UNIVERSITE DE BORDEAUX 1

# PROJETS TECHNOLOGIQUES LICENCE 2 INFORMATIQUE
# RAPPORT DU PROJET

4TIN403U-
GROUPE : TM4E
MEMBRES :
Thierno Sambegou DIALLO
Thierno Amadou DIALLO
Hadja Fatoumata DIAKITE
Seylim DJENG

15 Avril 2017

# Table des matières

# Chapitre 1

# LE PROJET :

Le projet consiste à réaliser un jeu appelé *Undead* en mode texte puis en mode graphique et de son solveur. Pour le mode texte dans un premier temps on utilise une grille de longueur 4 et largeur 4 dont certaines cases sont remplies par des MIRROIRS (/) et ANTIMIRROIRS (representés par antislash )et d'autres sont vides.Alors le but du jeu est de remplir chaque case de la grille qui ne contient pas de mirroir par un Zombie,un fantome ou un vampire.

- Les nombres de vampires/fantômes/zombies doiventêtre respectés.
- Les nombres autour de la grille indique combien de monstres sont visibles depuis le bord :
- Les zombies sont toujours visibles
- Les fantômes ne sont visibles qu'après une réflexion dans au moins un miroir
- Les vampires ne sont pas visibles après une réflexion dans un miroir

Dans un second temps le mode texte va evoluer avec cette fois ci une grille de longueur et largeur variables avec un nouveau type de monstre appelé Spirit( qui n'est jamais visible) et deux types de mirroirs : mirroirs vertical(|) et horizontal(-). Enfin,il faudra juste adapter cette derniere version du mode texte en mode graphique avec le sdl2.

Ce projet permettra de savoir utiliser les nombreux outils utiles et/ou nécessaires au développement. Les outils de compilations *Make* et *CMake*. Les logiciels de gestions de version *git*. L'outil de débogage *gdb* . L'outil de recherche des fuites mémoires *valgrind*. L'outil de couverture de code *gcov* et l'importance des tests. En plus,il permet de s'adapter au travail en equipe.

# Chapitre 2

# UNDEAD_TEXT

## 2.1 VERSION 1 avec taille de la grille 4*4

### 2.1.1 fichier undead_text.c

le fichier undead_text.c implemente une fonction que nous avons appelé *int affichage(game jeu* qui remplit une grille avec des mirroirs et l'affiche.Il faut rappelé que nous disposons d'une bibliothèque fournie par le professeur appelé game.h qui contient toutes fonctions nécessaires à l'implementation du jeu à savoir :

- *Une structure de type game ou cgame qui contient un jeu de type game*
- *game new_game()* :itinialise un jeu(grille 4*4)
- *game setup_new_game(int *labels[NB_DIR], content * board, int required_nb_ghosts, int required_nb_vampires, int required_nb_zombies)* : qui implemente le setup du jeu.
- *void add_mirror(game game, int dir, int col, int line)* :qui ajoute un mirroir à la position (col,line)
- *void set_required_nb_seen(game game, direction side, int pos, int value)* :ajoute le nombre de monstre visibles(value) sur un coté (side) à la position(pos).
- *void set_required_nb_monsters(game game, content monster, int value)* :ajoute le nombre (value) d'un type de monstre donné(monster)
- *game copy_game (cgame g_src)* :qui copie un jeu .
- *void delete_game (game g)* :qui supprime une grille si elle n'est plus utilisée(desallocation)
- *int required_nb_seen(cgame game, direction side, int pos)* :retourne le nombre de monstre visibles sur un coté donné à la position(pos).

- *content get_ content(cgame game, int col, int line)* :retourne le contenu d'une case à la position(col,line).
- *int required_ nb_ monsters(cgame game, content monster)* :retourne le nombre de monstres d'un type dans donné.
- *bool is_ game_ over (cgame g)* : retourne si la grille a été correctement remplie.
- *void restart_ game(game g)* :permet de reprendre le jeu
- *bool add_ monster(game game, content monster, int col, int line)* :ajoute un monstre et verifie ensuite si elle est réellement ajouté
- *int current_ nb_ seen(cgame game, direction side, int pos)* :retourne le nombre de monstre actuellement visible dans la grille à partir d'un coté et une position donnés
- *int current_ nb_ monsters(cgame game, content monster)* :retourne le nombre d'un type de monstre actuellement ajouté dans la grille

Avec ces fonctions,il suffit d'ajouter les monstres et les mirroirs.Pour jouer avec le terminal ,il faut lancer l'excutable generer(avec Makefile) puis on aura la forme :<x> <y> <G|V|Z> c'est à dire mettre les cordonnées(x,y) où ajouter le montre soit Z(zombie),V(Vampire) ou G(Ghost). voici ce que doit ressembler le jeu avant et après :

```
|   Z:5 V:2 G:2   |   Z:5 V:2 G:2   |
|                 |                 |
|     0 3 3 0     |     0 3 3 0     |
|                 |                 |
|  3  \     /  2  |  3  \ V V /  2  |
|  3  \        3  |  3  \ Z Z Z  3  |
|  2      \ /  0  |  2  Z G \ /  0  |
|  0  \     \  0  |  0  \ Z G \  0  |
|                 |                 |
|     0 3 2 3     |     0 3 2 3     |
|                 |                 |
|     start       |     solution    |
```

### 2.1.2 fichier game.c

: Maintenant il nous est demandé d'implementer nous nous meme les fonctions de la bibliothèque game.h qu'on nous avait preté dans un fichier appelé *game.c*. Nous avons choisit d'utiliser une structure game de la sorte

```
typedef unsigned int uint;
typedef struct game_s{
    uint **labels;
    content *board;
    uint nb_Z;
    uint nb_V;
    uint nb_G;
}game_s;
typedef struct game_s* game;
typedef enum content_e {EMPTY, MIRROR, ANTIMIRROR, VAMPIRE, GHOST, ZOMBIE} content;
```

- *uint **labels* :c'est le tableau à double dimension qui doit contenir les cases de la bordure de la grille pour stocker le nombre de monstre visibles sur un coté à une position donée.
- *content *board* :c'est la grille de 4*4
- *nb_Z* :le nombre de Zombie que contient le jeu
- *nb_V* :nombre de vampire.
- *nb_G* :nombre de Ghost.

Ensuite dans la fonction new_game nous avons alloué une variable de type game pour pouvoir initialisé les champs de notre structure,et ensuite implementer le reste des fonctions

```
game new_game(){
     game jeu = (game)malloc(sizeof(struct game_s));
     if (jeu == NULL) {
        fprintf(stderr, "error" );
        exit(EXIT_FAILURE);
     }
     jeu -> nb_Z = 0;
     jeu -> nb_V = 0;
     jeu -> nb_G = 0;
     jeu -> board = (content*)malloc(size*sizeof(content));
     if (jeu -> board == NULL) {
        fprintf(stderr, "error" );
        exit(EXIT_FAILURE);
     }
     for (int i=0; i < size; i++){
       jeu -> board[i] = EMPTY;
```

```
    }
    jeu -> labels =(uint**)malloc(NB_DIR*sizeof(uint *));
    if (jeu -> labels == NULL) {
       fprintf(stderr, "error" );
       exit(EXIT_FAILURE);
    }
    for (int i = 0; i < NB\_DIR; i++){
      jeu -> labels[i]=(uint *)malloc(NB\_DIR*sizeof(uint));
      if (jeu -> labels[i] == NULL) {
         fprintf(stderr, "error" );
         exit(EXIT_FAILURE);
      }
    }
    for (int x = 0; x < NB_DIR; x++){
      for (int y = 0; y < NB_DIR; y++){
        jeu -> labels[x][y] = 0;
      }
    }
    return jeu;
}
```

Après dans l'implementation des fonctions ,le seul souci que nous avions eu c'est au niveau de la fonction is_game_over car nous n'avons pas bien implementer la fonction current_nb_seen qui est appelée dedans.Mais nous l'avons corrigé à la deuxième pass.

### 2.1.3 les tests

Dans cette partie nous avons ecrit des fonctions qui testent est ce que réellemnt les fonctions que nous avons implementé dans game.c sont correctes sinon ils nous renvoient les erreurs rémarquées. Ces tests sont repartis dans les fichiers testAm.c testSAM.c testSEY.c test_DIAK.c . Chaque fichier de test est composé d'une fonction *main* exécutée au lancement de l'exécutable du test, comme celle-ci :

```
int main(void){
  bool result = true;
  result = test_new_game_ext() && result;
  result =  test_is_game_over () && result;
  result = test_add_mirror_ext() && result;
  result = test_copy_game() && result;
```

```
  result = test_add_mirror_ext() && result;
  result =  test_restart_game () && result;
  result = test_game_width() && result;
  result = test_game_height() && result;
  result = test_add_monster() && result;
  if(result){
      printf("Tests successfull\n");
      return EXIT_SUCCESS;
  }
  else{
      printf("Tests failed\n");
      return EXIT_FAILURE;
  }
}
```

## 2.2 UNDEAD_TEXT VERSION 2 avec taille variable

### 2.2.1 Nouveau game.c

Avec cette version la taille de la grille change et on a un nouveau type de monstre (SPIRIT). Donc on reprend la structure et les fonctions precedentes de game.c en tenant compte cette fois ci des dimensions et des nouvelles fonctions .

```
struct game_s{
  int required_nb_ghosts;
  int required_nb_vampires;
  int required_nb_zombies;
  int required_nb_spirits;
  int height;
  int width;
  int ** required_nb_seen;
  content * board;
}
```

- *int required_nb_ghosts* :nombre de ghost
- *int required_nb_vampires* : nombre de vampire
- *int required_nb_zombies* :nombre de zombies
- *int required_nb_spirits* :nombre de spirits
- *int height* :la hauteur de la grille
- *int width* :la largeur de la grille
- *int ** required_nb_seen* :tableaupour les bordures de la grille

— *content * board* :la grille

Pour implementer cette nouvelle version,il faut adapter la precedente tout en sachant que cette fois ci deux autres types de mirroirs :les mirroirs vertical et horizontal.Dans la nouvelle bibliothèque game.h il y aussi des nouvelles fonctions comme : game_height(game jeu) qui retourne la hauteur d'une grille de jeu passée en paramètre et aussi game_width(game) retourne la largeur. Comme la precedante nous avons eu certains problèmes d'allocation de memoire au niveau de la fonction new_game_ext pour allouer les bords de la grille(vue que les tailles varient ) mais nous l'avons reglé avec le valgrind et les tests. En plus l'adaptation de la fonction current_nb_seen a été compliquée vu les nouveaux mirroirs,chose qui a affecté aussi le is_game_over.

### 2.2.2 test et undead_text.c

on a repris les tests precedants tout en testant aussi les nouveaux paramètres apparus (mirrors et spirits). Pour un exemple de undead_text.c il suffit de faire appel en mettant les valeurs qu'on veut.exemple game jeu =new_game_ext(4, 4) et add_mirror_ext(game,ANTIMIRROR,0,2) pour la grille 4*4 donnée precedemment .

## 2.3 solveur

Dans cette partie,il fallait ecrire des fonctions qui chargent une grille à partir d'un fichier ,trouvent soit une solution(FIND_ONE) soit toutes les solutions(FIND_ALL) soit le nombre de solutions (NB_SOL) ensuite mettre la ou les solution(s) dans un ou des fichier(s) Pour generer l'executable (undeadd_solve) on a utilisé la methode Camke (CmakeList.text) . On le lance avec : undead_solve FIND_ONE|NB_SOL|FIND_ALL <nom_fichier_pb> <prefix_fichier_sol> nom_fichier_pb :fichier dans lequel se trouve la grille qu'il faut charger prefix_fichier_sol> :là où il faut mettre la /les solution(s).

### 2.3.1 le fichier game_io.c

: Dans ce fichier on a implementé deux fonctions :load_game qui charge le jeu depuis le fichier et save_game qui sauvegarde la solution dans un ou des fichier(s)

— *game load_game(char* filename)* : prend en paramètre un fichier. Pour l'implementer nous avons utilisé deux fonctions auxiliaires *char* read_next_line(FILE* p_f, long* p_size)* qui permet de lire une ligne dans un fichier puis de stocker chaque caractere(meme les espaces) dans une case d'un tableau et retourne ensuite ce tableau de caractere( tableau que nous avons allouer dynamiquement) ,ensuite *long* convert_line(char* line, long* p_size)* qui convertit un tableau de caracteres en un tableau d'entier de taille p_size (sans tenir compte des espaces) et retourne ce tableau ; puisque qu'il faut lire doit avoir un format fixe ,c'est à dire les 6 prèmieres lignes sont des des entiers qui sont la la hauteur,la largeur,le nombre de chaque type de monstre et les nombres de monstres visibles sur les bords(donc là on peut lire et convertir) et les autres lignes sont des caractères(mirroirs et cases vides donc ici on lit juste les lignes).

```
VOICI LE FORMAT:

<width> <height>
<required_nb_vampires>  <required_nb_ghosts> <required_nb_zombies> <required_nb_spirits>
<labels[n][0]> <labels[n][1]> ... <labels[n][width-1]>
<labels[s][0]> <labels[s][1]> ... <labels[s][width-1]>
<labels[e][0]> <labels[e][1]> ... <labels[e][height-1]>
<labels[w][0]> <labels[w][1]> ... <labels[w][height-1]>
<board[0][height-1]> <board[1][height-1]> ... <board[width-1][height-1]>
...
<board[0][1]> <board[1][1]> ... <board[width-1][1]>
<board[0][0]> <board[1][0]> ... <board[width-1][0]>
```

— *void save_game(cgame g, char* filename)* : cette fonction c'est apres avoir lancer le solver si on on trouve une/des solution(s) elles les stockent dans un/des fichier(s) sinon elle ecrit juste "pas de solution"

### 2.3.2 fichier solver.c

Le principe que nous avons utilisé est le suivant :en partant de l'inittialisation d'un jeu avec le load_game nous avons testé tous les deplacements possibles à l'aide de la fonction is_game_possible que nous avons ecrite et qui vérifie si on peut placer le monstre en question à la position indiquée si oui on le place sinon on fait marche arrière .Ensuite on verifie avec la fonction is_game_over si le jeu terminé sinon on teste d'autres possibilités. Le grand problème avec une telle methode est qu'elle demande énormément de memoire mais elle reste efficace. la fonction qui trouve une solution on l'a appelé sol_recursive. bool sol_recursive(game jeu, int position, bool retour) :elle commence à chercher à une position donné(position) et cherche toutes les possibilités et si elle trouve une solution elle retourne True(Vrai) sinon False(FAUX). C'est exactement la meme fonction fonction que nous avons utilisé pour le FIND_ONE en passant en parametre le jeu ensuite 0 comme position de depart et False(car pour un debut on a aucune solution) :bool s=sol_recursive(jeu,0,false) ; Le FIND_ALL :on a appelé aussi la sol_recursive une fois et si c'est true alors on l'appelle encore avec po pour position=longueur*largeur-1. on cherche une/des autres possibles la fonction sol_recursive mais cette fois avec comme condition "continue à chercher tant que tu trouve de solutions" traduction(while(s==true))

# Chapitre 3

# UNDEAD_SDL

## 3.1 Les difficultés rencontrées

Le but de cette dernière partie était de réaliser l'interface graphique de notre jeux Undead . Cette partie était impressionnante au premier abord . On ne savait pas vraiment comment s'y prendre car on était pas familier avec la SDL . Les fichiers fournis pour nous aider furent alors très utile, Aussi avant de passer à l'interface graphique on a de nouveau eu un bug sur notre fonction current_nb_seen que l'on à malheureusement pas pu corriger , cependant on avait la possibilité d'utiliser des game.o et game.io fonctionnels, c'est donc en utilisant ceux ci que nous avons pu mener à bien l'interface graphique.En plus nous avons quelques problemes de fuites de memoires dans la grille graphique pourtant nous avons liberé toutes les (SDL_Texture*) que nous avons alloué dans la structure.

## 3.2 la structure de SDL2

```
struct Env_t {
  SDL_Texture * background;
  SDL_Texture * zombie;
  SDL_Texture * ghost;
  SDL_Texture * vampire;
  SDL_Texture * spirits;
  SDL_Texture * point;
  SDL_Texture * textV;
  SDL_Texture * textG;
  SDL_Texture * textZ;
  SDL_Texture * textS;
  SDL_Texture * text;
  SDL_Texture * textR;
  SDL_Texture * textprincipe;
```

```
  SDL_Texture * textsolveur;
  int zombie_x, zombie_y;
  int ghost_x, ghost_y;
  int vampire_x, vampire_y;
  int spirit_x, spirit_y;
  int point_x, point_y;
  int i , j;
  game g;
};
```

## 3.3 Les fonctions

Dans laquelle nous avons défini les différentes textures et variables pour ainsi pouvoir les utilisé dans tout le code , car la consigne étais de ne pas utiliser de variable globale , pour avoir un code assez propre et clair. Ce fut la première étape , évidement notre structure ne ressemblait pas à ça au tout début , nous avons rajouté les éléments au fur et à mesure en fonction de nos besoins .

### 3.3.1 Fonction Init :

Cette fonction comme son nom l'indique avait pour but d'initialiser les différentes textures que ce soit le background ou encore les monstres ils furent tous initialisé dans cette fonction . Nous avons ainsi utilisé des « IMG_load_texture » pour charger les différentes images. Nous avons eu un léger soucis au niveau de l'affichage des required_nb_seen , car ces fonctions nous retournaient des entiers , il fallut donc les convertir en caractère pour pouvoir ainsi utiliser les fonctions « TTF_RenderText_Blended » et « SDL_createTextureFromSurface » nous permettant alors de créer des textures pour pour les entiers required_nb_seen .

### 3.3.2 Fonction Render :

C'est cette fonction qui permet l'affichage de tout ce qui fut initialisé dans la fonction init, C'est aussi dans cette fonction que l'on a défini notre grille, ce fut pour nous la partie la plus compliqué. Faire une grille en soit n'était pas si complexe mais c'est plutôt le fait de faire une grille capable de s'adapter en fonction du fichier de jeu donné en paramètre qui fut réellement complexe . Mais une fois ce problème résolu et la bonne formule trouvée , le reste du projet ne sembla plus si compliqué .

### 3.3.3 Fonction Process :

c'est dans cette fonction que l'on gère les événements de la souris et du clavier , c'est grâce à celle ci que l'on peut jouer au jeux en question. Notre stratégie à été d'utiliser le clavier pour jouer , on a placer un curseur qui commence en bas à gauche de la grille et lorsque l'on appui sur les touches up , down , left ou right ce curseur passe a la case suivante suivant la direction

indiqué. On à alors placé des curseurs x et y qui évolue en même temps que le curseur présent dans la grille , ainsi lorsque le curseur monte y = y + 1 , lorsqu'il descend , y = y - 1 , lorsqu'il va à gauche, x = x- 1 et enfin lorsqu'il va à droite x = x + 1. Ensuite selon que l'on appui sur V , G , Z ou S , un add_monster du monstre indiqué est utilisé pour l'ajouter dans la grille aux coordonnées x , y correspondent  voir figure (3.1 ).

### 3.3.4 Fonction clean :

Elle permet de vider le jeux et ainsi éviter les fuites mémoires.

## 3.4 le role des boutons :

- *Restart* :permet de reprendre le jeu.
- *solveur* :retourne le solveur c'est à dire la solution immediatement
- *About* :Pour savoir a propos du jeu.ET sur ce bouton nous avons esseyer de le faire fonctionner mais nous n'avons pas pu.

Pour se deplacer dans la grille sdl on utilise les 4 touches du clavier(HAUT,BAS,GAUCHE,DROITE) et en meme temps la petite souri sur l'ecran nous montre dans quelle cellule on est.Quand le jeu termine ça affiche you "CONGRUTALATION WIN"  4.2

**PTINCIPE DU JEU :**

**le jeu consiste à ajouter des monstres dans les cases vides d'une grille qui contient**

**des mirroirs vertical( |),horizontal( -),Antimirroir(\) et mirroirs(/)**

**• Les nombres de vampires/fantômes/zombies doivent être respectés.**

**• Les nombres autour de la grille indique combien de monstres sont visibles depuis le bord**

**les ghost(fantomes) :visibles après reflexions sur un mirroir**
**Ajout:placer la souris sur la case et taper la touche g sur le clavier**

**les spirits(esprits):ne sont jamais visibles**
**Ajout:placer la souris sur la case et taper la touche s sur le clavier**

**les vampires:ne sont pas visibles après reflexions sur un mirroirs**
**Ajout :placer la souris sur la case et taper la touche v sur le clavier**

**les Zombies:Toujours visibles**
**Ajout :placer la souris sur la case et taper la touche z sur le clavier**

**La souri qui nous permet de se deplacer dans la grille avec les touches du claveir(Haut,Bas,Gauche,Droite)**

FIGURE 3.1 – Commment jouer ?

# Chapitre 4

# ANNEXE :

LES COMMENTAIRES DE MEMBRES À PROPOS DE L'UE :

## 4.1 Conclusion de Thierno Sambegou Diallo :

Dans l'ensemble cette UE a été une veritable experience pour moi car chaque partie du projet pousse l'étudiant à faire des recherches ,des analyses profondes d'evènement.La decouverte des outils (cmake,makefile,valgrind....) et leurs utilisations va nous faciliter pour la réalisation d'autres projets individuels. Par contre,je trouve un peu severe le systeme de correction qui par exemple pour notre groupe,nous a doné 0 à cause d'une seule fonction ratée (cas : solveur premier rendu qui fonctionnait bien chez mais certainement lui il avait des soucis avec nous fonctions auxiliaires qui lis te convertis une ligne) ou encore parcequ'on a mis un espace de trop dans le Makefile ou parcequ'il n y a pas nommer les fonctions comme le veut le prof.Et d'un coup le groupe se retrouve avec 0,chose qui dessoit evidemment parcequ'on a l'impression qu'on a rien foutu du tout.Meme s'il y a l'idée de la deuxieme pass mais c'est un petit peu decevant de voir que c'est à la deuxieme tentative que tu reussis à cause juste d'un espace.

## 4.2 Conclusion de Seylim DIENG :

j'ai trouvé cette Ue assez bien dans l'ensemble , j'ai bien aimé le système de rendu régulier , ce fut une bonne méthode de suivi pour nous , nous permettant ainsi de ne pas prendre trop de retard . Il y a eu cependant des problèmes embêtant concernant ces rendus , notamment les bugs des systèmes de corrections automatiques , certaines fois le résultat attendu n'était pas faux mais il arrivait que l'on reçoive un 0 sans savoir pourquoi . Heureusement cependant les professeurs étaient à l'écoute et réglaient assez vite le problème. À part ça j'ai trouvé le suivi très bien , le forum sur l'ent nous permettant de poser nos questions fut aussi très utile pour remédier à nos problèmes. En conclusion je dirai que ce fut une bonne expérience et utile car je maîtrise dorénavant bon nombre d'outil de programmation .

## 4.3 Conclusion de Thierno Amadou Diallo :

cette UE a été d'une importance capitale pour moi car elle m'a permise d'apprendre non seulement des notions sur un langage que je n'avais jamais vu au paravant : le langage C et plusieurs outils très importants dans la programmation.Les outils de debuggage comme le gdb,valgrind mais aussi des outils d'optimisation de code. elle m'a aussi permit de travailler en groupe c'est qui n'est vraiment pas facile sans le git.En conclusion, apart quelques problemes rencontrés sur certains de nos rendu je trouve bien cette UE et j'esperes tirer profit sur tout ce qu'on a vu durant cette année.

## 4.4 conclusion de Hadja Fatoumata :

Tout d'accord ce cours m'a permis de comprendre comment fonctionne le système d'exploitation linux , connaitre les lignes de commandes permettant d'exécuter une programme. Le cours de projet technologique m'a aussi permis de connaitre les bases préliminaires de la programmation en langage C, de combler mes lacune dans l'écriture des algorithmes programmations c, chaque fonction coder dans ce projet m'a appris un peu plus et l'intérêt qu'avoir de travailler en groupe de ce partager les idées m'a permis de comprendre l'intérêt qu'à avoir de travailler en équipe. Cette UE m'a permis de comprendre qu'un projet ne se limite pas au fait que les codes fonctionnent ou que le jeu fonctionne mais aussi à penser à tester tous les fonctions utiliser pour faire le code qui a permis à concevoir le jeu. Les fonctions solveur malgré qu'elles n'ont pas été facile à programmer car elles ont fait preuves d'une grande réflexion ce qui nous a d'ailleurs couter en temps et aussi en points car on n'a pas réussi à Termier les fonctions à temps, mais elles nous aider à comprendre l'intérêt qu'a d'avoir le solveur, déjà ce plier au caprice du client, pour un jeu un client pourrai bien ce demander si le jeux admet bien une solution ou pas ou bien d'autre chose et un programme doit pouvoir satisfaire tous les caprices du clients et c'est bien le travail complet qui attend un programmeur L'interface graphique avec SDL des fonctions de programmations encore nouveau pour moi mais qui m'a beaucoup marqué et beaucoup apprécié et dont je crois qu'il est d'une grande importance dans la programmation car il m'a servi à comprendre comment transformer un programme en interfaces graphiques avec l'aide de toutes ces petites fonctions qui nous permet d'avoir toutes sortes de style d'interfaces a notre convenances . le SDL les l'une des fonctions qui m'a beaucoup plus marquer dans ce cours Le cours de projet technologique est l'une des UE les plus indispensables du semestre car il nous enseigne pas seulement les bases de la programmations mais aussi il nous forme à devenir un bon programmeur avec tous les atouts dont on aura besoins a l'avenirs et s'a serai bien que ça soit une suite continuelle car elle nous servira en nous perfectionner de plus.

## 4.5 EXEMPLES :

**Une grille vide fig 4.1 et cette grille après avoir été bien remplit fig 4.2**

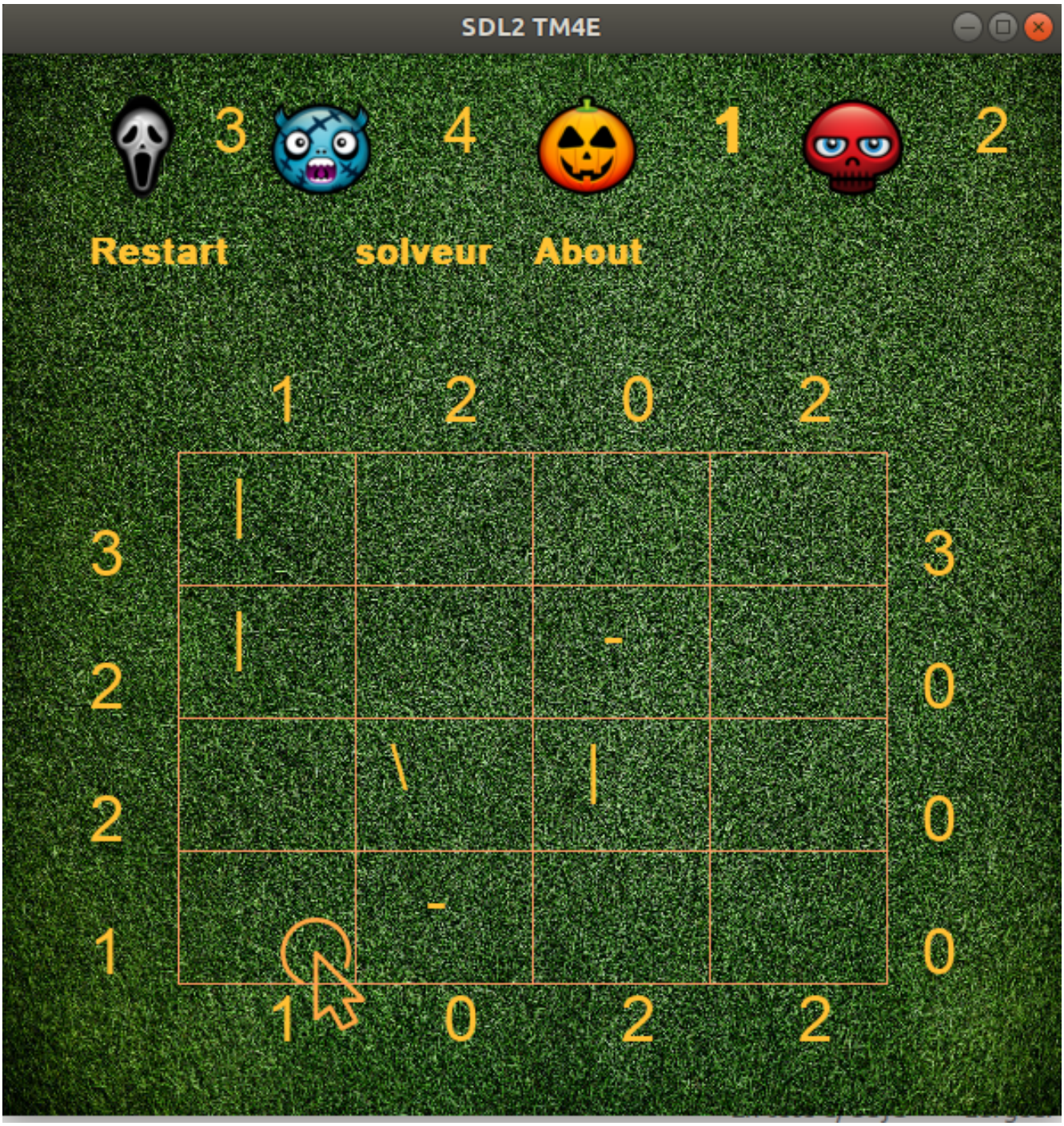


FIGURE 4.1 – Exemple d'une grille

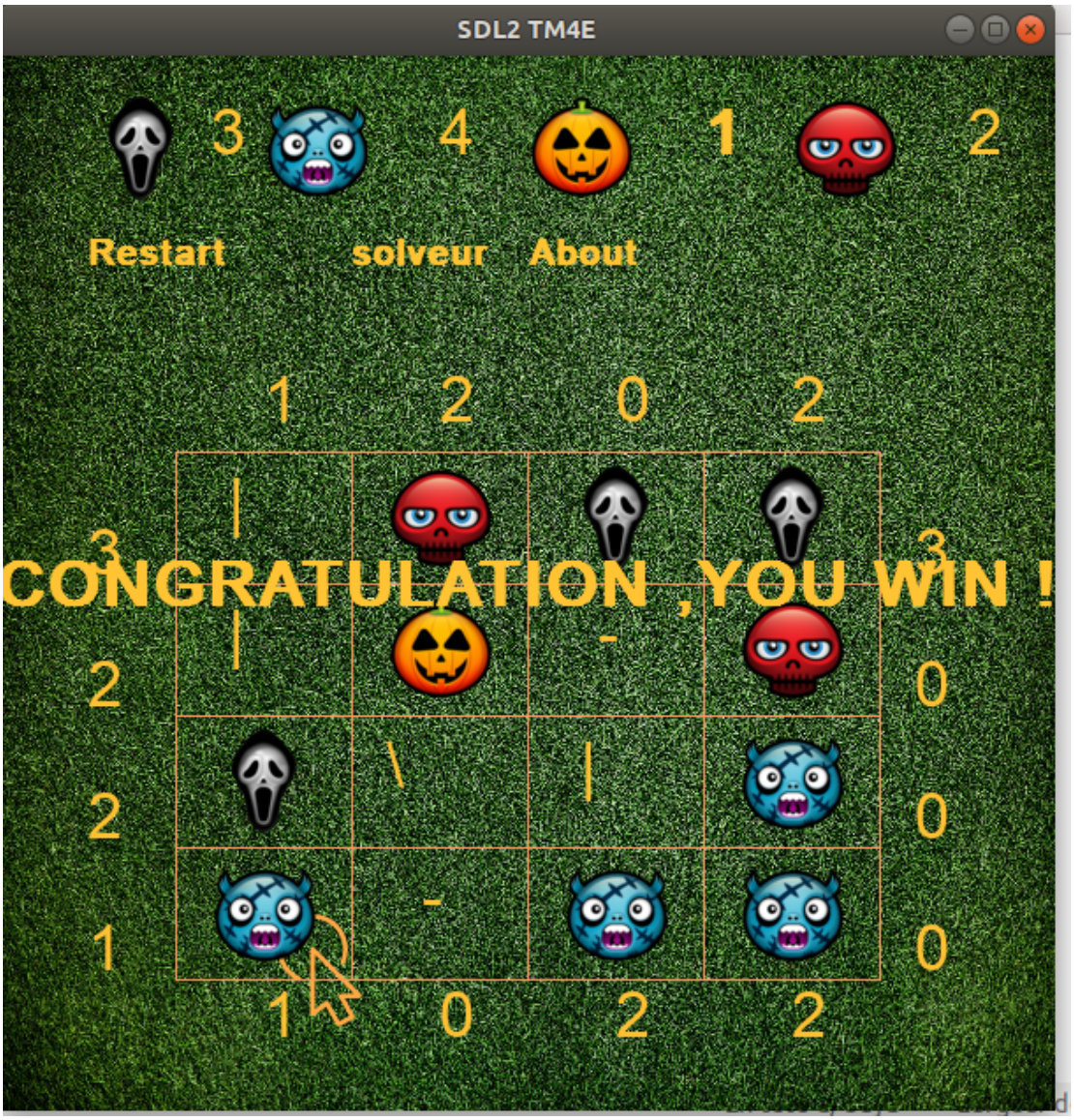


FIGURE 4.2 – Grille Apres avoir gagné