

Début de partie : Le labyrinthe sera rempli de blocs aléatoires (nombre variable selon la taille du labyrinthe mais pour une taille 20*20, on dispose 180 blocs). Chaque joueur démarrera au bord du labyrinthe dans une position qui changera aussi selon le nombre de participants. Le nombre de blocs augmentera par paliers et non à chaque fois qu'un joueur en plus arrive. L'ordre des joueurs sera choisi aléatoirement avant le début de la partie

Milieu de partie : A chaque tour, le joueur pourra choisir parmi deux actions. Soit avancer, soit casser un mur. Le but est d'éliminer les autres joueurs et pour cela il faut marcher dessus c'est-à-dire qu'un joueur, lors de son tour, devra choisir l'option de casser un mur sur la case où le joueur adverse se trouve ou tout simplement avancer dessus pour l'éliminer. Le gagnant est le dernier survivant. Pour pouvoir tuer plus facilement les autres joueurs, des bonus seront dispersés sur le labyrinthe (pas forcément intégrés au jeu lorsque le projet sera remis le 6/03).

Fin de partie : Le gagnant est le dernier survivant. Le classement s'affiche pour le moment en temps réel et sans récapitulatif à la fin. Si au bout de 120 tours, il reste plusieurs joueurs, les joueurs restants seront remis dans le labyrinthe qui aura une taille plus petite pour ne pas que la partie dure trop longtemps.

Les images ont été prises sur le site suivant : https://www.spriters-resource.com/, et google images

Présentation des différentes parties du codes :

- Fonctions qui génèrent le labyrinthe :

```
void afficher_carte(int tab[SIZE][SIZE]){
    int i,j, k;
    system("cls");
    for(i=0;i<SIZE;i++){ //on parcourt un tableau 2d
        for(j=0;j<SIZE;j++){
                if(tab[i][j]==0) {      //Si la tableau vaut 0, on affiche un espace
                    printf(" ");}
            else if(tab[i][j]==1) {
                    printf("|");}      //S'il vaut 1, on affiche un mur
            else
                {printf("%d", tab[i][j]-1);} //
        }
        printf("\n");
    }
}
//return 0;
```

Cette fonction permet d'afficher la carte du labyrinthe. On parcourt tout simplement le labyrinthe avec une double boucle for et on met un mur si la case vaut 1 et un espace si elle vaut 0. Avant d'afficher la carte, on remplit le tableau de 0 ou de 1 avec une fonction qui parcourt le tableau avec une double boucle for et qui stocke dans chaque case un 0 ou un 1 qu'on obtient avec le return d'une fonction qui détermine un nombre aléatoire entre les bornes souhaitées (ici 0 ou 1) .

Voici tout d'abord la fonction qui renvoie un nombre aléatoire puis la fonction qui remplit le tableau aléatoirement avec des 0 ou des 1.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/*#ifdef _WIN32
#include <windows.h>
#endif
#include <time.h>*/


#define SIZE 20
#define joueurs 8

//srand(time(NULL));
int rand_a_b(int a, int b){
    //srand(time(NULL));
    return rand()%(b-a)+ a;  //retourne un entier aléatoire compris entre les nombres donnés
                             //lors de l'appel de la fonction
}
```

```c
void alea_carte(int tab[SIZE][SIZE]){ //pose un nombre défini de 1 dans un tableau
    int i,j;
    int nbrblocs;
    nbrblocs = 180; //le nombre de 1 à mettre
    printf("%d\n", nbrblocs);
    while(nbrblocs > 0){
        for(i = 0; i < SIZE; i++){
            for(j = 0; j < SIZE; j++){
                if(tab[i][j] == 0){ //si la case est vide, on remplit aléatoirement la case avec un 0 ou un 1
                    tab[i][j] = rand_a_b(0,2);

                    if (tab[i][j] == 1){ //si on a posé un 1 le nombre de blocs diminue de 1 jusqu'à obtenir 0
                            nbrblocs = nbrblocs - 1;
                    }
                }
            }
        }
    }

    if (nbrblocs !=0) {
        printf("faux\n"); //si jamais nombre de blocs inférieur à celui demandé, on le signale
    }
    tab[(SIZE/2)-1][(SIZE/2)-1] = 2; //on initialise la case centrale à 2 pour avoir le premier joueur dessus.
}
```

Avec cette fonction, on remplit le tableau aléatoirement jusqu'à avoir le nombre de blocs souhaités (ici 180). Si on parcourt plusieurs fois le tableau pour avoir le nombre de blocs demandé, on laisse les cases où il y a déjà un mur (case avec un 1) et on redéfinit un nombre à stocker dans les cases qui valent 0.

Ensuite, on initialise la position de chaque joueur dans le labyrinthe, position qui est déterminée aléatoirement. Pour cela, on utilise l'algorithme de Knuth. Néanmoins, cette fonction ne marche pas avec la SDL pour le moment. Nous avons donc initialisé les joueurs à la main.

Voici le code de cette fonction :

```c
void init_tab(int tab[SIZE][SIZE]){ //pose aléatoirement les joueurs sur la map
    int i,j, compt = 0, aleaX, aleaY, switcher;
    int nbjoueurs[SIZE*SIZE]; //liste des joueurs à placer( de même taille que le tableau)

    for(i=0; i < SIZE*SIZE; i++) nbjoueurs[i] = 0; //on vide le tableau pour initialiser nbjoueurs

    for(i=1; i < joueurs; i++) nbjoueurs[i] = i + 2; //on y met les joueurs

    for(i = 0; i < SIZE; i++){ //on reconie les joueurs dans le tableau
        for(j = 0; j < SIZE; j++){
           tab[i][j] = nbjoueurs[compt];
            compt++;
        }
    }
    for(i = 0; i < SIZE; i++){    //Algo de Knuth pour mélanger
        for(j = 0; j < SIZE; j++){
            aleaX = rand_a_b(0,SIZE);
            aleaY = rand_a_b(0,SIZE);
            switcher = tab[i][j];
            tab[i][j] = tab[aleaX][aleaY];
            tab[aleaX][aleaY] = switcher;
        }
    }
}
```

Fonctions qui permettent de se déplacer :

Avant de se déplacer, on détermine la position du joueur qui joue grâce à deux fonctions, une pour la ligne, une pour la colonne, qui parcourent le tableau jusqu'à trouver la position du joueur dedans avant de renvoyer cette position. Dans le programme, la case représentant le joueur 1 contient un 2, celle du joueur 2 contient un 3,etc. Ce décalage existe car une case représentant un mur contient un 1. C'est pour cela qu'on doit trouver la case où n+1 est présent pour chaque joueur n.

```c
int ligne_joueur(int tab[SIZE][SIZE], int n) {//retourne la ligne d'un joueur n et le numero du joueur
    int i, j;
    for(i=0; i<SIZE; i++){
        for (j=0; j<SIZE; j++){
            if (tab[i][j]==n+1){ /*on retourne si le tableau vaut le nombre au-dessus du joueur car
                                   la case où se trouve le J1 vaut 2, celle du J2 vaut 3,...
                                   puisqu'une case avec un 1 correspond à un mur*/
                return(i);
            }
        }
    }
}
int col_joueur(int tab[SIZE][SIZE], int n) {//retourne la colonne d'un joueur, n est le nimeo du joue
    int i,j;
    for(i=0; i<SIZE; i++){
        for (j=0; j<SIZE; j++){
            if (tab[i][j]==n+1){ /*même chose que pour la ligne*/
                return(j);
            }
        }
    }
}
```

Ensuite, grâce aux coordonnées du joueur, on peut décider s'il fait le déplacement souhaité ou non. Nous avons choisi 1 pour aller en haut, 2 pour le bas, 3 pour la droite, 4 pour la gauche et 5 pour pouvoir casser un mur. De plus, si l'option choisie est de casser un mur, il faudra faire un deuxième choix pour décider de la direction dans laquelle le mur sera cassé. Pour faire tout cela, nous avons utilisé un premier switch pour le déplacement et un deuxième si le joueur décide de casser un mur. De plus, nous avons rajouté des conditions pour éviter de sortir du labyrinthe ou de se déplacer sur

un mur (par exemple, un joueur sur la première ligne du tableau ne pourra pas se déplacer vers le haut même s'il tape 1. Dans ce cas, son tour se terminera sans qu'il n'ait fait quelque chose.).

Voici le code de cette fonction :

```
void avancer(int tab[SIZE][SIZE], int n){

int direction,cote;
direction=(rand()%5)+1;       //ici on décide que les ia jouent aléatoirement c'est à dire qu'elle choisisse aléatoirement la direction où allera vont
switch (direction) //switch pour différencier les actions
{
case 1 :
    if (ligne_joueur(tab,n)==0){   //Si l'ia est sur la première ligne du tableau et qu'elle décide d'aller en haut, elle ne pourra pas
        break;}
    if (tab[ligne_joueur(tab,n)-1][col_joueur(tab,n)]==1){ //Même chose s'il y a un mur, elle ne pourra pas aller sur la case
        break;}
    tab[ligne_joueur(tab,n)-1][col_joueur(tab,n)]=n+1;  //Si l'espace est libre et à l'intérieur du tableau, elle se déplace
    tab[ligne_joueur(tab,n)+1][col_joueur(tab,n)]=0;
    break;
case 2 :
    if (ligne_joueur(tab,n)==SIZE-1){       /*Si l'ia est sur la dernière ligne, elle ne peut pas aller en bas. S'il y a un mur, pareil.
                                              Si espace libre alors elle se déplace*/
        break;}
    if (tab[ligne_joueur(tab,n)+1][col_joueur(tab,n)]==1){
        break;}
    tab[ligne_joueur(tab,n)+1][col_joueur(tab,n)]=n+1;
    tab[ligne_joueur(tab,n)][col_joueur(tab,n)]=0;
    break;
case 3 :
    if(col_joueur(tab,n)==SIZE-1){  /*Si l'ia est sur la dernière colonne ou s'il y a un mur à droite, déplacement à droite
                                    impossible, sinon déplacement effectué*/
        break;}
    if (tab[ligne_joueur(tab,n)][col_joueur(tab,n)+1]==1){
        break;}

    tab[ligne_joueur(tab,n)][col_joueur(tab,n)+1]=n+1;
    tab[ligne_joueur(tab,n)][col_joueur(tab,n)]=0;
    break;
```

```
case 4 :      /*Même chose qu'avant mais pour la gauche (1ère colonne ou mur = déplacement impossible*/
    if (col_joueur(tab,n)==0){
        break;}
    if (tab[ligne_joueur(tab,n)][col_joueur(tab,n)-1]==1){
        break;}
    tab[ligne_joueur(tab,n)][col_joueur(tab,n)-1]=n+1;
    tab[ligne_joueur(tab,n)][col_joueur(tab,n)+1]=0;
    break;

case 5 : //après avoir choisi de casser un mur, on choisit la direction. Cette action permet de tuer un joueur
    cote=(rand()%4)+1;
    switch(cote)
    {
        case 1: //pour détruire un bloc au-dessus
            tab[ligne_joueur(tab,n)-1][col_joueur(tab,n)]=0;
            printf(" ");
            break;
        case 2 : //détruire un bloc en dessous
            tab[ligne_joueur(tab,n)+1][col_joueur(tab,n)]=0;
            printf(" ");
            break;
        case 3 : //permet de détruire un bloc à droite
            tab[ligne_joueur(tab,n)][col_joueur(tab,n)+1]=0;
            printf(" ");
            break;
        case 4 : //permet de détruire un bloc à gauche
            tab[ligne_joueur(tab,n)][col_joueur(tab,n)-1]=0;
            printf(" ");
        default :
            break;
    }

default:

    break;}}
```

Fonctions de l'ia et de visibilité du jeu :

Ensuite, nous avons créé une i.a qui choisit aléatoirement un chiffre parmi les 5 proposés. Le but de cette i.a est simplement de montrer que des i.a peuvent jouer dans notre arène et que plusieurs d'entre elles peuvent s'affronter. Cette i.a ne réalise que la fonction avancer vu précédemment et la fonction pause (ci-dessus). La fonction pause permettait de voir le tour de chaque ia mais en SDL nous avons utilisé un Wait event à la place. La fonction pause est donc inutie ici.

Voici le code de l'ia :

```
void ia(int tab[SIZE][SIZE],int joueur){ //une ia qui choisit aléatoirement sa direction
avancer(tab, joueur);
pause();
}
```

Fonction qui gère l'affichage grâce à la SDL :

```
void jouer(SDL_Surface* ecran)
{   // on initialise les pointeurs SDL qui nous permettront d'afficher , les murs...
    SDL_Surface *j1= NULL;
    SDL_Surface *j2= NULL;
    SDL_Surface *mur = NULL;
    SDL_Surface *blanc = NULL;
    SDL_Rect position, positionJoueur;
    SDL_Event event;


    int continuer = 1, i = 0, j = 0 , k=0;
    int carte[SIZE][SIZE];
    int numerojoueur=0;
    mur = SDL_LoadBMP("murr.bmp");// on affecte un pointeur aux differentes textures qu'on devra afficher
    j1=SDL_LoadBMP("J2.bmp");
    j2=SDL_LoadBMP("J1.bmp");
    blanc=SDL_LoadBMP("blanc.bmp");



    alea_carte(carte);// chargement de la carte (on la remplit de 0 et de 1 aleatoirement)


    while (continuer ==1)
    {
        SDL_WaitEvent(&event);
        switch(event.type)
        {
        case SDL_QUIT:
            continuer = 0;
            break;
        case SDL_KEYDOWN:
            switch(event.key.keysym.sym)
            {
            case SDLK_ESCAPE:
                continuer = 0;
                break;
            case SDLK_d:
```

```
// Effacement de l'écran
                SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 255, 255, 255));
                // on remet un ecran blanc par dessus le menu pour avoir la surface de jeu

// Placement des objets à l'écran


                for (i = 0 ; i < SIZE ; i++)
                {
                    for (j = 0 ; j < SIZE ; j++)
                    {
                        position.x = i * TAILLE_BLOC;
                        position.y = j * TAILLE_BLOC;

                        switch(carte[i][j])
                        {
                        case 1:
                            SDL_BlitSurface(mur, NULL, ecran, &position);//affiche un mur
                            break;

                        case 2 :
                            SDL_BlitSurface(j1, NULL, ecran, &position);// affiche le joueur 1
                            break;
                        case 3 :
                            SDL_BlitSurface(j2, NULL, ecran, &position);// affiche le joueur 2
                            break;
                        default:
                            break;

                        }
                    }
                }
                for(i=0;i<(joueurs*10);i++){
                        numerojoueur=numerojoueur+1;
```

```
            while(carte[ligne_joueur(carte,numerojoueur)][col_joueur(carte,numerojoueur)]==0){
        numerojoueur=numerojoueur+1;
    }
        ia(carte,numerojoueur);
        for (i = 0 ; i < SIZE ; i++)
    {
        for (j = 0 ; j < SIZE ; j++)
        {
            position.x = i * TAILLE_BLOC;
            position.y = j * TAILLE_BLOC;

            switch(carte[i][j])
            {
            case 0 :
                SDL_BlitSurface(blanc, NULL, ecran, &position);// permet de pas avoir de dedoublement du personnage dans l'affichage
                break;
            case 1:
                SDL_BlitSurface(mur, NULL, ecran, &position);// par court le tableau et a chaque fois qu'il croise un 1 il blitte un mur
                break;

            case 2 :
                SDL_BlitSurface(j1, NULL, ecran, &position);// affiche joueur 1 sur la carte
                break;
            case 3 :
                SDL_BlitSurface(j2, NULL, ecran, &position);// affiche joueur 2 sur la carte
                break;
            default:
                break;

            }
```

```
                        }
                    }
                }
                            if(numerojoueur==joueurs){
            numerojoueur=0; //Si on atteint la dernier joueur, on recommence à 1
           }
                }

                SDL_Flip(ecran);
                continuer=1;
                break;
            default :
                break;
            }
        default :
            break;

        }
    }
}
```

NB : Nous ne décrirons pas ni ne montrerons dans ce rapport l'initialisation des variables qui est normale et plutôt longue.

Tout d'abord pour effacer le menu on rajoute un ecran blanc puis on affiche la map une première fois avec la position de départ des joueurs et ensuit lorsque les I.A. ont jouer une première fois on réaffiche la carte avec leurs nouvelles positions et la nouvelle carte (s'ils ont casser des blocs)

Néanmoins, il faut aussi que les joueurs éliminés ne puissent pas rejouer. Pour cela on utilise une boucle while qui permet d'augmenter le numero du joueur qui va jouer de 1 si jamais la case du joueur vaut 0, c'est-à-dire que le joueur n'est plus visible sur la map (dans ce cas il est éliminé). Par exemple, si le joueur 2 a été éliminé, sa case vaudra 0 et donc après le tour du joueur 1, la variable numerojoueur va augmenter de 2 au lieu de 1 passant directement au joueur 3.

Création du menu :

Tout d'abord on affiche une image (qui nous servira d'écran d'accueil) à l'aide la SDL. On utilise les différentes commandes SDL.Puis on utilise la SDL event pour attendre que l'utilsateur lance le jeu.

```
int main(int argc, char *argv[])  ///Menu
{
    srand(time(NULL));
    SDL_Surface *ecran = NULL, *menu = NULL;// initialisation des pointeurs
    SDL_Rect positionMenu;
    SDL_Event event;

    int continuer = 1;

    SDL_Init(SDL_INIT_VIDEO);

    ecran = SDL_SetVideoMode(LARGEUR_FENETRE, HAUTEUR_FENETRE, 32, SDL_HWSURFACE | SDL_DOUBLEBUF);// on genere la fenetre d'ecran avec la bonne taille
    SDL_WM_SetCaption("Project Laby", NULL);// nom de la fenetre SDL

    menu = SDL_LoadBMP("menuu.bmp");// affichage du menu
    positionMenu.x = 0;
    positionMenu.y = 0;

    while (continuer)// permet d'attendre que le joueur appuie sur d pour lancer le jeu
    {
        SDL_WaitEvent(&event);
        switch(event.type)
        {
        case SDL_QUIT: // pour quitter le jeu
            continuer = 0;
            break;
        case SDL_KEYDOWN:
            switch(event.key.keysym.sym)// utilisation des keycode
            {
            case SDLK_ESCAPE: // echap pour quitter jeu
                continuer = 0;
                break;
            case SDLK_d: // appuyer sur d pour lancer le jeu
                jouer(ecran);
                break;
            }
        default:
            break;
        }
```

```
        // Effacement de l'écran
        SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 0, 0, 0));
        SDL_BlitSurface(menu, NULL, ecran, &positionMenu);
        SDL_Flip(ecran);
    }

    SDL_FreeSurface(menu);// on libere l'ecran
    SDL_Quit();

    return EXIT_SUCCESS;
}
```