I Introduction

Avant toute chose : **bien lire tout le projet, plusieurs fois s'il le faut**, pour avoir en tête le travail à réaliser. **Partager-vous les tâches**.

Les travaux sont effectués en binôme : un travail copié ou effectué en collaboration entre N binômes divise les notes des binômes concernés par N.

Un rapport en LaTeX devra être rédigé. Il exposera les algorithmes utilisés, ainsi que l'utilisation de la SDL utilisée, les problèmes rencontrés.

L'archive zip contenant le code et le rapport manuscrit devra être rendue **le dimanche 5 janvier 2025 à 23h59 au plus tard**. Pour créer l'archive, suivre ces consignes :

1. Créer un répertoire nommé `NOM1_NOM2` (avec `NOM1` et `NOM2` le nom de famille de chacun des binômes).
2. Mettre dans ce répertoire les fichiers sources et d'en-tête.
3. Vérifier que tout compile et s'exécute correctement, puis **supprimer le fichier exécutable**, sinon l'archive ne pourra pas être reçue. Ce répertoire contiendra donc le rapport LaTeX, le PDF correspondant, ainsi que les fichiers source et d'en-tête uniquement.
4. Compresser le répertoire, qui fournira le fichier `NOM1_NOM2.zip`.

Le projet devra compiler ainsi et sans erreur :

```
gcc -g -O2 -Wall -Wextra -o tetris *.c $(pkg-config --cflags --libs sdl2  SDL2_ttf)
```

II Tetris

II.1 Le premier Tetris

Ce jeu a été créé par l'ingénieur soviétique Alekseï Pajitnov pendant l'année 1985. Il a été aidé par deux développeurs : Dmitri Pavlovski et Vadim Guerassimov. Le but du jeu est de déplacer latéralement des pièces, appelées *tetrimino*, qui défilent du haut vers le bas sur l'écran. Quand la pièce ne peut plus descendre (elle atteint le bas de l'écran ou bien est bloquée par un *tetrimino*), elle remplit le tableau et un autre tétromino apparaît en haut de l'écran et défile aussi vers le bas. Quand une ou plusieurs lignes sont remplies, elles disparaissent et les lignes au dessus descendent pour prendre leur place.

Le fait de remplir des lignes permet d'augmenter le score du jeu, et de passer à un niveau supérieur. Plus le niveau est élevé, plus le défilement des *tetrimino* vers le bas est rapide (*cf* III.3).

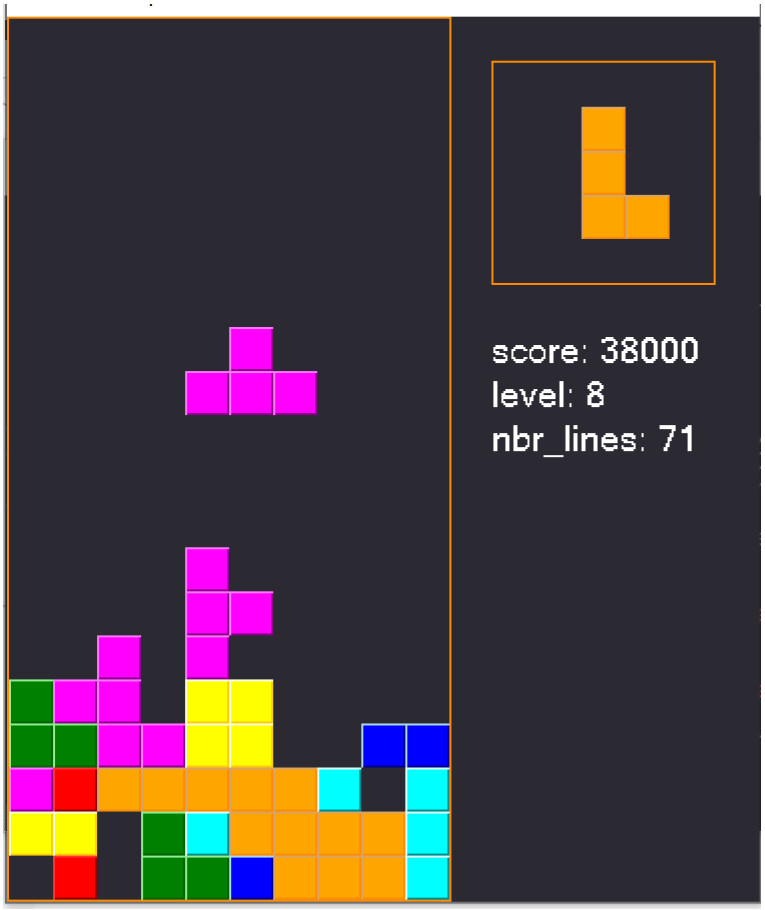
Ce jeu a eu à sa sortie énormément de succès. Il a été porté depuis sur tous les types d'ordinateur et d'environnement. De plus, en 1996, une société, nommée *The Tetris Company*, a pour but de faire respecter les droits de la licence du jeu, ainsi que les spécifications qu'un jeu nommé *Tetris* doit respecter.

II.2 Objectif du projet

Le but du projet est d'implémenter un tetris respectant une partie des spécifications du jeu. Voici de manière succinte la liste des tâches à acomplir :

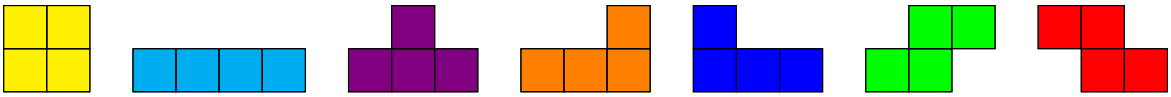
- défilement vers le bas d'un *tetrimino*,
- déplacement à l'aide des touches duclavier (gauche droit et vers le bas) et rotation d'un *tetrimino*,
- gestion du score (en particulier si plusieurs lignes sont détruites en même temps),
- gestion de la rapidité de défilement des *tetrimino*,
- affichage du *tetrimino* suivant, du score, du niveau et du nombre de lignes.

On utilisera la bibliothèque *SDL* pour la partie graphique. Exemple de rendu :



II.3 Glossaire

- **Bloc** : un block remplissant un élément de la **Matrice**.
- **Buffer** : une matrice de 20 lignes et 10 colonnes, invisible, au dessus de la **Matrice**, servant à représenter la position d'un *tetrimino* et à calculer les éventuelles collisions avec les *Blocs* de la *Matrice*. Les *tetrimino* partent du haut de cette matrice.
- **Matrice** : une matrice de 20 lignes et 10 colonnes, invisible, servant à représenter les **Blocs** visibles du jeu.
- **Mino** : un carré permettant de créer la forme d'un des 7 *tetrimino* possibles.
- **Tetrimino** : forme géométrique formée de 4 **Minos** connectés le long de leurs côtés, et représentées par une couleur, ainsi que par une lettre (similaire à leur forme). Les voici dans leur position quand ils apparaissent dans la **Matrice** (sans rotation) :



— Tetrimino `O` : jaune, forme carrée, 4 blocs 2 x 2.
— Tetrimino `I` : cyan, forme d'un I majuscule, 4 blocs sur une ligne.
— Tetrimino `T` : violet, forme d'un T majuscule, 3 blocs sur une ligne et un au centre au dessus.
— Tetrimino `L` : orange, forme d'un L majuscule, 3 blocs sur une ligne et un à droite au dessus.
— Tetrimino `J` : bleu, forme d'un J majuscule, 3 blocs sur une ligne et un à gauche au dessus.
— Tetrimino `S` : vert, forme d'un S majuscule, 2 blocs sur une ligne et 2 blocs sur une ligne à droite.
— Tetrimino `Z` : rouge, forme d'un S majuscule, 2 blocs sur une ligne et 2 blocs sur une ligne à gauche.

III Implémentation

Comme dans tout jeu (*cf* les jeux en L1), on commence par créer une structure de données. On modifie son contenu. Et en fonction de ce contenu, on affiche l'état du jeu à l'écran. Ici, ce sera principalement deux tableaux à deux dimensions, pour l'affichage des *tetrimino*, et un tableau à quatre dimension pour stocker la définition des *tetriminos* en fonction de leut type (`O`, `I`, ...), de leur rotation et d'un tableau à deux coordonnées.

III.1 Tableaux de jeu

Le tableau de jeu est un tableau à deux dimensions de 20 lignes et 10 colonnes. Ce tableau contiendra les *tetrimino* qui ont été bloqués. Vu me glossaire ci-dessus, on l'appellera dorénavant la **Matrice**.

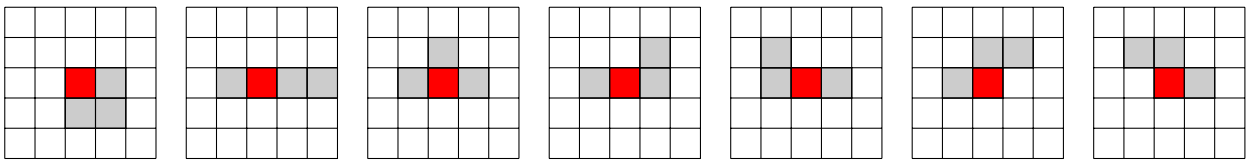
Pour faciliter la détection d'un *tetrimino* sur les côtés du tableau, ainsi que sur les **Blocs** existant, on va utiliser un autre tableau à deux dimensions de 20 lignes et 10 colonnes, que l'on appellera **Buffer**. C'est dans ce **Buffer** que le *tetrimino* qui défile bougera. Avant de le déplacer, on vérifiera s'il touche un bord, ou bien s'il touche un **Bloc**.

Le contenu de chaque élément d'un tel tableau sera le type du *mino* à afficher. Donc il y aura en tout 8 valeurs maximum (7 valeurs associées chacune à un *tetrimino* et une valeur choisie pour dire qu'un *mino* ne se trouve dans un des tableaux.

Un tel tableau peut donc se déclarer ainsi : `char tab[20][10]`.

III.2 Tetrimino

Voir la section II.3 pour une description. Pour faciliter la gestion de la rotation, ils seront stockés dans un tableau de 5 par 5 cases. Voici les tableaux lorsqu'il n'y a pas de rotation (le carré central, en rouge, sera utilisé pour la rotation des *tetrimino*, à l'exception du *tetrimino* `O` qui ne sera pas modifié par une rotation) :



Pour faciliter l'utilisation d'un *tetrimino*, on utilisera, comme mentionné dans la section III, le tableau

à 4 dimensions `char tetriminos[7][4][5][5];`. Ainsi `tetriminos[t][r][l][c]` correspondra au *tetrimino* de type `t`, pour la rotation `r`, les deux dernières coordonnées sont pour les cases du tableau 5x5 définissant le *tetrimino* correspondant, `l` pour la ligne et `c` pour la colonne de la case. Pour chaque *tetrimino* avec la rotation 0, les tableaux 5x5 sont donnés ci-dessus.

Il faudra remplir ce tableau pour chaque *tetrimino*, leur version tournée dans le sens des aiguilles d'une montre (`r=1` pour la rotation de 90° dans le sens des aiguilless d'une montre, etc...). On pourra le faire au choix à la main, ou bien avec des calculs.

III.3 Score

La gestion du score se fait en fonction du nombre de lignes éliminées. De ce score en découle immédiatement le niveau de jeu et la vitesse de défilement des *tetrimino*.

III.3.1 Calcul

Le score du joueur est calculé en fonction du nombre de lignes éliminées quand une pièce est bloquée, c'est-à-dire une, deux, trois ou quatre lignes. Cette dernière possibilité ne peut se faire qu'avec le *tetrimino* `I` et s'appelle *élimination Tetris*. La table ci-dessous résume le calcul du score :

Action	Total	Description
Simple	100 * niveau	1 ligne éliminée
Double	300 * niveau	2 lignes éliminées
Triple	500 * niveau	3 lignes éliminées
Tetris	800 * niveau	4 lignes éliminées

Donc plus on élimine des lignes avec un **tetrimino**, plus on gagne de points.

III.3.2 Niveau

Les niveaux vont de 1 à 15. Au début du jeu, le joueur commence avec le niveau 1. A chaque fois que 10 lignes ont été éliminées, le niveau augmente de 1. Le calcul se fait à chaque fois que au moins une ligne a été éliminée.

III.3.3 Vitesse de défilement

Quand un *tetrimino* est généré en haut du tableau, il descend immédiatement d'une ligne. A partir de là, le défilement se fait vers le bas. Le nombre de secondes écoulées pour passer d'une ligne à l'autre est calculée avec la formule suivante :

$$(0.8 - ((l-1) * 0.007))^{l-1} \ \forall l \in \{1, \ldots, 15\},$$

où l est le niveau en cours. Le tableau suivant présente une approximation des vitesses de défilement :

Niveau	Vitesse (secondes par ligne)
1	1.0
2	0.793
3	0.618
4	0.473
5	0.355
6	0.262
7	0.190
8	0.135
9	0.094
10	0.064
11	0.043
12	0.028
13	0.018
14	0.011
15	0.007

III.4 Initialisation

Avant de commencer à jouer, le programme doit au moins effectuer les tâches suivantes :

— allouer les resources,
— mettre le score et le nombre de lignes à 0, le niveau à 1.

III.5 Déroulement du jeu

Pendant que le jeu tourne :

1. on choisit le **tetrimino** qui va appaître, ainsi que le suivant, tous les deux en position sans rotation,
2. on positionne le **tetrimino** qui va apparaître **centré** sur le **Buffer**,
3. on gère les touches et on teste si la nouvelle position du **tetrimino** dans **Buffer** est possible (c'est-à-dire à sa future position, il recouvre ou non les **minos** dans **Matrice**).
4. on gère la nouvelle position du **tetrimino** à la ligne suivante est possible.
5. Deux cas sont possibles :
   (a) si le **tetrimino** peut être déplacé, on le déplace dans **Buffer** et on met à jour l'affichage dans la fenêtre.
   (b) sinon, on le copie du **Buffer** dans la **Matrice**, on met à jour l'affichage, et on retourne à l'étape 1. ci-dessus.

III.6 Fin de partie

Quand un nouveau *tetrimino* apparaît en haut du **Buffer**, s'il ne peut pas descendre, la partie se termine.

IV Description du programme

Le programme sera constitué de plusieurs fichier d'en-tête et source :

- *game.h* : contiendra la définition de la structure `Game` qui contiendra l'état du jeu (position de la ball, de la raquette, état du mur, fenêtre et renderer *SDL*, etc...). Elle contiendra aussi les déclarations de fonction nécessaires à la compilation.
- *main.c* : contiendra la fonction `main()`, elle appellera les fonctions démarrant le jeu.
- *game.c* : contiendra, entre autres, la définition des fonctions d'initialisation de la *SDL*, ainsi que la boucle des évènements de la *SDL*.
- *mino.c* : contiendra la fonction d'affichage d'un **mino** lorsqu'un **tetrimino** doit être affiché. La déclaration de la fonction se fera dans *mino.h*
- *tetris.c* : contiendra les fonctions de mise à jour de la position d'un **tetrimino** dans le **Buffer**, ainsi que les **tetrimino** qui sont déjà dans la **Matrice**. La déclaration des fonctions se fera dans *tetris.h*.

Voir ci-dessous pour plus de détails.

IV.1 Fonction `main()`

Elle se trouvera dans le fichier *main.c*. Elle effectue les tâches suivantes :

- initialisation du jeu,
- lancement de la boucle infinie des évènements,
- libération des ressources quand le jeu se termine.

```c
#include "game.h"

int main(int argc, char *argv[])
{
  Game *g;
  /* position de la fenetre */
  int pos_x;
  int pos_y;

  pos_x = 50;
  pos_y = 50;

  g = game_new(pos_x, pos_y);
  if (!g)
      return 1;

  game_run(g);

  game_del(g);

  return 0;

  (void)argc;
  (void)argv;
}
```

Ces trois fonctions seront déclarées dans le fichier *game.h* et définies dans *game.c*.

IV.2 La structure `Game`

La fonction `game_new()` alloue un pointeur vers la structure ci-dessous. Celle-ci sera à compléter (voir les instructions plus loin). Elle est définie dans le fichier *game.h*.

```
typedef struct
{
    SDL_Window *win;
    SDL_Renderer *ren;
    TTF_Font *font;

    Tetris *tet;
    int tet_offset_x;
    int tet_offset_y;

    int mino_size;

    Uint64 freq;
    Uint64 count;
} Game;
```

Concernant les membres de cette structure :

- `font` : pour afficher du texte.
- `tet` : contiendra entre autres les structures de donnée (**Buffer**, **Matrice** et les *tetrimino*. Voir plus loin.
- `tet_offset_x` et `tet_offset_x` permettent de déplacer le plateau de jeu pour mettre quelque chose autour (comme une ligne orange dans l'exemple ci-dessus).
- `mino_size` : la taille en pixels d'un *mino*. Il sera choisi de telle sorte que le plateau de jeu soit le plus grand possible à l'écran.
- `freq` et `count` permettront de déterminer le moment où on doit déplacer un *tetrimino*.

IV.3 Le fichier *game.c*

Il contient la définition de ces 4 fonctions :

- `game_new()` pour l'initialisation du jeu,
- `game_del()` pour la libération des ressources,
- `game_run()` pour le lancement de la boucle infinie,
- `game_update()` qui mettra à jour la fenêtre.

Voici le contenu du fichier qu'il faudra compléter. En particulier (mais pas seulement) les parties commentées avec `TODO`.

```
static void game_board_update(Game *g)
{
    SDL_SetRenderDrawColor(g->ren, 0x2b, 0x2a, 0x33, 0xff);
    SDL_RenderClear(g->ren);

    /* TODO : afficher le plateau de jeu */

    /* TODO : afficher le prochain tetrimino */

    /* TODO : afficher le score, le niveau et le nombre de lignes eliminees */
```

```
    SDL_RenderPresent(g->ren);
}

Game *game_new(int x, int y)
{
    /* TODO : a faire */
}

void game_new(Game *g)
{
    /* TODO : a faire */
}
```

```
  void game_run(Game *g)
  {
    int running;

    running = 1;

    while (running)
    {
        SDL_Event events;
        Uint64 c;

        while (SDL_PollEvent(&events))
        {
            switch (events.type)
            {
                case SDL_QUIT:
                {
                    running = 0;
                    break;
                }
                case SDL_KEYDOWN:
                {
                    switch (events.key.keysym.sym)
                    {
                        case SDLK_q:
                        {
                            running = 0;
                            break;
                          }
                        /*
                         * TODO: gérer les touches suivantes
                         * 'Esc' : pour sortir du jeu
                         * 'Espace' : pour descendre immediatement le
                         * tetrimino et mettre a jour
                         * fleche 'gauche' : pour deplacer le tetrimino
                         * a gauche et mettre a jour
                         * fleche 'droite' : pour deplacer le tetrimino
```

```
                    * a droite et mettre a jour
                    * fleche 'bas' : pour deplacer le tetrimino
                    * vers le bas et mettre a jour
                    * fleche 'haut' : pour pivoter dans le sens
                    * horaire le tetrimino et mettre a jour
                    * 'n' : pour pivoter dans le sens anti-horaire
                    * le tetrimino et mettre a jour
                    */
                }
                break;
            }
        }
    }

    c = SDL_GetPerformanceCounter();

    /* TODO : appeler game_board_update(g); quand cela est
    necessaire */
    if ((float)(c - g->count) / g->freq > tetris_get_drop_speed(g->tet))
    {
        g->count = c;
        if (tetris_can_go_down(g->tet))
        {
            game_board_update(g);
        }
        else
        {
            /*
             * detecter si une ou plusieurs lignes doivent etre retirees
             * si c'est le cas, mettre a jour le score, le niveau
             * et le nombre de lignes
             */
            tetris_shift_board(g->tet);

            /* nouveau tetrimino */
            tetris_reset(g->tet);
            game_board_update(g);
        }
    }
  }
}
```

IV.4 Le fichier *tetris.c*

Ce fichier contiendra :

- le tableau `tetrimino` qui est décrit dans la section III.2. Comme mentionné, il faudra le remplir.
- une énumération pour le type de chacun des sept **tetrimino**, il aura pour type

```
char tetrominos[7][4][5][5];
```

- la définition de la structure `Tetris` :

```c
struct Tetris
{
    char matrix[20][10];
    char buffer[20][10];

    Type current_type;
    int current_line;
    int current_column;
    int current_rotation;

    Type next_type;

    int level;
    int score;
    int nbr_lines;
    float drop_speed[15]; /* in s */
};
```

— la définition des fonctions, ainsi que du type `Tetris`, du fichier *tetris.h* ci-après.

IV.5 Le fichier *tetris.h*

Le type des **tetrimino** sera

```c
typedef enum
{
    TYPE_I,
    TYPE_J,
    TYPE_L,
    TYPE_O,
    TYPE_S,
    TYPE_T,
    TYPE_Z
} Type;
```

Il y aura **au moins** les fonctions suivantes :

```c
/* cree la structure Tetris pour un nouveau jeu */
Tetris *tetris_new();

/* libere les ressources */
void tetris_del(Tetris *tet);

/* met en place le nouveau tetrimino */
void tetris_reset(Tetris *tet);

/* tests du tetrimino qui defile */
int tetris_can_go_left(Tetris *tet);
int tetris_can_go_right(Tetris *tet);
int tetris_can_rotate(Tetris *tet);
int tetris_can_go_down(Tetris *tet);
```

IV.6 Le fichier *mino.c*

Un **mino** sera affiché comme un carré de côté `mino_size` pixels. Comme indiqué dans la section II.3, chacun des **tetrimino** a une couleur. Le **mino** n'aura pas une couleur uniforme. La ligne du haut et la colonne de gauche sera dans la couleur claire de la couleur du **tetrimino**. On complètera la ligne du bas et celle de droite par la couleur sombre. Cela donnera une sorte de relief au **mino**.

Pour avoir les composantes rouge, vert et bleue de chaque couleur, ainsi que leur version claire et sombre, vous pourrez consulter cette liste.

La fonction qui affichera un **mino** aura pour déclaration

```
void mino_display(Game *g, Type t, int l, int c);
```

où `t` est le type du **tetrimino**, et `l` et `c` sont respectivement la ligne et la colonne dans les tableaux **Buffer** et **Matrice**.