

Compte-rendu SAE11_2024 DEV1.1

TABLES DES MATIÈRES:

1) INTRODUCTION

- 1.1 Règles du jeu
- 1.2 Bibliothèques Graphique (IUT)
- 1.3 Répartition des tâches

2) FONCTIONNALITÉS

3) ORGANISATION DU CODE

- 3.1 Découpage fichiers.
- 3.2 Fichiers et utilité.
- 3.3 Avantage du découpage.
- 3.4 Diagramme.

4) EXPLICATION DES DONNÉES

- 4.1 Problèmes rencontrés (peut être)

5) CONCLUSION

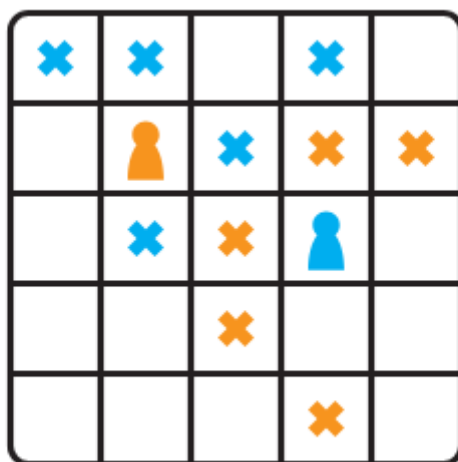
6) SOURCES

INTRODUCTION:

Dans le cadre de la SAé 1.1, nous avons pour consigne de réaliser le jeu **Blocus** en langage C. Celui-ci devait être créé en utilisant la bibliothèque graphique de l'IUT.

Notre Professeur nous a donné des instructions :

- L'interface de jeu sera contrôlée entièrement à la souris et consistera en 3 écrans :
 - 1er écran doit proposer de choisir la taille de la grille et de démarrer à 1 ou 2 joueurs.
 - 2ème écran la partie.
 - 3ème écran afficher le gagnant et de choisir entre terminer ou relancer une partie.



Règle du jeu :

Le blocus se déroule sur une grille dont la taille (entre 3x3 et 9x9) est définie au début de chaque partie. Deux joueurs (ou contre

l'ordinateur) s'affrontent en plaçant et déplaçant leurs pions sur la grille.

Déroulement:

Placement initial: Chaque joueur place son pion sur une case libre au début de la partie.

Tours de jeu: A chaque tour, un joueur doit:

- Déplacer son pion vers une case adjacente libre (horizontale, verticale ou diagonale).
- Condamner une case libre (qui est inaccessible jusqu'à la fin de la partie).

Condition de Victoire: Le premier joueur qui ne peut plus déplacer son pion perd la partie.

Bibliothèque graphique

Pour la réalisation de ce projet, une bibliothèque graphique a été mise à notre disposition, celle-ci nous fournit les bases de dessins. L'utilisation de cette bibliothèque graphique nous oblige à utiliser "lgraph" lors de la compilation, et à inclure "graph.h" dans le programme

Répartition des tâches :

Nathan : Il s'occupe de concevoir la grille du jeu et de son fonctionnement. Cela comprend la mise en place de la structure de données pour représenter la grille ainsi que les fonctions permettant d'interagir avec. Son travail assure l'affichage correct

des éléments du jeu, de sorte que les actions de joueurs soient bien intégrés dans l'interface

Lakshman : S'occupe de la création de l'écran d'accueil ainsi que de l'écran de fin. Cela inclut la conception de l'interface utilisateur, l'ajout des boutons de navigation, ainsi que des boutons pour relancer ou quitter le jeu à la fin. Son travail assure une expérience utilisateur plus intuitive, qui offre des transitions fluides entre le début et la fin de la partie.

FONCTIONNALITÉS:

Notre jeu dispose des fonctionnalités suivantes :

Menu d'accueil :

Au lancement du programme, on accède au menu suivant ;







En cliquant sur les flèches, on peut ajuster la taille de la grille.

Ensuite, en sélectionnant "1 player" ou "2 player" puis en appuyant sur le bouton "Select", le jeu se charge.

Jeu (Grille) :

La taille de la grille s'adapte en fonction de celle choisie dans le menu d'accueil, puis le jeu démarre automatiquement.

Exemple de Grille 3x3 et le placement des pions ainsi que des cases bloquées.

Ecran de fin:

Une interface s'affiche, indiquant le gagnant de la partie et offrant le choix de rejouer ou de quitter le jeu.



Organisation du code

Découpage fichiers

Le découpage du jeu Blocus a été structuré en plusieurs fichiers source afin d'améliorer la lisibilité, la maintenabilité et la modularité du code. Chaque fichier est dédié à une fonctionnalité spécifique, ce qui rend le code plus facile à comprendre et permet une gestion des fichiers et du code plus efficace.

Fichiers et Utilité

Sur le GIT se trouve un document README.md qui est un guide d'utilisation du programme.

home.h : Ce fichier d'en-tête définit la gestion de l'écran d'accueil du jeu, il définit la structure *home* pour plusieurs éléments de l'écran d'accueil : Les flèches pour ajuster la taille, les boutons, les images pour indiquer les options. Les fonctions *new_home* pour initialiser les boutons, les sprites et les paramètres par défaut (1 joueur grille 3x3), la fonction *draw_home* qui affiche les éléments d'interface du jeu, la fonction *update_home* qui réagit au changements d'état en fonction des choix actuels ou des choix sélectionnés par l'utilisateur, la fonction *mouse_click_home* qui gère les interactions de la souris par l'utilisateur. Il utilise également le fichier d'en-tête *utils.h* qui lui permet d'intégrer des fonctions utilitaires tels que *is_pressed_button*, *load_sprite*, *print_text*.

```
#ifndef HOME_H
#define HOME_H

#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include <string.h>
#include "/usr/include/X11/keysymdef.h"

#include "utils.h"

typedef struct home {
    int screen;
    Window window;
    Button up_button, down_button, select_button, one_player_button, two_players_button;
    int grid_size;
    int right_arrow_sprite;
    int left_arrow_sprite;
    int select_sprite;
    int one_player_sprite;
    int two_players_sprite;
    int one_player_selected_sprite;
    int two_players_selected_sprite;
    Sprites* sprites_manager;
    int number_players;
    int end;
} Home;

Home* new_home(Window window, Sprites* sprites_manager);

void draw_home(Home* home);

void update_home(Home* home);

int mouse_click_home(Home* home);

#endif /* HOME_H */
```

home.c : Ce fichier initialise l'interface de démarrage du jeu, permettant aux utilisateurs de définir les paramètres avant de lancer une partie, tels que la taille de la grille de jeu et le nombre de joueurs (1 ou 2). Le code utilise la structure *home* pour plusieurs éléments : Les flèches pour ajuster la taille, les boutons, les images pour indiquer les options. Les fonctions *new_home* pour initialiser les boutons, les sprites et les paramètres par défaut (1 joueur grille 3x3), la fonction *draw_home* qui affiche les éléments d'interface du jeu, la fonction *update_home* qui réagit au changements d'état en fonction des choix actuels ou des choix sélectionnés par l'utilisateur, la fonction *mouse_click_home* qui gère les interactions de la souris par l'utilisateur. Il utilise également le fichier d'en-tête *utils.h* qui lui permet d'intégrer des fonctions utilitaires tels que *is_pressed_button*, *load_sprite*, *print_text*.

```
Home* new_home(Window window, Sprites* sprites_manager) {
    Home* home = (Home*) malloc(sizeof(Home));

    home->screen = 1;
    home->window = window;

    home->grid_size = 3;
    home->up_button = new_button(600, 300, 30, 30);
    home->down_button = new_button(300, 300, 30, 30);
    home->select_button = new_button(400, 500, 280, 60);
    home->one_player_button = new_button(50, 400, 280, 60);
    home->two_players_button = new_button(50, 450, 280, 60);

    home->right_arrow_sprite = load_sprite("assets/right-arrow.png", sprites_manager);
    home->left_arrow_sprite = load_sprite("assets/left-arrow.png", sprites_manager);
    home->select_sprite = load_sprite("assets/select.png", sprites_manager);
    home->one_player_sprite = load_sprite("assets/one-player.png", sprites_manager);
    home->two_players_sprite = load_sprite("assets/two-players.png", sprites_manager);
    home->one_player_selected_sprite = load_sprite("assets/one-player-selected.png", sprites_manager);
    home->two_players_selected_sprite = load_sprite("assets/two-players-selected.png", sprites_manager);

    home->sprites_manager = sprites_manager;

    home->number_players = 1;

    home->end = 0;

    return home;
}
```

```
typedef struct home {
    int screen;
    Window window;
    Button up_button, down_button, select_button, one_player_button, two_players_button;
    int grid_size;
    int right_arrow_sprite;
    int left_arrow_sprite;
    int select_sprite;
    int one_player_sprite;
    int two_players_sprite;
```



```

void draw_home(Home* home) {
    char text[20];

    ChoisirEcran(home->screen);
    EffacerEcran(CouleurParNom("white"));

    AffichersSprite(home->right_arrow_sprite, home->up_button.x, home->up_button.y);
    AffichersSprite(home->left_arrow_sprite, home->down_button.x, home->down_button.y);
    AffichersSprite(home->select_sprite, home->select_button.x, home->select_button.y);

    AffichersSprite(home->one_player_selected_sprite, home->one_player_button.x, home->one_player_button.y);
    AffichersSprite(home->two_players_sprite, home->two_players_button.x, home->two_players_button.y);

    sprintf(text, "Taille %dx%d", home->grid_size, home->grid_size);
    print_text((home->window.width / 2) - (TailleChaineEcran(text, 2) / 2), 100, 2, "black", text);
}

```

```

int mouse_click_home(Home* home) {

    if(SourisCliquee()) {
        int x = _X, y = _Y;

        if(is_pressed_button(home->up_button, x, y)) {
            if(home->grid_size < 9) {
                home->grid_size++;
            }
            return 1;
        }

        if(is_pressed_button(home->down_button, x, y)) {
            if(home->grid_size > 3) {
                home->grid_size--;
                return 1;
            }
        }

        if(is_pressed_button(home->select_button, x, y)) {
            home->end = 1;
            return 1;
        }

        if(is_pressed_button(home->one_player_button, x, y)) {
            home->number_players = 1;
            return 1;
        }

        if (is_pressed_button(home->two_players_button, x, y)) {
            home->number_players = 2;
            return 1;
        }
    }

    return 0;
}

```

Pour avoir plus de détails vous pouvez consulter sur le git dans src/home.c

utils.h : Le fichier d'en-tête *utils.h* définit des structures et des fonctions utilitaires utilisées pour interagir avec l'interface graphique du jeu. Il inclut la structure *Button* (et ses fonctions *new_button* et *is_pressed_button* pour créer et vérifier les boutons), la structure *Sprites* pour la gestion des sprites (avec les fonctions *load_sprite* et *free_sprites*), ainsi que des fonctions pour afficher des écrans (*show_screen*), gérer les erreurs graphiques (*close_graph_error*), et afficher du texte (*print_text*).

```
#ifndef UTILS_H
#define UTILS_H

#include <stdlib.h>
#include <stdio.h>
#include <graph.h>

typedef struct button {
    int x;
    int y;
    int width;
    int height;
} Button;

typedef struct sprites {
    int count;
} Sprites;

typedef struct window {
    int width;
    int height;
} Window;

typedef struct coordinates {
    int i, j;
} Coordinates;

Button new_button(int x, int y, int width, int height);

unsigned int is_pressed_button(Button button, int x, int y);

int show_screen(int screen, int width, int height);

void close_graph_error(char* error);

int load_sprite(char* src, Sprites* manager);

void free_sprites(Sprites* manager);

void print_text(int x, int y, int size, char* color, char* text);

#endif /* UTILS_H */
```

utils.c : Le fichier *utils.c* fournit des fonctions utilitaires essentielles à la gestion de l'interface graphique du jeu. Il définit la structure *Button* pour gérer les boutons (avec les fonctions *new_button* pour créer des boutons et *is_pressed_button* pour vérifier si un bouton est cliqué). Il inclut également des fonctions pour afficher des écrans (*show_screen*), gérer des erreurs graphiques (*close_graph_error*), charger et libérer des sprites (*load_sprite*, *free_sprites*), et afficher du texte (*print_text*).

```
int is_pressed_button(Button button, int x, int y) {
    return x >= button.x && x <= button.x + button.width && y >= button.y && y <= button.y + button.height;

    return 0;
}
```

```
int load_sprite(char* src, Sprites* manager) {
    int sprite = ChargerSprite(src);

    if(sprite == -1) {
        close_graph_error("Erreur lors du chargement d'un sprite.\n");
    }

    manager->count++;

    return sprite;
}

void free_sprites(Sprites* manager) {
    int sprite;

    for(sprite = 1; sprite <= manager->count; sprite++) {
        LibereSprite(sprite);
    }
}

void print_text(int x, int y, int size, char* color, char* text) {
    ChoisirCouleurDessin(CouleurParNom(color));
    EcrireTexte(x, y, text, size);
}
```

Pour avoir plus de détails vous pouvez consulter sur le git dans *src/utils.c*

game.h : Le fichier d'en-tête *game.h* définit les structures et les fonctions nécessaires pour gérer une partie de 1 ou 2 joueurs. Ce fichier est inclus dans *game.c*, permettant à *game.c* d'accéder aux

structures comme *Game* et *Player* ainsi qu'aux fonctions liées. Ainsi, *game.c* peut manipuler et initialiser la partie, gérer le tour des joueurs et suivre l'état de jeu en utilisant les éléments spécifiés dans *game.h*.

```
#ifndef GAME_H
#define GAME_H

#include <graph.h>

#include "grid.h"

#define ONE_PLAYER 1
#define TWO_PLAYERS 2

#define PLACE_TYPE 1
#define BLOCK_TYPE 2

typedef struct player {
    int id;
    unsigned short int bot;
    int next_move[2];
} Player;

typedef struct game {
    unsigned int started; /* Partie débutée (booléen) */
    unsigned int ended; /* Partie terminée (booléen) */
    Sprites* sprites_manager;
    Window window;
    Grid grid;
    Player player1;
    Player player2;
    Player player_turn;
    int turn_type;
} Game;

/* Initialise une nouvelle partie */
Game* new_game(Sprites* sprites_manager, Window window, int number_players, Grid grid);

/* Démarre la partie */
void start_game(Game* game);

/* Passe au tour suivant */
void next_turn(Game* game);

/* Mets fin définitivement à la partie */
void stop_game(Game* game);

#endif /* GAME_H */
```

game.c: Ce fichier gère la partie de jeu d'un ou de 2 joueurs en utilisant les structures de *Player* et *Game*. Il utilise également les fichiers *grid.h* et *graph.h* pour intégrer la grille du jeu et les fonctions graphiques.

```

typedef struct player {
    int id;
    unsigned short int bot;
    int next_move[2];
} Player;

typedef struct game {
    unsigned int started; /* Partie débutée (booléen) */
    unsigned int ended; /* Partie terminée (booléen) */
    Sprites* sprites_manager;
    Window window;
    Grid grid;
    Player player1;
    Player player2;
    Player player_turn;
    int turn_type;
} Game;

```

Pour avoir plus de détails vous pouvez consulter sur le git dans `src/game.c`

[grid.h](#) : Le fichier d'en-tête [grid.h](#) définit les structures et fonctions nécessaires pour gérer et afficher une grille de jeu. La structure [Grid](#) comprend des informations de position, de taille, et de données de la grille, ainsi que des cases ([Button](#)) qui représentent les emplacements. Les fonctions associées permettent de créer la grille ([new_grid](#)), de l'afficher ([draw_grid](#)), et de détecter les interactions de l'utilisateur avec la grille ([get_box_clicked](#)). L'inclusion de l'en-tête [utils.h](#) est essentielle car elle fournit les structures et les fonctions pour la gestion des boutons.

```

/* Structure de la représentation d'une grille */
typedef struct grid {
    int originX;
    int originY;
    int side;
    int size;
    int screen;
    int** data;
    Button*** boxes;
    Window window;
    Sprites* sprites_manager;
    int* clicked_index;
    int orange_cross_sprite;
    int blue_cross_sprite;
    int orange_player_sprite;
    int blue_player_sprite;
} Grid;

/* Initialise une nouvelle grille */
Grid new_grid(Sprites* sprites_manager, Window window, int originX, int originY, int side, int size, int screen);

/* Dessine une grille vide */
void draw_grid(Grid grid);

void update_grid(Grid grid);

/* Renvoie si une case de la grille a été cliquée */
int box_clicked(Grid grid);

int is_clicked_box_is_next_to_player(int player_id, Grid grid);

```

Pour avoir plus de détails vous pouvez consulter sur le git dans `src/grid.h`

grid.c : Le fichier `grid.c` crée et affiche la grille du jeu. Pour initialiser la grille de façon précise, un schéma et des calculs ont été nécessaires, afin de définir une taille de grille adaptée. Il définit la structure *Grid* qui contient les informations de positions, de taille et les boutons représentant les cases de la grille. La fonction `draw_grid` dessine la grille sur l'écran, et `get_box_clicked` détecte la case cliquée. L'inclusion de `utils.h` pour les fonctions utilitaires liées aux boutons, ce qui permet de structurer et manipuler chaque case de la grille. Ce fichier crée une grille où chaque case est un bouton, et dessine des cases et lignes à l'écran pour une interface interactive du jeu.

```

typedef struct grid {
    int originX;
    int originY;
    int side;
    int size;
    int screen;
    int** data;
    Button** boxes;
    Sprites* sprites_manager;
    Windw window;
    int* clicked_index;
    int orange_cross_sprite;
    int blue_cross_sprite;
    int orange_player_sprite;
    int blue_player_sprite;
} Grid;

/* Initialise une nouvelle grille vide */
Grid new_grid(Sprites* sprites_manager, Windw window, int originX, int originY, int side, int size, int screen) {
    int i, j;
    int x, y;
    char src[30];

    Grid grid;

```

```

void draw_grid(Grid grid) {
    int i;
    int x = grid.originX;
    int y = grid.originY;
    int side = grid.side;
    int box = side / grid.size;

    ChoisirEcran(2);
    EffacerEcran(CouleurParNom("white"));

    DessinerRectangle(x, y, side, side);

    for(i = y + box; i < y + side; i += box) {
        DessinerSegment(x, i, x + side, i);
    }

    for(i = x + box; i < x + side; i += box) {
        DessinerSegment(i, y, i, y + side);
    }
}

```

Pour avoir plus de détails vous pouvez consulter sur le git dans `src/grid.c`

main.c : Le fichier est important car c'est l'initialisation et la gestion de l'interface graphique du jeu. Il commence par créer une partie et la grille, puis initialise l'écran d'accueil avec les boutons et les sprites. L'écran d'accueil et d'abord dessiné puis la grille est affichée à l'utilisateur. L'inclusion des fichiers d'en-tête [grid.h](#), [home.h](#), [game.h](#) et [utils.h](#) est essentielle pour fournir les fonctions et structures nécessaires à la création du jeu. [grid.h](#) est indispensable pour manipuler la grille. Des fonctions comme [newgrid\(\)](#) qui initialise la grille et [draw_grid](#) qui permet de dessiner la grille sur l'écran. Il définit aussi la structure [Grid](#) qui contient les informations relatives aux cases de la grille. [Game.h](#) est nécessaire pour le jeu, il définit des fonctions comme [newgame\(\)](#) pour initialiser une partie et des structures et fonctions qui permettent de suivre l'état du jeu si la partie est terminée ou en cours. [home.h](#) est nécessaire également car il définit la structure [home](#) pour l'écran d'accueil et contient des fonctions comme [new_home\(\)](#) pour initialiser l'écran, [draw_home\(\)](#) qui affiche les éléments de l'écran d'accueil et [update_home](#) pour gérer les mis à jours des éléments comme les bouton. [utils.h](#) est un fichier nécessaire au jeu car il contient des fonctions utilitaires pour le jeu.


```

if(!home->end) {
    if(mouse_click_home(home)) {
        update_home(home);

        if(home->end) {
            game = new_game(sprites_manager, window, home->number_players, new_grid(sprites_manager, window, gridx, gridy, side, home->grid_size, 2));
            start_game(game);
            show_screen(game->grid.screen, WIDTH, HEIGHT);
        }
    }
} else if(game->started) {
    update_grid(game->grid);

    if(game->player_turn.bot) {
        if(is_void_grid(game->grid)) {
            place_bot(game->player_turn.id, game->grid);
        } else {
            play_bot(game->player_turn.id, game->grid);
            next_turn(game);
        }
    } else if(box_clicked(game->grid)) {
        if(is_void_grid(game->grid)) {
            if(is_free_clicked_box(game->grid)) {
                move_player_to_clicked_box(game->player_turn.id, game->grid);
                next_turn(game);
            }
        } else {
            if(game->turn_type == PLACE_TYPE) {
                if(is_free_clicked_box(game->grid) && is_clicked_box_is_next_to_player(game->player_turn.id, game->grid)) {
                    move_player_to_clicked_box(game->player_turn.id, game->grid);
                    game->turn_type = BLOCK_TYPE;
                }
            } else {
                if(is_free_clicked_box(game->grid)) {
                    block_clicked_box(game->player_turn.id, game->grid);
                    next_turn(game);
                }
            }
        }
    }
}

```

Pour avoir plus de détails vous pouvez consulter sur le git dans `src/main.c`

end.c: Ce fichier gère l'affichage d'un écran de fin de jeu avec deux boutons interactifs : "Rejouer" et "Quitter".

```

3  typedef struct end {
4      int screen;
5      int winner_id;
6      Button quit_button, restart_button;
7      Sprites* sprites_manager;
8      Window window;
9      int quit_sprite;
10     int restart_sprite;
11 } End;
12
13 End* new_end(Window window, Sprites* sprites_manager) {
14     End* end = (End*)malloc(sizeof(End));
15
16     end->screen = 3;
17
18     end->quit_button = new_button(300, 300, 280, 60);
19     end->restart_button = new_button(600, 300, 280, 60);
20
21     end->quit_sprite = load_sprite("assets/quit.png", sprites_manager);
22     end->restart_sprite = load_sprite("assets/restart.png", sprites_manager);
23
24     return end;
25 }

```

Pour avoir plus de détails vous pouvez consulter sur le git dans `src/end.c`

`end.h` : Structure de la fin de la partie avec les bouton “Quit” et “Again” les images et le gagnant.

```
#ifndef END_H
#define END_H

#include "utils.h"

#define QUIT_ACTION 1
#define RESTART_ACTION 2

typedef struct end {
    int screen;
    int winner_id;
    Button quit_button, restart_button;
    Sprites* sprites_manager;
    Window window;
    int quit_sprite;
    int restart_sprite;
} End;

End* new_end(Window window, Sprites* sprites_manager);

void draw_end(End* end);

int get_action(End* end);

#endif /* END_H */
```

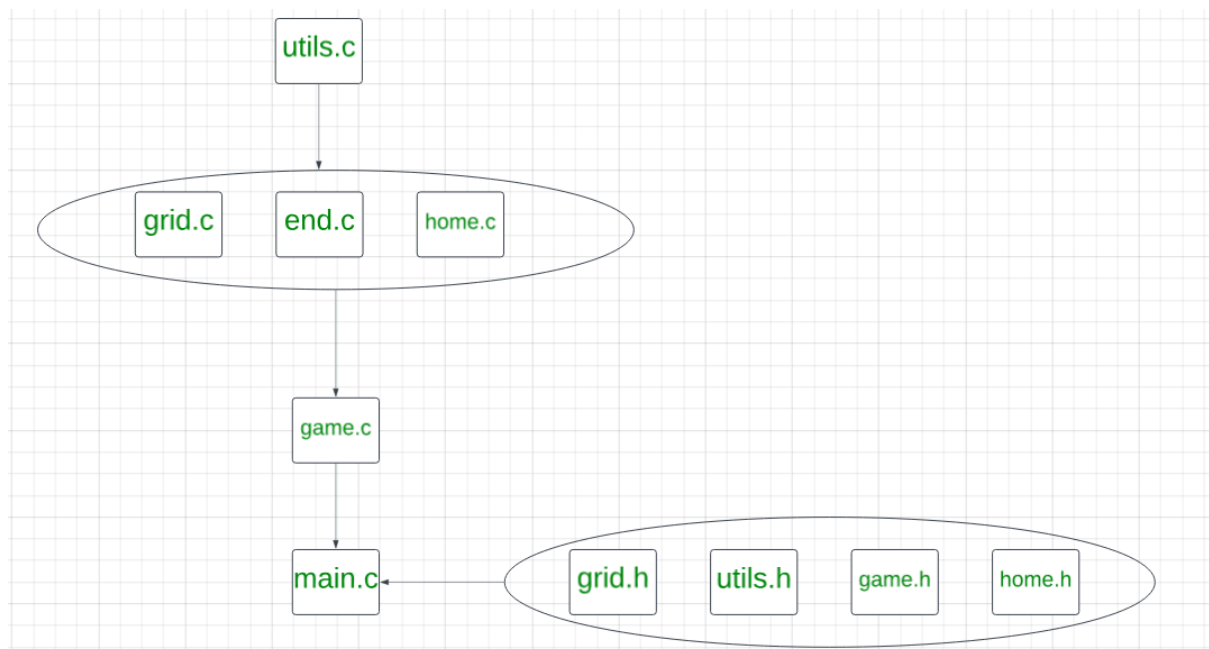
Avantage du Découpages

Lisibilité optimisée : En séparant chaque fonctionnalité dans des fichiers source distincts, le code devient plus intuitif et facile à suivre. Chaque fichier se concentre sur un aspect précis du projet, permettant de naviguer rapidement dans le code et de comprendre plus facilement le rôle de chaque fonction.

Amélioration de la maintenabilité : Grâce à l'organisation de plusieurs fichiers source distincts, il est plus simple d'apporter des modifications ou d'ajouter de nouvelles fonctionnalités sans perturber les autres parties du code. Les mises à jour se font de manière isolée, ce qui réduit les risques d'erreurs dans d'autres codes et facilite les tests. Cette structure rend également le code plus facile à déboguer et à adapter au fil du temps.

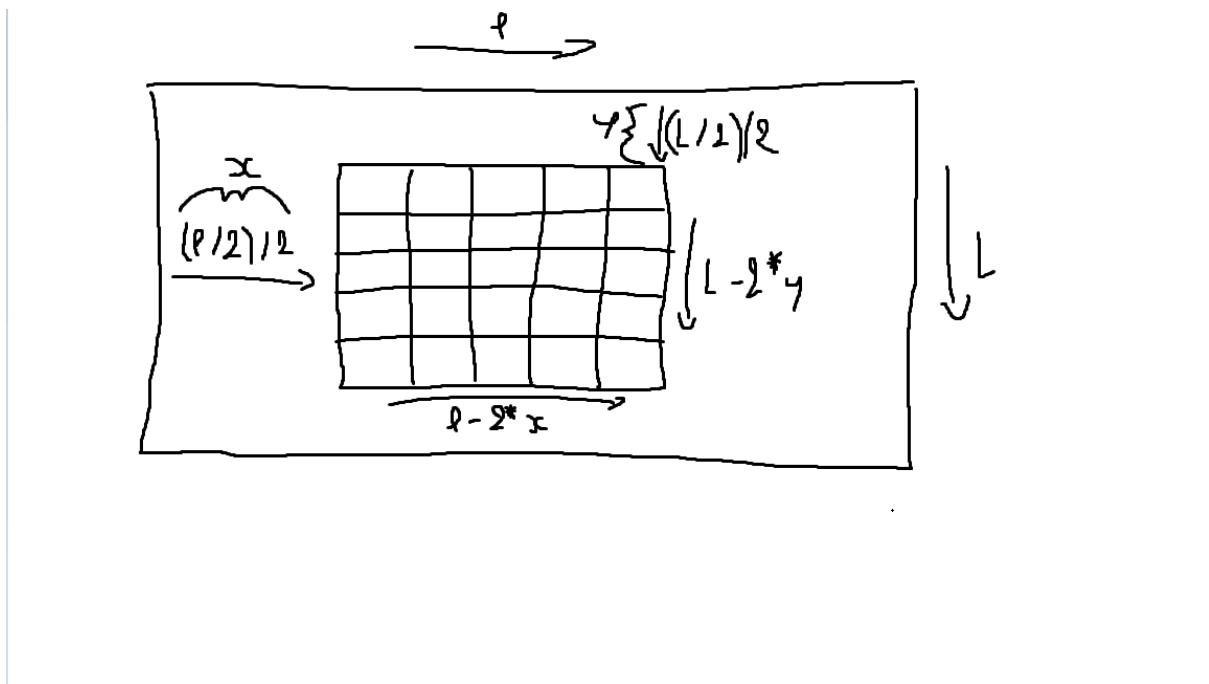
Réutilisation du code : Les fonctions peuvent être ré-utilisées dans d'autres codes et projets sans modification, facilitant le partage de code entre équipes ou entre projets, et accélère le développement global du projet.

Diagramme :



EXPLICATION DES DONNÉES

La grille est un élément clé du jeu. Elle représente l'espace où les joueurs interagissent en cliquant sur les cases pour se déplacer ou marquer des cases. La taille de la grille varie en fonction de ce que l'utilisateur choisit, la taille de la grille va de 3x3 à 9x9. Chaque grille a des dimensions spécifiques qui déterminent la taille des cases et en fonction de la centrer sur la fenêtre .
Voici comment nous avons fait le calcul des grilles



Voici les tailles de cases calculés pour les différentes grilles

Côté de la grille 500 :

- 3x3 -> Dimension d'une case 167x167
- 4x4 -> Dimension d'une case 125x125
- 5x5 -> Dimension d'une case 100x100
- 6x6 -> Dimension d'une case 84x84
- 7x7 -> Dimension d'une case 72x72
- 8x8 -> Dimension d'une case 63x63
- 9x9 -> Dimension d'une case 56x56

ATTENTION : l'image ne doit pas prendre l'entièreté de la case, il faut qu'elle puisse se positionner au milieu sans toucher les côté de la case, il faudrait prendre des mesures inférieures aux mesures que je t'ai donné à chaque fois.

Problèmes Rencontrés

Une fois les dimensions de la grille et des cases définies, nous avons rencontré un problème lié à l'affichage des images des joueurs et des cases bloquées. En effet, il ne suffisait pas seulement de calculer la taille des cases, mais aussi de s'assurer que les images placées à l'intérieur de ces cases soient bien positionnées, sans toucher les bords des cases.

Problème spécifique :

Les images (celles des joueurs ou des cases bloquées) doivent être **centrées** dans chaque case. Cependant, si les images occupaient l'entièreté de la case, elles seraient mal affichées, notamment sur les petites grilles. Il fallait donc redimensionner les images pour qu'elles ne couvrent pas toute la surface de la case et pour qu'elles restent bien positionnées au centre.

Solution : Redimensionnement des Images

Afin de résoudre ce problème, nous avons redimensionné toutes les images pour qu'elles s'adaptent aux dimensions spécifiques de chaque grille. Cela a permis de garantir que les images des joueurs et des cases bloquées soient bien centrées dans chaque case sans

dépasser les limites de la case. Les images ont ainsi été ajustées à des tailles inférieures aux dimensions des cases, en tenant compte des marges pour que l'image soit bien centrée sans toucher les bords.

Par exemple :

- Pour une grille **3x3**, la taille de la case est de **167x167 pixels**, mais l'image des joueurs ou des cases bloquées a été redimensionnée à environ **160x160 pixels** pour laisser une petite marge autour de l'image.
- Pour une grille **9x9**, où chaque case mesure **56x56 pixels**, l'image a été redimensionnée à environ **46x46 pixels** pour préserver la proportion et garantir qu'elle reste bien au centre.

CONCLUSION

Nathan BAUDRIER :

Ce projet a été une expérience particulièrement enrichissante et formatrice. Il m'a permis de mettre en pratique toutes les notions abordées depuis le début de l'année, tout en approfondissant mes connaissances en programmation. Le développement du jeu a également été l'occasion d'expérimenter le travail en équipe, un aspect essentiel dans ce type de projet. Grâce à l'utilisation de Git, j'ai considérablement amélioré mon organisation et appris à gérer efficacement les versions de code, ce qui a facilité la collaboration et le suivi des progrès. En somme, ce projet m'a permis de renforcer mes compétences techniques et humaines, tout en découvrant les défis et la satisfaction liés à la création d'une application ludique et interactive.

Lakshman MURALITHARAN :

Ce projet à été très enrichissant pour moi. Cela m'a permis de mettre en pratique ce que j'ai appris en cours en C. En travaillant sur ce projet, j'ai pu approfondir ma compréhension des différents aspects de la programmation en C, notamment les pointeurs, les structures et la manipulation des adresses que je trouvais difficile. Grâce à l'aide de mon camarade, j'ai pu surmonter certaines difficultés et mieux comprendre ces concepts qui n'était pas très clair pour moi.

Sur le plan organisationnel, le travail en équipe a été une expérience positive. Nous avons bien structuré notre travail en utilisant GIT qui nous a permis de travailler efficacement et fluidement.

Ce projet m'a aussi permis de développer des compétences en résolution de problèmes comme le problèmes des images en fonction des tailles de la grille que nous avons résolu en changeant toutes les dimensions des images.

Cette expérience n'est que bénéfique car elle m'a permis de progresser et de travailler en équipe.

Sources

Liens Images : https://www.flaticon.com/fr/icône-gratuite/personne_5631515
<https://www.istockphoto.com/fr/vectoriel/repères-de-coche-icône-croix-rouge-simple-vecteur-gm1131230925-299466418>
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.flaticon.com%2Ffr%2Ficône-gratuite%2Ffleche-droite_786199&psig=AOvVaw0JzWQ6-ivIX9C6khrolYbB&ust=1732541769669000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCJjv35aL9YkDFQAAAAAdAAAAABAE
<https://fr.textstudio.com/logo/generateur-de-polices-de-calligraphie-1344>

Puis redimensionnées et colorées sur paint.net.

Schéma : <https://lucid.app/>

