Tetris

Komba DOUMBIA et Louka JOVANOVIC

December 2024

Table des matières

Chapitre 1

Introduction

Le but de ce projet est de créer une jeu vidéo Tetris. Créer par Alekseï Pajitnov à partir de 1985, le but de ce jeux est d'empiler des structures géométriques appelées tetrimino les une sur les autre en formant des lignes complète afin d'augmenter le score. Le jeu dure autant de temps que le joueur arrive à ne pas créer une structure touchant le haut du plateau. Les consignes du projet nous on permis de structurer la création du jeu, et de comprendre a quoi servent les fichiers, les fonctions et comment les utiliser. Nous avons eu un dealait de deux semaines pour programmer le jeux en langage C et pour effectuer le rapport latex. Nous avons en premier lieu programmer le jeux puis écrit le rapport. Les doit respecter plusieurs condition.

1. Le tetrimino défile vers le bas.
2. Déplacement possible à l'aide des touches du clavier.
3. Gestion du score.
4. Gestion de la vitesse à la quelle défilent les tetriminos.
5. Affichage du tetrimino suivant, du score, du niveau et du nombre de lignes.

Le calcule du score se fait selon les règles suivantes :

$$\begin{aligned}
\text{Si une ligne est éliminée : score } &+= 100 \times \text{niveau} \\
\text{Si deux lignes sont éliminées : score } &+= 300 \times \text{niveau} \\
\text{Si trois lignes sont éliminées : score } &+= 500 \times \text{niveau} \\
\text{Si cinq lignes sont éliminées : score } &+= 800 \times \text{niveau}
\end{aligned}$$

Les niveau vont de 1 à 15. La vitesse de descente se calcule à l'aide la fonction suivante $(0.8-((l-1)\times 0.007))^{l-1}$. Où l est égale au niveau. Le jeu doit pouvoir s'initialiser correctement en allouant la mémoire et en affectant le score et le nombre de lignes à 0, et le niveau à 1. Il faut également que le jeu initialise un premiers tetrimino centré dans le buffer et afficher le prochain tetrimino en haut à droite. Il faut pouvoir gérer les touches d'évènements, gérer les possibilités de déplacement, gérer les nouvelles positions possible du tetrimino et mettre à jour et si le tetrimino ne peut plus descendre, il faut le copier dans la matrice et recommencer. Le jeu se termine si un tetrimino touche le haut du plateau.
Pour présenter notre travail, nous allons tout d'abord présenter le fichier main.c qui est le corps même du jeux, puis nous allons présenter les fichers d'en-tête, qui sont nécessaires pour le bon fonctionnement du programme. Puis nous allons présenter les fichiers ".c" et les fonctions qui les constitues. Dans cette dernière partie nous allons commencer par présenter le fichier mino.c qui est le plus court et qui est utilisé dans le fichier game.c, par la suite nous allons présenter le fichier tetris.c dans lequel nous avons le plus de fonction et qui est le pilier du jeu permettant une évolution dans le jeu, et enfin nous allons présenter le fichier game.c dans lequel réside l'utilisation de la SDL, qui nous permet d'avoir un affichage et une dynamique d'évènement.

Chapitre 2

main.c

```
1  #include "game.h"
2
3  int main(int argc, char *argv[])
4  {
5    // Initialisation de la SDL qui est obligatoire pour utiliser la SDL
6    if (SDL_Init(SDL_INIT_VIDEO) < 0)
7      {
8        //Gérer l'erreur
9        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
10       return 0;
11     }
12   //Initialisation de la bibliothèque TTF
13   if(TTF_Init() < 0)
14     {
15       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
16       return 0;
17     }
18
19   Game *g;
20
21   int pos_x = SDL_WINDOWPOS_CENTERED;
22   int pos_y = SDL_WINDOWPOS_CENTERED;
23
24   g = game_new(pos_x, pos_y);
25
26   if (!g)
27     return 1;
28
29
30       game_run(g);
31
32       game_del(g);
33
34   return 0;
35
36   (void)argc;
37   (void)argv;
38 }
```

En premier lieu le code présent sur le projet semblait fini et ne semblait pas nécessiter de changement supplémentaire. Cependant, en réalité il était nécessaire d'y ajouter certaines lignes.

- Les ligne 6 et 13 devait être rajoutées pour initialiser la SDL et TTF afin de pouvoir les utiliser. Initialement nous ne savions pas ou les mettre, donc ils étaient dans toutes les fonctions utilisant la SDL et TTF, mais après avoir étudier la SDL sur le site zestedesavoir et avoir parcouru différents forums sur la SDL et TTF, nous avons fini par comprendre qu'une seul initialisation de la SDL et de TTF était nécessaire et également une seul fonction respective pour les quitter suffisait, ce qui a été fait dans le fonction "game_del" 4.3.4. Pour initialiser la SDL et TTF il fallait faire un teste sur "SDL_Init" et "TTF_Init" pour vérifier que l'initialisation c'est bien faite et retourner une erreur si le teste était faut.

- Les ligne 22 et 23 donne la position supérieur gauche de la fenêtre. Elles sont affectées respectivement à "SDL_WINDOWPOS_CENTERED" afin de pouvoir centrer la fenêtre sur l'écran de l'utilisateur.

Chapitre 3

Les fichiers d'en-tête

Avant de présenter les fichiers C nous allons voir les trois fichiers d'en-tête du programme nécessaire pour le bon fonctionnement du programme, qui sont les suivantes :

1. game.h 3.1
2. tetris.h 3.2
3. mino.h 3.3

Il est aussi important de préciser que dans les trois fichiers sources qui seront présentés, les lignes 1, 2 et la dernière ligne auront la même utilité, qui permettra d'éviter les inclusions multiples. La différence sera dans la nomination des fichiers.

3.1 game.h

```
1  #ifndef GAME_H
2  #define GAME_H
3  #include "tetris.h"
4
5  typedef struct
6  {
7    //Texture pour écrire "score", "nbr_lines" et "level"
8    SDL_Texture *tex_s;
9    SDL_Texture *tex_l;
10   SDL_Texture *tex_nl;
11
12   SDL_Window *win;
13   SDL_Renderer *ren;
14   TTF_Font *font;
15
16   Tetris *tet;
17
18   int tet_offset_x;
19   int tet_offset_y;
20   int mino_size;
21
22   Uint64 freq;
23   Uint64 count;
24 } Game;
25
26 //Affichage du jeu
27 static void game_board_update(Game *g);
28
29 //Création d'une nouvelle partie
30 Game *game_new(int x, int y);
31
32 //Boucle infini contenant également les différents évènement du jeux
33 void game_run(Game *g);
34
35 //Libération de la mémoire, quite SDL et TTF
36 void game_del(Game *g);
37
38 #endif
```

3.1.1 Les inclusions

Le fichier game.h est inclue dans le fichier game.c 4.3 et main.c 2 . Il n'y est inclue que le fichier tetris.h 3.2,car il était nécessaire de l'inclure pour que le fichier game.h puisse identifier la structure Tetris, donc pour éviter des répétitions d'inclusion, nous avons préféré inclure tout les fichiers nécessaires dans tetris.h uniquement.

3.1.2 La structure Game

Initialement comme pour la fonction main.c, nous pensions que la structure Game était achevée et ne nécessitait pas d'ajout ou de modification. Cependant pour pouvoir afficher le score, le niveau et le nombre de ligne détruite nous sommes passés par une méthode nécessitant la création de structure, méthode que nous utilisons dans la fonction "game_board_update()", dans la partie TODO n°3 4.3.2 présent de le fichier game.c. Et donc pour correctement libérer la mémoire des structures, il nous a fallu passer par un transfert de pointeur à un autre et donc par la déclaration de pointeur. Ceci explique les lignes 8, 9 et 10. Le reste de la structure permet la déclaration de pointeurs et de variables nécessaires au bon fonctionnement du programme, que nous utiliserons dans le fichier game.c.

3.1.3 Les déclarations de fonction

La suite du fichier game.h n'a pour bute que de déclarer les fonctions qui seront écrites dans le fichier game.c. Les fonction sont les suivantes :

- game_board_update() 4.3.2
- game_new() 4.3.3
- game_run() 4.3.5
- game_del() 4.3.4

3.2 tetris.h

```
1  #ifndef TETRIS_H
2  #define TETRIS_H
3  #include <SDL.h>
4  #include <SDL_ttf.h>
5  #include <time.h>
6  #include <string.h>
7  #include <math.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 typedef enum
12   {
13     TYPE_I,
14     TYPE_J,
15     TYPE_L,
16     TYPE_O,
17     TYPE_S,
18     TYPE_T,
19     TYPE_Z
20   } Type;
21
22 typedef struct Tetris
23 {
24   char matrix[20][10];
25   char buffer[20][10];
26
27   Type current_type;
28   int current_line;
29   int current_column;
30   int current_rotation;
31
32   Type next_type;
33
34   int level;
35   int score;
36   int nbr_lines;
37   float drop_speed[15]; /* in s*/
38 } Tetris;
```

```
39
40 /*créer la structure tétris pour une nouveau jeu */
41 Tetris *tetris_new();
42
43 /* libère les ressources */
44 void tetris_del(Tetris *tet);
45
46 /* met en place le nouveau tetrimino */
47 void tetris_reset(Tetris *tet);
48
49 /* teste des évènements du tetrimino */
50 int tetris_can_go_left(Tetris *tet);
51 int tetris_can_go_right(Tetris *tet);
52 int tetris_can_rotate_h(Tetris *tet);
53 int tetris_can_rotate_ah(Tetris *tet);
54 int tetris_can_go_down(Tetris *tet);
55
56 /*renvoie la vitesse de déscente du tétrimino */
57 double tetris_get_drop_speed(Tetris *tet);
58
59 /*Nombre aléatoire entre 0 et 7 */
60 int alea();
61
62 //Initialise un générateur d'aléatoire
63 void init_alea();
64
65 //Met le tetrimino dans la matrice
66 void tetris_matrix_update(Tetris *tet);
67
68 // Permet de mettre à jours le plateau de jeux avec les lines détruites en moin
69 void tetris_shift_board(Tetris *tet);
70
71 #endif
```

3.2.1 Les inclusions

Le fichier est inclue dans le fichier game.h 3.1 et dans le fichier tetris.c 4.2. Ici nous avons inclue différentes bibliothèques. Notamment la bibliothèque SDL pour l'utilisation de la SDL, la biliothèque "SDL_ttf" pour utiliser les fonctions de la TTF et "time.h" pour utiliser des fonctions permettant la génération de nombre aléatoire, utilisées dans la fonction "init_alea()" et la fonction "alea()" que nous verrons dans le fichier tetris.c 4.2.
Cependant nous avons remarqué que le programme réussit à compiler alors que les bibliothèques "string.h", "math.h", et "stdlib.h" ne sont pas présent dans le fichier d'en-tête. Ors des fonctions provenant de ces fichiers d'en-tête sont utilisées comme la fonction pow() du fichier "math.h", utilisée dans la fonction "tetris_get_drop_speed" dans le fichier tetris.c 4.2. Nous avons préféré tout de même les laisser pour éviter tout conflit, car le programme fonctionne avec comme sans.

3.2.2 L'énumération

De la ligne 11 à la ligne 20 nous avons un type d'énumération qui renvoie un int pour chaque type de tetrimino allans de 0 à 6 dans l'ordre de déclaration des types.

3.2.3 Le structure Tetris

Cette structure renvoie différents pointeurs.
Le premiers, ligne 24 pointe vers la matrice qui affichera les minos et les blocs de collision.
La deuxième, ligne 25 vers le buffer où est placé le tetrimino utilisable par le joueur.
De la ligne 27 à 30 nous avons les pointeurs vers les caractéristiques du tetrimino présent dans le buffer : son type, sa position en ligne et colonne (centré au milieu du tableau 5x5 qui le défini) et sa rotation.
Ensuite nous avons un pointeur vers le prochain type du tetrimino.
Pour finir nous avons les caractéristiques de la partie actuelle : le niveau, le score, le nombre de ligne détruite et la vitesse de descente du tetrimino.

3.2.4 Les fonctions déclarés

En plus des fonctions initialement déclarées dans le fichier tetris.h, il existait dans le fichier tetris.c d'autre fonction qu'il fallait également écrire pour faire fonctionner le programme, il a donc fallu les déclarer également dans le fichier tetris.h. Voici les fonctions déclarés, qui seront expliquées :

- tetris_new() 4.2.3
- tetris_del() 4.2.4
- tetris_reset() 4.2.5
- tetris_can_go_left() 4.2.6
- tetris_can_go_right() 4.2.7
- tetris_can_rotate_h() 4.2.8
- tetris_can_rotate_ah() 4.2.9
- tetris_can_go_down() 4.2.10
- tetris_get_drop_speed() 4.2.11
- alea() 4.2.2
- init_alea() 4.2.2
- tetris_matrix_update() 4.2.12
- tetris_shift_board() 4.2.13

Et les fonctions supplémentaires :

- tetris_get_drop_speed() ; 4.2.11
- alea() ; 4.2.2
- init_alea() ; 4.2.2
- tetris_matrix_update() ; 4.2.12
- tetris_shift_board() ; 4.2.13

3.3 mino.h

```
#ifndef MINO_H
#define MINO_H

//Définition de la strucutre Color afin d'affecter les couleurs à chaques pixels
typedef struct Color
{
  char r;
  char g;
  char b;
}Color;

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

#endif
```

Il y a trois points à noter dans ce fichier qui sont les suivants :

- Le fichier est inclue dans game.c et mino.c 4.1.
- Cette structure nous permet de pouvoir associer au pointeur char r,g et b des valeurs comprises entre 0 et 255 et à travers différentes fonctions convertir ces valeurs en couleur.
- La seul fonction à déclarer est la fonction "mino_display" 4.1.1 qui est programmée dans le fichier mino.c.

Chapitre 4

Les fichiers ".c"

4.1 mino.c

Ce fichier est très important pour l'affichage du jeu. Il permet au jours de pouvoir observer l'état du jeu et voir l'affichage de la matrice et du buffer. La fonction mino_display() est utilisée dans le fonction game_board_update() et la méthode de création du mino également. J'ai donc du compléter ce fichier avant de pouvoir travailler plus en profondeur sur la fonction game_board_update(). Le seul problème rencontré dans ce fichier à été de faire un effet de relief avec un assombrissement de la couleur lors de l'affichage des tetriminos.

```c
#include "game.h"
#include "mino.h"


void mino_display(Game *g, Type t, int l, int c)
{
  int i;
  int j;
  int z;

  //Conditionnement en fonction du type de tetrimino
  if(t==0)// I
    {
      Color *color_I;

      //Allocation de mémoire
      color_I = calloc(31,sizeof(Color));

      //Teste si la mémoire à bien été alloué.
      if(!color_I)
  return;

      //On remplit les pointeurs de couleur, du plus claire au plus sombre
      for(i=0;i<30;i++)// 255/30=8.5
  {
    //Cyan
    color_I[i].r=0;
    color_I[i].g=255-(i*8.5);
    color_I[i].b=255-(i*8.5);
  }
      //On dessine le mino
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_I[i].r,color_I[i].g,color_I[i].b,255);
    //Les ligne horizontal
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
      }
    //Les lignes verticale
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
      }

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

```
49  else if(t==1)//J
50    {
51      Color *color_J;
52      color_J = calloc(31,sizeof(Color));
53      if(!color_J)
54  return;
55
56      for(i=0;i<30;i++)// 255/30=8.5
57  {
58    //Bleu
59    color_J[i].r=0;
60    color_J[i].g=0;
61    color_J[i].b=255-i*8.5;
62  }
63      for(i=0;i<30;i++)
64  {
65    SDL_SetRenderDrawColor(g->ren,color_J[i].r,color_J[i].g,color_J[i].b,255);
66    for(j=i;j<30;j++)
67      {
68        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
69      }
70    for(z=i+1;z<30;z++)
71      {
72        SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
73      }
74
75  }
76      SDL_RenderPresent(g->ren);
77    }
78  else if(t==2)//L
79    {
80      Color *color_L;
81      color_L = calloc(31,sizeof(Color));
82      if(!color_L)
83  return;
84
85      for(i=0;i<30;i++)// 255/30=8.5
86  {
87    //Orange
88    color_L[i].r=255-i*8.5;
89    color_L[i].g=165.75-i*5.525;
90    color_L[i].b=0;
91  }
92
93      for(i=0;i<30;i++)
94  {
95    SDL_SetRenderDrawColor(g->ren,color_L[i].r,color_L[i].g,color_L[i].b,255);
96    for(j=i;j<30;j++)
97      {
98        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
99      }
100   for(z=i+1;z<30;z++)
101     {
102       SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
103     }
104
105 }
106     SDL_RenderPresent(g->ren);
107   }
108 else if(t==3)//O
109   {
110     Color *color_O;
111     color_O = calloc(31,sizeof(Color));
112     if(!color_O)
113 return;
114
115     for(i=0;i<30;i++)// 255/30=8.5
116 {
117   //Jaune
118   color_O[i].r=255-i*8.5;
119   color_O[i].g=255-i*8.5;
120   color_O[i].b=0;
121 }
122
123     for(i=0;i<30;i++)
124 {
```

```
    SDL_SetRenderDrawColor(g->ren,color_O[i].r,color_O[i].g,color_O[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
      }

  }
      SDL_RenderPresent(g->ren);
    }
  else if(t==4)//S
    {
      Color *color_S;
      color_S = calloc(31,sizeof(Color));
      if(!color_S)
  return;

      for(i=0;i<30;i++)// 255/30=8.5
  {
    //Vert
    color_S[i].r=0;
    color_S[i].g=127.5-i*4.25;
    color_S[i].b=0;
  }

      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_S[i].r,color_S[i].g,color_S[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
      }

  }
      SDL_RenderPresent(g->ren);
    }
  else if(t==5)//T
    {
      Color *color_T;
      color_T = calloc(31,sizeof(Color));
      if(!color_T)
  return;

      for(i=0;i<30;i++)// 255/30=8.5
  {
    //Violet
    color_T[i].r=237.15-i*7.905;
    color_T[i].g=130.05-i*4.335;
    color_T[i].b=237.15-i*7.905;
  }

      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_T[i].r,color_T[i].g,color_T[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
      }

  }
      SDL_RenderPresent(g->ren);
    }
  else if(t==6)//Z
    {
      Color *color_Z;
```

```
201       color_Z = calloc(31,sizeof(Color));
202       if(!color_Z)
203   return;
204 
205       for(i=0;i<30;i++)// 255/30=8.5
206   {
207     //Rouge
208     color_Z[i].r=255-i*8.5;
209     color_Z[i].g=0;
210     color_Z[i].b=0;
211   }
212 
213       for(i=0;i<30;i++)
214   {
215     SDL_SetRenderDrawColor(g->ren,color_Z[i].r,color_Z[i].g,color_Z[i].b,255);
216     for(j=i;j<30;j++)
217       {
218         SDL_RenderDrawPoint(g->ren,c*30+j,l*30+i);
219       }
220     for(z=i+1;z<30;z++)
221       {
222         SDL_RenderDrawPoint(g->ren,c*30+i,l*30+z);
223       }
224 
225   }
226       SDL_RenderPresent(g->ren);
227     }
228 }
```

Pour commencer il faut y inclure les fichiers d'en-tête game.h et mino.h pouvoir utiliser la structure Game et Color.

4.1.1 mino_display()

Elle ne contient qu'une seule fonction. La fonction mino_display(). C'est une fonction qui prend en argument un pointeur de type Game, une variable de type Type, et deux autres variables pour la ligne l et la colonne c, et elle revoit le dessin d'un mino dans le plateau de jeu.
Le code de cette fonction est répétitif et suit un schéma semblable d'un type de mino à l'autre, à la différence que le type sera différent et donc les couleurs renvoyées aussi. Nous rappelons que le les consignes pour le projet sont :

- Couleur jaune pour le tetrimino O.
- Couleur cyan pour le terimino I.
- Couleur violet pour le terimino T.
- Couleur orange pour le terimino L.
- Couleur bleu pour le terimino J.
- Couleur vert pour le terimino s.
- Couleur rouge pour le terimino Z.

En réalité le début de la fonction est le même que dans la fonction game_board_update() dans la partie TODO n°2 4.3.2 d'écrite plus bas. La différence sera dans la boucle for dans la quelle cette foi-ci, nous allons dessiner au point $x = c \times 30 + j$ et $y = l \times 30 + i$.

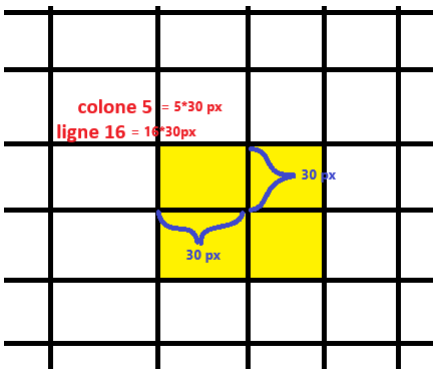

FIGURE 4.1 – Schéma représentatif d'un mino dans le plateau

Le schéma ci-dessus prend en exemple le mino à la ligne 16 et à la conne 5.
On comprend d'après le schéma que le premier pixel de la ligne l est égale à 30xl, respectivement pour la colonne c nous avons 30xc. Donc nous avons la coordonnée supérieure gauche du mino dans la matrice et le buffer en ligne l et colonne c. Cependant il faudra dessiner autant de point qu'il y à de pixel dans le mino et le mino est de dimension 30x30. C'est pour cela donc que nous ajoutons "j" à la coordonné colonne et "i" à la coordonnée de ligne afin de parcourir tous les pixels du mino.

La description plus détaillée sur l'affectation, l'utilisation et le calcule de la couleur, et la fonctionnement des boucles for se trouve au label suivant : 4.3.2

4.2 tetris.c

Le fichier tetris.c est le fichier de lequel il y a le plus de fonction. Certaines étaient déjà données par l'énoncé du projet, d'autre n'étaient pas directement visible, car elles n'étaient pas données directement comme consigne, mais elles étaient utilisées dans le programme et certaines devaient être imaginées pour faire fonctionner le programme correctement. Nous pensons que ce fichier était le plus accessible, car il ne nécessitait pas l'apprentissage d'un concept nouveau contrairement au fichier game.c 4.3, qui nécessitait de comprendre la bibliothèque SDL. De plus ce fichier permet de suivre les "états" du jeux et d'utiliser ces "états", qui changerons en fonction du temps et des actions du joueur. Ces "états" sont les différents pointeurs de la structure Tetris 3.2, qui sont beaucoup utilisés.
le seul fichier inclue dans tetris.c est tetris.h, car tout le nécessaire y est :

- La structure Tetris.
- Les types de tetrimino enuméré par Type.
- Les bibliothèques nécessaires.
- La déclaration des fonctions.

```
1 #include "tetris.h"
```

4.2.1 Initialisation du tableau tetrimino et du tableau de vitesse

Pour travailler sur notre jeux il était essentielle de pouvoir représenter les différents tetrimino en fonction de type (O, I, J, L, S, Z, T) et de leurs rotations (rotation 0, rotation 1, rotation 2, rotation 3). Pour cela il a fallu initialiser " char tetriminos[7][4][5][5] ". Nous avons eu du mal à écrire ce tableau. A l'origine nous voulions le faire à l'aide de boucle for, et nous voulions le remplir par "rien" pour les cases vides et "mino" pour les cases remplies, ors bien évidemment, cela ne peut pas fonctionner car "rien" et "mino" ne sont pas des char mais des chaînes de caractères. Cependant nous avons pensé que le problème venait de la boucle for, car nous avions lu sur un forum qu'il fallait décrire un tableau lors de son initialisation, cependant cela n'était pas le cas pour les boucles for. Nous avons donc tenté de remplir le tableau directement, mais encore une fois l'erreur persistait. C'est alors quand lisant plus précisément l'erreur et en taper l'erreur sur internet, nous avons fini par comprendre que le problème venait du contenu du tableau et non de sa construction. Nous avons donc gardé la construction lors de l'initialisation, et cette fois le contenu est devenu '1' pour les cases remplies et '0' pour les cases vide. Pour la construction du tableau multidimensionnel nous nous somme aider du site zestedesavoir.
Le code de construction est le suivant :

```
char tetriminos[7][4][5][5]={
  {/*Tetrimino I*/
    {/*Rotation 0 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','1','1','1','1'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 1*/
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'}
    },
    {/*Rotation 2*/
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'1','1','1','1','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 3*/
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    }
  },
  {/*Tetrimino J*/
    {/*Rotation 0 */
      {'0','0','0','0','0'},
      {'0','1','0','0','0'},
      {'0','1','1','1','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 1*/
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 2*/
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','1','1','1','0'},
      {'0','0','0','1','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 3*/
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','1','1','0','0'},
      {'0','0','0','0','0'}
    }
  },
  {/*Tetrimino L*/
    {/*Rotation 0*/
      {'0','0','0','0','0'},
      {'0','0','0','1','0'},
      {'0','1','1','1','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 1 */
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
```

```
    {/*rotation 2 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','1','1','1','0'},
      {'0','1','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/* Rotation 3 */
      {'0','0','0','0','0'},
      {'0','1','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    }
  },
  {/*Tetrimino 0 */
    {/* Rotation 0 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 1 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 2 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 3 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
  },
  {/* Tetrimino S */
    {/* Rotation 0*/
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','1','1','0','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/* Rotation 1*/
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','1','0'},
      {'0','0','0','1','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 2*/
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','0','1','1','0'},
      {'0','1','1','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 3 */
      {'0','0','0','0','0'},
      {'0','1','0','0','0'},
      {'0','1','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    }
  },
  {/*Tetrimino T */
```

```
    {/* Rotation 0 */
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','1','1','1','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/* Rotation 1 */
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','0','1','1','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 2 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','1','1','1','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 3 */
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','1','1','0','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    }
  },
  {/* Tetrimino Z */
    {/* Rotation 0 */
      {'0','0','0','0','0'},
      {'0','1','1','0','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'},
      {'0','0','0','0','0'}
    },
    {/*Rotation 1 */
      {'0','0','0','0','0'},
      {'0','0','0','1','0'},
      {'0','0','1','1','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    },
    {/* Rotation 2 */
      {'0','0','0','0','0'},
      {'0','0','0','0','0'},
      {'0','1','1','0','0'},
      {'0','0','1','1','0'},
      {'0','0','0','0','0'}
    },
    {/* Rotation 3 */
      {'0','0','0','0','0'},
      {'0','0','1','0','0'},
      {'0','1','1','0','0'},
      {'0','1','0','0','0'},
      {'0','0','0','0','0'}
    }
  }
};
```

Pour correctement construire le tableau il fallait suivre l'ordre des types du tetrimino imposé par Type, l'ordre est donc le suivant : I, J, L, O, S, T, Z. Ensuite il fallait respecter l'ordre de rotation horaire de 90°, on a donc : rotation 0 = 0°, rotation 1 = 90°, rotation 2 = 180°, rotation 3 = 270°. Les deux dernières valeurs du tableau sont les emplacements des minos pour chaque tétrimino. Pour les emplacements vides nous avons décidé de les remplir par '0', et '1' pour les emplacements contenant un mino. Nous savions qu'il était possible d'utiliser la valeur 1 et 0 mais non avons préférer rester sur des char. Le contenu est donc le "dessin" en '1' et '0' de chaque tetrimino et de chaque rotation dans un tableau 5x5.

Par exemple le tetrimino Z à la rotation 1 :

```
    ...
    ...
    },
    {
      {'0','0','0','0','0'},
      {'0','0','0','1','0'},
      {'0','0','1','1','0'},
      {'0','0','1','0','0'},
      {'0','0','0','0','0'}
    },
    {...
    ...
    ...
```

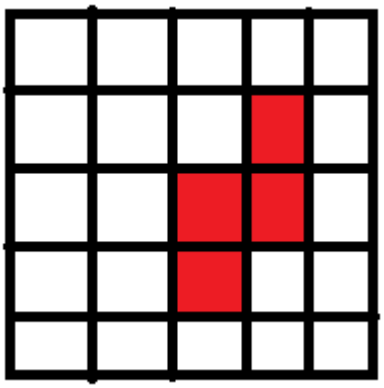

FIGURE 4.2 – Schéma représentatif du tetrimino Z en rotation 1

Pour le tableau de vitesse nous avons préféré le remplir directement par les valeurs approchées de chaque vitesse en fonction du niveau afin de faciliter les calcules du programme, cela permettra de rendre le jeu plus efficace. Le code est le suivant :

```
float tab2
    [15]={1.0,0.793,0.618,0.473,0.355,0.262,0.190,0.135,0.094,0.064,0.043,0.028,0.0018,0.011,0.007};
```

Nous aurions pu aussi faire le code suivant :

```
float tab2[15];

for(i=1;i<=15;i++)
  {
    tab2[i]=pow(0.8-((i-1)*0.007),i-1);
  }
```

Mais comme mentionné plus haut, pour des raisons techniques, nous ne l'avons pas fait.

4.2.2 Les fonctions aléatoires

L'utilisation de fonctions aléatoires était nécessaire pour le bon fonctionnement du jeu, car pour choisir le premier tétrimino du jeu et le tétrimino suivant il fallait une fonction pour le faire. Plus précisément une fonction qui choisit un nombre entre 0 et 6, valeur égale au type de tetrimino associé. Pour l'implémentation des ces fonctions nous nous sommes aidés d'un forum sur openclassrooms et du site developpez pour mieux comprendre leurs utilisations et comment les coder. La fonction aléatoire à eu deux formes, la première était celle-ci :

```
int alea()
{
//Initialisation d'un générateur d'aléatoire
  srand(time(NULL));
  return rand()%7;;
}
```

Nous avons donc initialisé un générateur de nombre aléatoire pour que la fonction renvoie un nombre aléatoire entre 0 et 6.
A la ligne 4 la fonction srand() initialise un générateur de nombre aléatoire basé sur l'heure.

Ligne 5 on retourne un nombre aléatoire compris entre 0 et 7 grâce au modulo 7.

Cette fonction fonctionnait très bien initialement, mais nous pouvions constater lors de l'utilisation du jeu que certains tetriminos se répétaient. Cela était dû au générateur aléatoire qui, étant basé sur l'heure, si la seconde entre deux tétriminos était la même, les tétriminos seraient les mêmes, rendant le jeu plus répétitif. Nous avons donc décidé de changer le code pour celui-ci :

```
1  //Initialisation d'un générateur d'aléatoire
2  void init_alea()
3  {
4    srand(time(NULL));
5  }
6
7  //Renvoie un nombre aléatoire entre 0 et 6.
8  int alea()
9  {
10   return rand()%7;;
11 }
```

Ainsi un unique générateur aléatoire est initialisé au début du jeu avec init_alea() et alea() renverra une valeur "réellement" aléatoire.

4.2.3 tetris_new()

Cette fonction n'est utilisée qu'une seule fois dans la fonction game_new() 4.3.3, car elle permet d'initialiser un nouveau setup Tetris, elle ne sera donc pas utilisée ailleurs.
Le code est le suivant :

```
1  Tetris *tetris_new()
2  {
3    Tetris *t;
4    int i;
5    int j;
6
7    //Allocation dans la mémoire
8    t=malloc(sizeof(Tetris));
9    if(t==NULL)
10     return NULL;
11
12   //Remplissage du buffer et de la matrice de '0'.
13   for (i=0;i<20;i++)
14     {
15       for(j=0;j<10;j++)
16   {
17     t->matrix[i][j]='0';
18     t->buffer[i][j]='0';
19   }
20     }
21
22   //Choix d'un premier type aléatoire
23   t->current_type=alea();
24
25   //Positionnement sur la premiere ligne en fonction du type
26   if(t->current_type==0||t->current_type==3)
27     {
28       t->current_line=0;
29     }
30   else
31     {
32       t->current_line=1;
33     }
34
35   //Posistionnement sur le colonne 4 pour centrer le tetrimino
36   t->current_column=4;
37
38
39   t->current_rotation=0;
40
41   t->next_type=alea();
42
43   t->level=1;
44   t->score=0;
45   t->nbr_lines=0;
46
47   for(i=0;i<15;i++)
```

```
48      {
49        //Remplir le tableau par des valeurs prédéfinies est plus intéressant que de faire les
        calcules pour des questions de vitesse.
50        t->drop_speed[i]=tab2[i];
51      }
52    return t;
53  }
```

Premièrement il faut initialiser les éléments nécessaires. Le pointeur t de type Tetris et les entiers i et j pour faire des boucles for.

Ensuite de la ligne 8 à 9 on alloue de la mémoire et on teste si l'allocation c'est bien passée.

Ensuite de la ligne 13 à 20 on utilise une double boucle for pour remplir les pointeurs du buffer et de la matrice de t de '0' élément considérer nul. Cela permet de bien avoir un nouveau jeu à la ligne 23.

Puis nous utilisons la fonction alea() pour choisir un type pour le premiers tetrimino du jeu.

Pour le pointeur de ligne et de colonne actuelle de t, la valeur de la colonne sera 4 et le ligne dépendra du pointeur de tetrimino actuel. Car si le tetrimino est O ou I, en rotation 0 il y a assez de place pour les mettre sur la ligne 0 du buffer, sinon la ligne sera 1. Cela a été défini par le tableau tetrimino 4.2.1. Le pointeur de rotation actuel de t est 0 et le pointeur du prochain tétrimino de t est une valeur encore aléatoire entre 0 et 6.

On pointe ensuite le score à 0, le niveau à 1 et le nombre de ligne détruite à 0 et on remplit le pointeur drop_speed de t par les valeurs du tableau de vitesse 4.2.1.
Enfin on retourne t.

4.2.4 tetris_del()

```
1  void tetris_del(Tetris *tet)
2  {
3    free(tet);
4  }
```

La fonction est très courte et nécessite qu'une seul ligne, car il n'y a qu'une seule allocation de mémoire nécessaire lors de la création d'un nouveau Tetris. Donc utiliser la fonction free() pour libérer la mémoire allouée par tet est suffisant.

4.2.5 tetris_reset()

Cette fonction à pour but de relancer un nouveau tetrimino une fois que le tetrimino actuel ne peut plus descendre. Elle prend en argument un pointer appelé tet de type Tetris.

```
1  void tetris_reset(Tetris *tet)
2  {
3    int i;
4    int j;
5
6    tet->current_type=tet->next_type;
7    tet->next_type=alea();
8    tet->current_column=4;
9    tet->current_rotation=0;
10
11   //Remise du buffer à '0'
12   for (i=0;i<20;i++)
13     {
14       for(j=0;j<10;j++)
15   {
16     tet->buffer[i][j]='0';
17   }
18     }
19
20   //Positionnement du tetrimino dans le buffer en fonction de son type
21   if(tet->current_type==0||tet->current_type==3)
22     {
23       tet->current_line=0;
24       for(i=2;i<=4;i++)
25   {
26     for(j=0;j<=4;j++)
27       {
```

```
28          tet->buffer[i-2][j+2]=tetriminos[tet->current_type][tet->current_rotation][i][j];
29        }
30    }
31      }
32    else
33      {
34        tet->current_line=1;
35        for(i=1;i<=4;i++)
36    {
37      for(j=0;j<=4;j++)
38        {
39          tet->buffer[i-1][j+2]=tetriminos[tet->current_type][tet->current_rotation][i][j];
40        }
41    }
42      }
43  }
```

Le code commence donc par initialiser deux entiers pour faire des doubles boucles for (ligne 3 et 4)

Par le suite nous devons implémenter le nouveau tetrimino, pour cela nous commençons par faire pointer le type actuel vers le pointeur du prochain type de tet. Cela à pour effet de faire passer le prochain type en type actuel (ligne 6).

Après il faut définir un nouveau type pour le pointeur du prochain type avec la fonction alea() (ligne 7).

Il faudra aussi remettre le pointeur à la colonne 4 et la rotation à 0 pour centrer le tetrimino dans sa position initiale (ligne 8 et 9).

Puis on remet le pointeur du buffer à '0' avec une double boucle for (ligne 12 à 18)

Enfin, il faut mettre le nouveau tetrimino dans le buffer à la ligne correspondante en fonction du type.
Donc si le type actuel du tetrimino est O ou I (0 ou 3) alors la ligne actuelle sera 0 sinon la ligne sera 1. Dans ces deux conditions on y ajoute deux boucles for la première est le suivante :

```
1  for(i=2;i<=4;i++)
2    {
3      for(j=0;j<=4;j++)
4        {
5          tet->buffer[i-2][j+2]=tetriminos[tet->current_type][tet->current_rotation][i][j];
6        }
7    }
```

Ici nous sommes dans le cas où le tetrimino est de type O ou I. Nous commençons par une boucle for allant de 2 à 4 car les deux premières lignes du tetrimino sont vides, puis nous poursuivons une autre boucle for qui parcourt l'ensemble des colonnes. Ensuite dans ces deux boucles for on effectue la commande suivante :

tet->buffer[i-2][j+2]=tetriminos[tet->current_type][tet->current_rotation][i][j] ;

Cela aura pour effet de placer le tetrimino dans la partie supérieur centré du buffer. Illustrons un exemple avec un dessin. On illustre avec le schéma que les valeurs i-2 et j+2, que nous avons utilisées permettent de faire

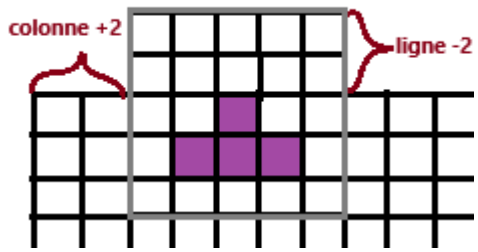

FIGURE 4.3 – Schéma de tetris_reset

correspondre la colonne et la ligne du tetrimino à sa place dans le buffer. Nous ajoutons 2 pour la colonne soustrayons 2 pour la ligne.

Le travail est le même pour les autres types de tetrimino à la différence que nous faisons une première boucle pour les lignes allant de 1 à 4, car uniquement la première ligne sera vide.

4.2.6 teris_can_go_left()

Elle prend en argument un pointeur de type Tetris et renvoie 1 si le tetrimino peut aller à gauche et 0 sinon. Cette fonction est utilisée pour vérifier si l'évènement "aller à gauche" est possible 4.3.5. Cette fonction ainsi que les autres fonctions tetris_can étaient au début difficile à comprendre car il fallait trouver une méthode pour que les minos associés au tetrimino soient reconnus au bonne endroit dans le buffer, afin de faire les testes. Cependant une fois la méthode trouvée pour une fonction, il ne fallait plus que la répéter pour le reste. Il s'agit aussi des fonctions pour lesquelles nous avons rencontré le plus de bug, notamment beaucoup de dépassement de limite. Ces bug étaient principalement dû à des coquilles dans le code.

```
int tetris_can_go_left(Tetris *tet)
{
  int i;
  int j;

  for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    //On cherche les case remplies par tetrimino
    if(tetriminos[tet->current_type][tet->current_rotation][i][j] == '1')
      {
        //ici on teste si la colonne à gauche du mino est toujours dans le tableau et on teste
     si la case à gauche du mino est remplie ou non
        //le calcule "current_column + j - 2" nous permet d'avoir la position du mino
        if(tet->current_column + j-3 <0 || tet->matrix[tet->current_line + i -2][tet->
    current_column + j-3] == '1')
    return 0;
      }
  }
    }
  return 1;
}
```

Les commentaires expliquent quelque peu le code cependant penchons nous sur la raison de cette méthode.
Nous commençons par faire une double boucle for avec i et j allant de 0 à 5 pour parcourir tout le tetrimino et tester si une case du tetrimino est égale à '1', donc pour vérifier si il y à un mino à cet endroit du tableau. Une fois l'emplacement localisé il faut refaire un teste pour vérifier si la case de la matrice située à gauche du mino est vide ou non (dans la matrice).
Pour cela il faut accéder à la valeur de la ligne actuelle et la colonne actuelle dans le buffer du mino correspondant. On utilise alors les valeurs suivantes :

$$\text{tet->current\_column} + j - 2$$

$$\text{tet->current\_line} + i - 2$$

Cette méthode fonctionne quelque soit le tetrimino. D'après le dessin si nous faisons tous les calcules en fonction

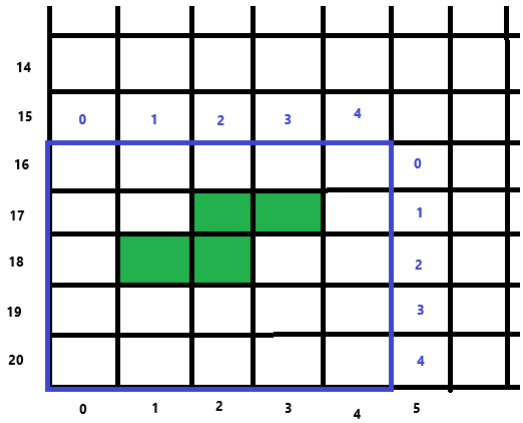

FIGURE 4.4 – Schéma d'un tetrimino dans le buffer

de la ligne et de la colonne dans le tableau tetrimino, qu'on soustrait par 2 nous obtenons les bons résultats.

Inutile de tester pour les colonnes. Pour les lignes nous avons bien :

$$\text{tet->current\_line} + 0 - 2 = 18 + 0 - 2 = 16$$
$$\text{tet->current\_line} + 1 - 2 = 18 + 1 - 2 = 17$$
$$\text{tet->current\_line} + 2 - 2 = 18 + 2 - 2 = 18$$
$$\text{tet->current\_line} + 3 - 2 = 18 + 3 - 2 = 19$$
$$\text{tet->current\_line} + 4 - 2 = 18 + 4 - 2 = 20$$

Une fois qu'on a accédé à la position dans le buffer on vérifie si dans la position à la même ligne, une colonne à gauche il y à un mino ou si nous nous trouvons à l'extérieur du plateau. Pour cela on fait un teste sur tet->current_column + j - 2 - 1, dans la matrice et dans les valeurs du tableau.
Donc si la cases à gauche du mino est remplie dans la matrice ou que l'emplacement à gauche du mino est un emplacement en dehors du terrain (emplacement inférieur à 0), alors on retourne 0, sinon on retourne 1.
Il est intéressant de préciser qu'il n'y aura pas de conflit si le mino sélectionné dans le buffer est un mino positionné à droite d'un autre mino dans le buffer, car la comparaison se fait entre le buffer et la matrice, ce qui implique que les minos sélectionnés dans le buffer ne sont pas dans la matrice.

4.2.7 tetris_can_go_right()

Elle prend en argument un pointeur de type Tetris et renvoie 1 si le tetrimino peut aller à droite et 0 sinon. Elle est utilisée pour vérifier si l'évènement, pour le déplacement à droite, est possible 4.3.5. La fonction teris_can_go_right() est quasiment identique à la fonction tetris_can_go_left(), à la différence que nous ne travaillons plus sur le colonne à gauche du mino, mais sur la colonne à droite. Voici le code :

```
int tetris_can_go_right(Tetris *tet)
{
  int i;
  int j;

  for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    if(tetriminos[tet->current_type][tet->current_rotation][i][j] == '1')
      {
        if((tet->current_column + j - 1 >=10) || (tet->matrix[tet->current_line +i -2][tet->
    current_column + j -1] == '1'))
    return 0;
      }
  }
    }
  return 1;
}
```

Donc la différence entre tetris_can_go_right() et tetris_can_go_left() est dans la dernière condition. Cette fois nous voulons vérifier si la case dans la colonne de droite est remplie ou non et si la colonne de droite est toujours dans le plateau du jeux. Donc nous vérifions de la même façons que dans tetris_can_go_left() cependant nous le vérifions à la colonne tet->current_column + j - 2 + 1. De plus pour être sûr que l'emplacement est bien dans le terrain de jeu il faut que cette valeur soit cette fois inférieur strictement à 10.

4.2.8 tetris_can_rotate_h()

Elle prend en argument un pointeur de type Tetris et renvoie 1 si le tetrimino peut tourner dans le sens horaire et 0 sinon. La fonction est utilisée pour vérifier si nous pouvons actionner l'évènement rotation horaire 4.3.5. Cette fonction ressemble également au précédente et suit un même principe qui est de vérifier si les tetriminos dans la matrice gène l'évènement.

```
for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    //Teste spécifique pour revenir à la rotation 0
    if(tet->current_rotation + 1 == 4)
      {
        if(tetriminos[tet->current_type][0][i][j]=='1')
    {
      //On teste si des minos bloquent la rotation, ou si le tetrimino dépasse les limites du
    jeux.
```

```
11        if(tet->matrix[tet->current_line +i -2][tet->current_column + j -2]=='1' ||tet->
      current_column +j-2 < 0 || tet->current_column+j-2>=10 || tet->current_line +i-2 >=20 ||
      tet->current_line +i-2 <0 )
12            return 0;
13      }
14        }
15
16
17      else if(tetriminos[tet->current_type][tet->current_rotation+1][i][j]=='1')
18        {
19          if(tet->matrix[tet->current_line +i -2][tet->current_column + j -2]=='1' ||tet->
      current_column +j-2 < 0 || tet->current_column+j-2>=10 || tet->current_line +i-2 >=20 ||
      tet->current_line +i-2 <0 )
20      return 0;
21        }
22    }
23      }
24    return 1;
25  }
```

Pour commencer nous faisons donc une double boucle for, puis il faut faire un teste sur le pointeur de rotation actuelle pour éviter un bug. Si nous ne faisons pas ce teste nous obtenons ce bug : Donc à l'issue de ce teste pour

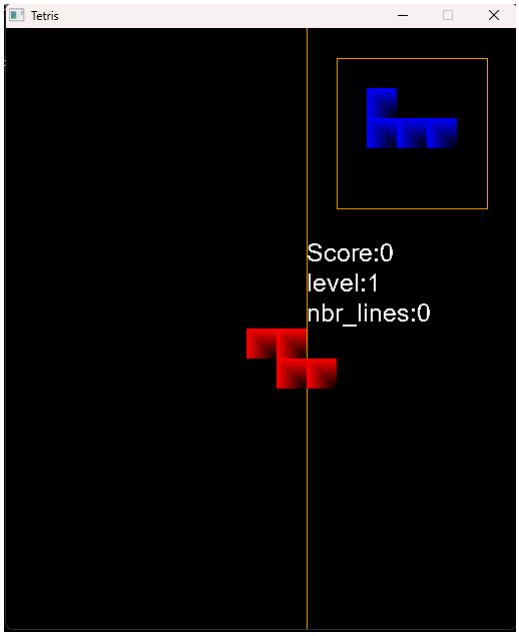

FIGURE 4.5 – Bug du Tetris

rester dans les rotations possibles du tetrimino, nous vérifions si les emplacements dans la matrice de la rotation suivante du tetrimino sont libres ou non et également que le tetrimino reste dans les bordures du terrain. Pour cela nous commençons par identifier les lignes et les colonnes où seront potentiellement placés les minos puis on fait le teste suivant :

tet->matrix[tet->current_line + i -2][tet->current_column + j -2]== '1 ' || tet-> current_column +j -2 < 0 || tet->current_column +j -2 >=10 || tet->current_line +i -2 >=20 || tet->current_line +i -2 <0

Si le teste est vrai on retourne 0 sinon on retour 1. Dans le cas ou l'incrémentation de la rotation est égale à 4 on remplace tet->current_rotation par 0 pour que le programme fonctionne correctement.

4.2.9 tetris_can_rotate_ah

Elle prend en argument un pointeur de type Tetris et renvoie 1 si le tetrimino peut tourner dans le sens anti-horaire et 0 sinon. Elle est utilisée pour effectuer l'évènement de rotation anti-horaire 4.3.5. Elle est quasiment la même que la fonction précédente hormis que dans celle-ci le premier teste sur la rotation actuel est en fonction de la décrémentation de celui-ci, car si la rotation actuelle -1 est égale à -1, cela veut dire que nous avons effectué une rotation complète donc la rotation actuelle doit être égale à 3. Le code est le suivant :

```
1  int tetris_can_rotate_ah(Tetris *tet)
2  {
```

```
  int i;
  int j;

  for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    if(tet->current_rotation - 1 == -1)
      {
        if(tetriminos[tet->current_type][3][i][j]=='1')
    {
      if(tet->matrix[tet->current_line +i -2][tet->current_column + j -2]=='1' ||tet->
    current_column +j-2 < 0 || tet->current_column+j-2>=10 || tet->current_line +i-2 >=20||tet
    ->current_line +i-2 <0 )
        return 0;
    }
      }

    if(tetriminos[tet->current_type][tet->current_rotation-1][i][j]=='1')
      {
        if(tet->matrix[tet->current_line +i -2][tet->current_column + j -2]=='1' ||tet->
    current_column +j-2 < 0 || tet->current_column+j-2>=10 || tet->current_line +i-2 >=20||tet
    ->current_line +i-2 <0 )
    return 0;
      }
  }
    }
  return 1;
}
```

Donc cette fois-ci nous travaillons sur la rotation précédente et non la rotation suivante, d'où les testes pour tet->current_rotation-1.

4.2.10 tetris_can_go_down()

Elle prend en argument un pointeur de type Tetris et renvoie 1 si le tetrimino peut descendre et 0 sinon. Elle est utilisée pour l'évènement "descendre une case" 4.3.5 et l'évènement "descendre tout en bas" 4.3.5 et dans la fonction game_run 4.3.5. Elle est quasiment identique à la fonction tetris_can_go_right(), sauf que cette fois-ci il sera question de vérifier la ligne d'en-dessous et de vérifier si la ligne suivant est inférieur à 20. Voici le code :

```
int tetris_can_go_down(Tetris *tet)
{
  int i;
  int j;

  for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    if(tetriminos[tet->current_type][tet->current_rotation][i][j]=='1')
      {
        //On teste si un mino est en dessous du tetrimino ou si le tetrimino dépasse le
    terrain.
        if((tet->current_line + i -1 >= 20)  || (tet->matrix[tet->current_line +i -1][tet->
    current_column + j -2]=='1'))
    return 0;
      }
  }
    }
  return 1;
}
```

Comme les précédentes fonction il sera question d'identifier les cases du tetrimino qui sont remplies avec une double boucle for et un teste en fonction du pointeur de type actuel, du pointeur de rotation actuel et la colonne j et la ligne i en question. Ensuite pour vérifier si le tetrimino peut descendre on utilise la valeur de ligne suivante :

$$\text{tet->current\_line} + \text{i} - 2 + 1$$

Avec cela, comme dans les précédentes fonctions on vérifie la contenance de la case en-dessous et on vérifie si l'indice de la ligne du dessous est supérieur à 20. Si le tetrimino dépasse le terrain ou si il y a un mino en-dessous alors on retourne 0, sinon 1.

4.2.11 tetris_get_drop_speed()

Cette fonction permet d'obtenir la vitesse actuelle du jeux en fonction du niveau, elle est utilisé dans le fonction game_run(). Son code est le suivant :

```
double tetris_get_drop_speed(Tetris *tet)
{
  return pow(0.8-((tet->level -1)*0.007),tet->level -1);
}
```

Elle prend donc en argument un pointeur de type Tetris et renvoie la vitesse en fonction du pointeur sur le niveau. On utilise aussi la fonction pow() provenant de la bibliothèque math.h.

4.2.12 tetris_matrix_update()

Elle fait partie d'une des fonctions que nous avons dû imaginer pour que le programme fonction correctement, car initialement nous voulions mettre à jour directement la matrice dans la fonction game_run(), mais cela causait des problèmes de compilation si nous déclarions le tableau tetrimino dans tetris.h et si nous l'incluons dans game.c. Donc il était plus judicieux de programmer une fonction dans tetric.c.

```
void tetris_matrix_update(Tetris *tet)
{
  int i;
  int j;
  for(i=0;i<5;i++)
    {
      for(j=0;j<5;j++)
  {
    if(tetriminos[tet->current_type][tet->current_rotation][i][j]=='1')
      {
        tet->matrix[tet->current_line+i-2][tet->current_column+j-2]=tetriminos[tet->
    current_type][tet->current_rotation][i][j];
      }
  }
    }
}
```

Le programme est assez simple, nous parcourons le tableau tetrimino avec une double boucle for de 0 à 5, pour identifier les cases remplies, une fois cela fait nous les affectons aux pointeurs de la matrice correspondants en fonction de la colonne j et de la ligne i.

4.2.13 tetris_shift_board()

Cette fonction prend en argument un pointeur de type Tetris et à pour effet de faire descendre tous les minos une fois qu'une ou plusieurs destructions de ligne ait été effectuées. Le code est le suivant :

```
void tetris_shift_board(Tetris *tet)
{
  int i;
  int j;
  int k;
  int count;

for(i=0;i<20;i++)
  {
    count=0;
    for(j=0;j<10;j++)
  {
    if(tet->matrix[i][j]=='1')
      {
        count+=1;
      }
  }
    //Teste si une ligne est complis
      if(count == 10)
  {
    //Remplir la ligne de '0'
    for(j=0;j<10;j++)
      {
        tet->matrix[i][j]='0';
      }
    //Parcourt le tableau de bas en haut
    for(k=i;k>0;k--)
      {
```

```
29          //Colonne par colonne
30          for(j=0;j<10;j++)
31      {
32        //Affecte la ligne du dessus à la ligne du dessous
33        tet->matrix[k][j]=tet->matrix[k-1][j];
34      }
35        }
36      //Necessaire dans le cas ou la premiers ligne est remplie de mino.
37      //Car la ligne 0 n'est pas affécté à la ligne 1;
38      for(j=0;j<10;j++)
39        {
40          tet->matrix[0][j]='0';
41        }
42    }
43      }
44  }
```

Pour le bon fonctionnement du code il nous faut 3 entiers pour les boucles et une variable de contage que nous initialisons.
Initialement nous ne voulions pas parcourir le tableau dans sont intégralité mais plutôt se restreindre au ligne dans les quelles le tetrimino se situe et par la suite faire un travail sur le nombre de ligne à éliminées et leurs emplacement, etc. Cependant c'est une méthode compliqué une fois arriver à la descente des minos. Nous avons alors préféré opter pour un parcours complet du tableau qui facilitera la descente des minos.

Donc nous commençons par une boucle for sur i allant de 0 à 19 pour parcourir toutes les lignes. Nous affectons ensuite à notre compteur la valeur de 0, puis de la ligne 11 à la ligne 16 nous effectuons notre comptage pour vérifier si la ligne est remplie. Il suffit d'incrémenter de 1 count et si à la sortie de la boucle, count est égal à 10 alors la ligne est remplie.
Donc si cette condition est vérifiée on commence d'abord par remplir la ligne de '0' (ligne 22 à 25).

Puis toujours en respectant la condition de comptage, nous effectuons une double boucle for, cette fois pour faire descendre les minos. La première boucle sera alors une boucle qui remontera les lignes à partir de la ligne remplie, donc k est affectée à i, et tant que k est supérieure à 0, nous décrémentons k. Il faut préciser que nous devons nous arrêter à 1 pour éviter de sortir du tableau. La seconde boucle va parcourir toutes les colonnes et la ligne 33 aura pour effet de faire descendre le mino au-dessus de la ligne k.

Pour finir nous faisons une dernière boucle for dans la première boucle for i, afin de remplir la première ligne de '0', au cas où si la ligne contenait des minos.

4.3 game.c

Nous allons maintenant voir le contenue du fichier game.c, en présentant chaque fonction de ce fichier. Comme déjà mentionné le site zestedesavoir a été essentielle pour la réussite du programme. Grâce au contenue du site, nous avons pu comprendre comment fonctionnent les différentes commandes de la SDL et en apprendre de nouvelles pour faciliter le codage, comme la création de ligne et de rectangle. Également, le travaille de Cam dans la première et deuxième vidéo sur l'utilisation de TTF_font nous a permis d'obtenir des résultats important sur l'affichage.
Cela nous à donc permit de résoudre les problèmes du projet avec ces outils. Parfois sans succès, notamment dans la fonction mino_display() 4.1.1 ou dans le TODO n°2 4.3.2, mentionné plus bas, où nous avons dû rester sur des méthodes plus accessibles. Mais nous pensons que le point le plus important était sur l'utilisation des commandes permettant de créer et de fermer des fenêtres ou des rendus, car si cette partie du travaille n'était pas correctement effectuée, il n'y aurait pas pu y avoir d'affichage.

4.3.1 Inclusion de fichier d'en-tête et déclaration de variable global

```
1 #include "game.h"
2 #include "mino.h"
3
4 static int update =1;
```

Nous incluons game.h pour pouvoir utiliser les bibliothèques nécessaires et la structure Game. Nous initialisons également update à 1 comme variable global car elle est utilisé dans deux fonctions du fichier.

4.3.2 game_board_update()

Cette fonction prend en argument un pointeur de type Game et elle renvoie un affichage dans une fenêtre. Cette fonction a été l'une des plus longue à coder, nous n'avons pas trouvé d'autre moyens que de répéter plusieurs fois des lignes de codes très similaires. Cependant l'étude de l'utilisation de la bibliothèque SDL à aussi été d'une très grande aide pour l'écriture de cette fonction. Pour présenter la fonction nous allons la découper en quatre parties.

- L'initialisation et la clôture de la fonction
- TODO n°1 : afficher le plateau de jeu 4.3.2
- TODO n°2 : afficher le prochain tetrimino 4.3.2
- TODO n°3 : afficher le score, le niveau et le nombre de lignes eliminées 4.3.2

L'initialisation et la clôture de la fonction

```
1  static void game_board_update(Game *g)
2  {
3    int i;
4    int j;
5    int z;
6
7    SDL_SetRenderDrawColor(g->ren, 0x2b, 0x2a, 0x33, 0xff);
8    SDL_RenderClear(g->ren);
9    ...
10   ...
11   ...
12   //Suite du programme
13   ...
14   ...
15   ...
16   SDL_RenderPresent(g->ren);
17
18   update = 0;
19 }
```

Nous initialisons les variables entières i, j et z car nous allons les utiliser dans des boucles for.
Les lignes 13 permettent d'afficher le rendu du du pointeur de g sur le rendu à l'écran.
Dans la partie suite du programme nous aurons les TODO.

TODO n°1 : afficher le plateau de jeu

```
1  // Colorer le fond du plateau en noir
2    SDL_Color black = {0,0,0,255};
3    SDL_SetRenderDrawColor(g->ren,black.r,black.g,black.b,black.a);
4    SDL_RenderClear(g->ren);
5
6    /*Bordure du plateau de jeux*/
7    SDL_Color orange = {255,165.75,0,255};
8    SDL_SetRenderDrawColor(g->ren,orange.r,orange.g,orange.b,orange.a);
9    SDL_RenderDrawLine(g->ren,300,600,300,0);
10
11   //Dessiner un rectangle en haut à droite
12   SDL_SetRenderDrawColor(g->ren,orange.r,orange.g,orange.b,orange.a);
13   SDL_Rect r={330,30,151,151};
14   SDL_RenderDrawRect(g->ren,&r);
15
16   //Affichage des minos dans le plateau de jeux pour avoir un visuel de la matrix.
17   for(i=0;i<20;i++)
18     {
19       for(j=0;j<10;j++)
20   {
21     if(g->tet->matrix[i][j]=='1')
22       {
23         mino_display(g,g->tet->current_type,i,j);
24       }
25   }
26     }
27
28   //Affichage du tetrimino actuelle, qui est  présent dans le buffer.
29   if(g->tet->current_type==0)
```

```
    {//Tetrimino I
      if(g->tet->current_rotation==0)
  {//Rotation 0
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+2);
  }
      else if(g->tet->current_rotation==1)
  {//Rotation 1
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+2,g->tet->current_column);
  }
      else if(g->tet->current_rotation==2)
  {//Rotation 2
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-2);
  }
      else if(g->tet->current_rotation==3)
  {//Rotation 3
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line-2,g->tet->current_column);
  }
    }
  else if(g->tet->current_type==1)
    {//Tetrimino J
      if(g->tet->current_rotation==0)
  {//Rotation 0
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
  }
      else if(g->tet->current_rotation==1)
  {//Rotation 1
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column+1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
  }
      else if(g->tet->current_rotation==2)
  {//Rotation 2
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column+1);
  }
      else if(g->tet->current_rotation==3)
  {//Rotation 3
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
  }
    }
  else if(g->tet->current_type==2)
    {//Tetrimino L
      if (g->tet->current_rotation==0)
  {//Rotation 0
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column+1);
  }
      else if (g->tet->current_rotation==1)
  {//Rotation 1
    mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
    mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column+1);
```

```
106   }
107       else if (g->tet->current_rotation==2)
108   {//Rotation 2
109     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
110     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
111     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
112     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column-1);
113   }
114       else if (g->tet->current_rotation==3)
115   {//Rotation 3
116     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
117     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
118     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
119     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column-1);
120   }
121     }
122   else if(g->tet->current_type==3)
123     {//Tetrimino O, pour toutes les rotations
124       mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
125       mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
126       mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
127       mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column+1);
128     }
129   else if(g->tet->current_type==4)
130     {//Tetrimino S
131       if(g->tet->current_rotation==0)
132   {//Rotation 0
133     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
134     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
135     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
136     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column+1);
137   }
138       else if(g->tet->current_rotation==1)
139   {//Rotation 1
140     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
141     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
142     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
143     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column+1);
144   }
145       else if(g->tet->current_rotation==2)
146   {//Rotation 2
147     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
148     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
149     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
150     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column-1);
151   }
152       else if(g->tet->current_rotation==3)
153   {//Rotation 3
154     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
155     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
156     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
157     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column-1);
158   }
159     }
160   else if(g->tet->current_type==5)
161     {//Tetrimino T
162       if(g->tet->current_rotation==0)
163   {//Rotation 0
164     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
165     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
166     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
167     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
168   }
169       else if(g->tet->current_rotation==1)
170   {//Rotation 1
171     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
172     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
173     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
174     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
175   }
176       else if(g->tet->current_rotation==2)
177   {//Rotation 2
178     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
179     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
180     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
181     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
```

```
182   }
183       else if(g->tet->current_rotation==3)
184   {//Rotation 3
185     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
186     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
187     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
188     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
189   }
190     }
191   else if(g->tet->current_type==6)
192     {//Tetrimino Z
193       if(g->tet->current_rotation==0)
194   {//Rotation 0
195     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column-1);
196     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
197     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
198     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
199   }
200       else if (g->tet->current_rotation==1)
201   {//Rotation 1
202     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column+1);
203     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
204     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
205     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
206   }
207       else if (g->tet->current_rotation==2)
208   {//Rotation 2
209     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column+1);
210     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column);
211     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
212     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
213   }
214       else if (g->tet->current_rotation==3)
215   {//Rotation 3
216     mino_display(g,g->tet->current_type,g->tet->current_line-1,g->tet->current_column);
217     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
218     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
219     mino_display(g,g->tet->current_type,g->tet->current_line+1,g->tet->current_column-1);
220   }
221     }
```

Les commentaires présents dans le code expliquent l'effet et l'utilité des groupes de lignes. Nous allons voir plus en détaille le fonctionnement des fonctions.

A la ligne 2 nous utilisons SDL_Color, qui est un pointeur vers une structure à 4 variables. Une pointe vers la couleur rouge (r), une autre vers la couleur verte (g), une pour la couleur bleu (b) et une pour l'opacité de la couleur (a). Ces trois variables prennent des valeurs comprises entre 0 et 255. Nous avons donc une déclaration de la forme :

$$\text{SDL\_Color color} = \{r,g,b,a\};$$

. Cette affectation nous permet d'avoir facile la couleur noir et la couleur orange (ligne 7) avec la SDL.
Ensuite ligne 3 nous utilisons la commande

$$\text{SDL\_SetRenderDrawColor(render,color.r,color.g,color.b,color.a)};$$

Cette commande permet d'affecter à un "renderer" traduit par rendu, la couleur obtenue avec les valeurs color.r, color.g, color.b, color.a. Cela permettra d'obtenir des résultats colorés en effectuant des commandes sur le rendu. Dans notre cas "renderer" est "g->ren" qui est un pointeur vers le rendu de la variable g, initialement généré par le fonction game_new() 4.3.3. La compilation ne posera donc pas de problème car dans le fichier main.c 2, game_new() est appelé en premiers et game_board_update() est utilisé dans la fonction game_run() 4.3.5.
Une fois le rendu associé à une couleur nous pouvons effectuer des commandes sur celui-ci pour générer des résultats visuels. Par exemple les commandes

$$\text{SDL\_RendererClear(renderer)};$$

$$\text{SDL\_RenderDrawLine(renderer,}x_1\text{,}y_1\text{,}x_2\text{,}y_2\text{)};$$

La première permet d'appliquer la couleur à l'ensemble de la fenêtre et la seconde de dessiner une ligne de la coordonnée x_1, y_1 à la coordonnée x_2, y_2.
Ensuite (ligne 13 nous avons initialisé un rect avec la commande

$$\text{SDL\_Rect rect x,y,w,h}$$

où x et y représentent les coordonnée supérieur gauche du rectangle, w et h représentent respectivement la largeur et la hauteur du rectangle. Cela nous permet de dessiner ensuite un rectangle (non rempli) avec la commande

SDL_RenderDrawRect(renderer,&rect) ;

Dans notre cas les coordonnés du coin supérieur gauche du rectangle sont 330 et 30 car nous sommes partis d'une fenêtre de taille 600x510, donc 600/20=30 (20 est égale au nombre de ligne du jeu), ce qui implique que la taille d'un bloc de notre fenêtre est égale à 30 pixel. Par contage on obtient donc ces coordonnées. Cette

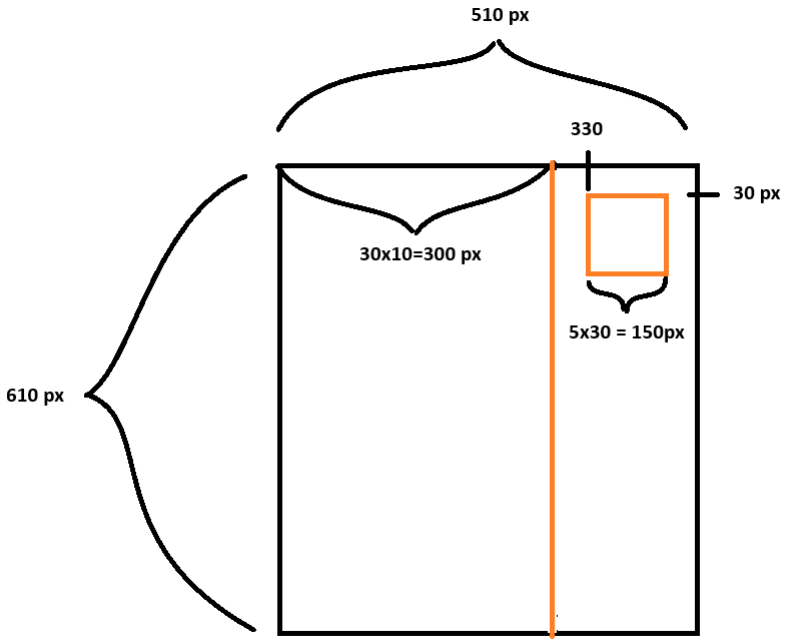

FIGURE 4.6 – Schéma représentatif du plateau

méthode de visualisation du plateau sera utilisée tout au long du programme dans les cas où il faudra compter en pixel des coordonnées.

De la ligne 17 à la ligne 26 nous avons une double boucle for.

```
for(i=0;i<20;i++)
    {
      for(j=0;j<10;j++)
  {
    if(g->tet->matrix[i][j]=='1')
      {
        mino_display(g,g->tet->current_type,i,j);
      }
  }
    }
```

Elle permet de parcourir la matrice et d'afficher un mino coloré la où la case de la matrice est remplie. Nous utilisons donc une condition pour détecter si la case pointer par g->tet->matrix[i][j] est remplie. Si c'est le cas alors nous dessinons un mino à cet emplacement avec le fonction

mino_display(g,g->tet->current_type,i,j) ; 4.1

qui dessinera le mino en fonction du type du mino présent dans le buffer. Cela aura pour effet de transformer tous les minos de la matrice en une couleur unique. Nous n'avons pas trouvé d'autre solution.

La suite du programme est une répétition de la fonction mino_display de façons conditionnée afin d'afficher le tetrimino présent dans le buffer. Il y aura donc une condition pour chaque types en fonction du pointeur g->tet->current_type et dans ces conditions, des conditions pour chaque rotation en fonction du pointeur g->tet->current_rotation.
Prenons pour exemple le tetrimino I à la rotation 0 :

```
1 if(g->tet->current_type==0)
2     {//Tetrimino I
3       if(g->tet->current_rotation==0)
4   {//Rotation 0
5     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column-1);
6     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column);
7     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+1);
8     mino_display(g,g->tet->current_type,g->tet->current_line,g->tet->current_column+2);
9   }
```

Ligne 1 nous avons la première condition. g->tet->current_type peut être égale à 0, 1, 2, 3, 4, 5, ou 6 qui correspond aux valeurs du type associé au tetrimino que nous avons décrit dans le fichier tetris.c 4.2.1. Les types associés respectent l'énumération présente dans le fichier tetris.h 3.2.

La ligne 3 décrit la rotation du tetrimino ou g->tet->current_rotation peut être égale à 0, 1, 2 ou 3, d'écrites également dans le fichier tetris.c.

Enfin nous avons quatre utilisations de la fonction mino_display 4.1.1, un par mino et tous les tetriminos sont constitués de quatre minos. La fonction prendra donc en paramètre ligne et colonne l'emplacement du mino dans le buffer. Pour se faire il faudra passer par une conversion de g->tet->current_line et g->tet->current_column. Pour cela il a fallu passer par un dessin (nous rappelons que la ligne et la colonne est centré sur le centre du tableau des tetrimino).

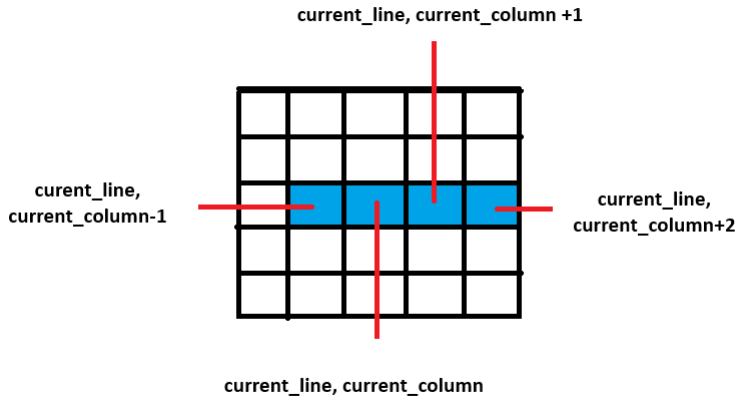

FIGURE 4.7 – Schéma représentatif du tetrimino dans la matrice

Nous réitérerons le processus pour tous les types et pour toutes les rotations et nous obtenons un affichage pour chaque tetrimino.

TODO n°2 : afficher le prochain tetrimino

```
if(g->tet->next_type==0)
    {//I
      //Déclaration d'un pointeur de type Color qui point vers les valeur rouge, vert et bleu d'une couleur.
      Color *color_I;

      //Allocation de mémoire pour la variable couleur.
      color_I = calloc(31,sizeof(Color));

      //Teste si l'allocation c'est bien passé.
      if(!color_I)
  return;

      //Affectation de la couleur à la variable. Pour l'obtenir on calcule le pourcentage de 255 et on soustrait par i*8.5 pour avoir 30 couleur différente. 30 pour le nombre de pixel.
      for(i=0;i<30;i++)// 255/30=8.5
  {
    color_I[i].r=0;
    color_I[i].g=255-(i*8.5);
    color_I[i].b=255-(i*8.5);
  }

      //Affichage de chaque mino, en haut à droite, en fonction de leur position en pixel.
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_I[i].r,color_I[i].g,color_I[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,360+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,420+j,90+i);
        SDL_RenderDrawPoint(g->ren,450+j,90+i);

      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,360+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,420+i,90+z);
        SDL_RenderDrawPoint(g->ren,450+i,90+z);
      }

  }
    }
  else if(g->tet->next_type==1)
    {
      Color *color_J;
      color_J = calloc(31,sizeof(Color));
      if(!color_J)
  return;

      for(i=0;i<30;i++)
  {
    color_J[i].r=0;
    color_J[i].g=0;
    color_J[i].b=255-i*8.5;
  }
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_J[i].r,color_J[i].g,color_J[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,360+j,60+i);
        SDL_RenderDrawPoint(g->ren,360+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,420+j,90+i);

      }
    for(z=i+1;z<30;z++)
```

```
      {
        SDL_RenderDrawPoint(g->ren,360+i,60+z);
        SDL_RenderDrawPoint(g->ren,360+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,420+i,90+z);
      }

  }
    }
  else if(g->tet->next_type==2)
    {
      Color *color_L;
      color_L = calloc(31,sizeof(Color));
      if(!color_L)
  return;

      for(i=0;i<30;i++)
  {
    color_L[i].r=255-i*8.5;
    color_L[i].g=165.75-i*5.525;
    color_L[i].b=0;
  }
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_L[i].r,color_L[i].g,color_L[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,360+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,420+j,90+i);
        SDL_RenderDrawPoint(g->ren,420+j,60+i);

      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,360+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,420+i,90+z);
        SDL_RenderDrawPoint(g->ren,420+i,60+z);
      }

  }
    }
  else if(g->tet->next_type==3)
    {
      Color *color_O;
      color_O = calloc(31,sizeof(Color));
      if(!color_O)
  return;

      for(i=0;i<30;i++)
  {
    color_O[i].r=255-i*8.5;
    color_O[i].g=255-i*8.5;
    color_O[i].b=0;
  }
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_O[i].r,color_O[i].g,color_O[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,420+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,120+i);
        SDL_RenderDrawPoint(g->ren,420+j,120+i);

      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,420+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,120+z);
        SDL_RenderDrawPoint(g->ren,420+i,120+z);
      }
  }

```

```
    }
  else if(g->tet->next_type==4)
    {
      Color *color_S;
      color_S = calloc(31,sizeof(Color));
      if(!color_S)
  return;

      for(i=0;i<30;i++)
  {
    color_S[i].r=0;
    color_S[i].g=127.5-i*4.25;
    color_S[i].b=0;
  }
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_S[i].r,color_S[i].g,color_S[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,360+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,60+i);
        SDL_RenderDrawPoint(g->ren,420+j,60+i);

      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,360+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,60+z);
        SDL_RenderDrawPoint(g->ren,420+i,60+z);
      }
  }
    }
  else if(g->tet->next_type==5)
    {
      Color *color_T;
      color_T = calloc(31,sizeof(Color));
      if(!color_T)
  return;

      for(i=0;i<30;i++)
  {
    color_T[i].r=237.15-i*7.905;
    color_T[i].g=130.05-i*4.335;
    color_T[i].b=237.15-i*7.905;
  }
      for(i=0;i<30;i++)
  {
    SDL_SetRenderDrawColor(g->ren,color_T[i].r,color_T[i].g,color_T[i].b,255);
    for(j=i;j<30;j++)
      {
        SDL_RenderDrawPoint(g->ren,360+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,90+i);
        SDL_RenderDrawPoint(g->ren,390+j,60+i);
        SDL_RenderDrawPoint(g->ren,420+j,90+i);

      }
    for(z=i+1;z<30;z++)
      {
        SDL_RenderDrawPoint(g->ren,360+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,90+z);
        SDL_RenderDrawPoint(g->ren,390+i,60+z);
        SDL_RenderDrawPoint(g->ren,420+i,90+z);
      }
  }

    }
  else if(g->tet->next_type==6)
    {
      Color *color_Z;
      color_Z = calloc(31,sizeof(Color));
      if(!color_Z)
  return;

      for(i=0;i<30;i++)
```

```
220   {
221     color_Z[i].r=255-i*8.5;
222     color_Z[i].g=0;
223     color_Z[i].b=0;
224   }
225       for(i=0;i<30;i++)
226   {
227     SDL_SetRenderDrawColor(g->ren,color_Z[i].r,color_Z[i].g,color_Z[i].b,255);
228     for(j=i;j<30;j++)
229       {
230         SDL_RenderDrawPoint(g->ren,360+j,60+i);
231         SDL_RenderDrawPoint(g->ren,390+j,60+i);
232         SDL_RenderDrawPoint(g->ren,390+j,90+i);
233         SDL_RenderDrawPoint(g->ren,420+j,90+i);
234
235       }
236     for(z=i+1;z<30;z++)
237       {
238         SDL_RenderDrawPoint(g->ren,360+i,60+z);
239         SDL_RenderDrawPoint(g->ren,390+i,60+z);
240         SDL_RenderDrawPoint(g->ren,390+i,90+z);
241         SDL_RenderDrawPoint(g->ren,420+i,90+z);
242       }
243   }
244     }
```

Dans cette partie du code nous avons également une répétition de condition pour chaque types de tetrimino, nous allons en prendre l'exemple du tetrimino J et le commenter.

```
1  else if(g->tet->next_type==1)
2      {
3        Color *color_J;
4        color_J = calloc(31,sizeof(Color));
5        if(!color_J)
6    return;
7
8        for(i=0;i<30;i++)
9    {
10     color_J[i].r=0;
11     color_J[i].g=0;
12     color_J[i].b=255-i*8.5;
13   }
14       for(i=0;i<30;i++)
15   {
16     SDL_SetRenderDrawColor(g->ren,color_J[i].r,color_J[i].g,color_J[i].b,255);
17     for(j=i;j<30;j++)
18       {
19         SDL_RenderDrawPoint(g->ren,360+j,60+i);
20         SDL_RenderDrawPoint(g->ren,360+j,90+i);
21         SDL_RenderDrawPoint(g->ren,390+j,90+i);
22         SDL_RenderDrawPoint(g->ren,420+j,90+i);
23
24       }
25     for(z=i+1;z<30;z++)
26       {
27         SDL_RenderDrawPoint(g->ren,360+i,60+z);
28         SDL_RenderDrawPoint(g->ren,360+i,90+z);
29         SDL_RenderDrawPoint(g->ren,390+i,90+z);
30         SDL_RenderDrawPoint(g->ren,420+i,90+z);
31       }
32
33   }
```

Nous pouvons remarquer que le code a des similitudes avec le code de mino_display() 4.1.1.
Ligne 1, la condition en fonction du type.

De la ligne 3 à la ligne 6, nous initialisons un pointeur de type Color pour le tetrimino J 3.3, ensuite nous allouons une quantité de mémoire égale à 31 x sizeof(Color) pour stocker dans Color_J 30 valeurs d'une couleur allant du plus claire au plus sombre, une par ligne de pixel. Nous effectuons également un teste pour vérifier que l'allocation a correctement fonctionné.

De la ligne 8 à la ligne 13, nous stockons dans color_J nos 30 couleurs. Le tableau aura une couleur allant du plus claire au plus sombre.
Il a fallu effectuer quelques calcules pour avoir le bon résultat. Le premiers est de diviser 255 par 30 ce qui donne 8.5, ainsi nous aurons 30 valeurs pour 30 intensités.
Le deuxième nécessite un calcule de pourcentage. A l'aide du liens wikipédia présent sur le pdf du projet, nous avons calculé pour chaque couleurs du tétrimino la valeur en pourcentage de l'indice de couleur 255. Dans le cas du tetrimino J la couleur étant le bleu nous avons comme rouge = 0%, vert = 0% et bleu = 100% et 100% de 255 est égale à 255, mais pour des cas plus compliqués, il faut utiliser la formule suivante :

$$\frac{\text{veleur} \times \text{pourcentage}}{100} \tag{4.1}$$

Nous avons donc utilisé cette formule pour chaque couleurs.

Ensuite nous avons une boucle for principale dans laquelle il y a deux boucles for différentes.
La boucle for principale permet de parcourir les 30 intensité de color_J et d'avoir 30 itération, une par ligne de pixel.
le seconde boucle for permet de dessiner pixel par pixel les lignes en commencent sur la diagonal du mino, allant de l'intensité la plus claire à la plus sombre. Pour cela nous avons utilisé la commande :

SDL_RenderDarwPoint(renderer,x,y) ;

qui dessine une pixel aux coordonnées x et y et il a fallu affecter à j la valeur i pour bien commencer à la valeur sur la diagonal de notre mino. Par la suite l'incrémentation de j par la boucle for, nous permettra de bien parcourir tous les pixels de la ligne i.

Une chose à préciser, lors de la conception du code, nous voulions à l'origine dessiner ligne par ligne le mino, mais le code semblait plus compliqué, voir potentiellement infaisable. Nous avons donc préféré garder la méthode des pixels afin de ne pas perdre trop de temps si le codage n'était pas possible.

Dans notre cas il fallait identifier les coordonnées x et y pour. Pour cela nous sommes passés par un dessin. La figure ci-dessus est une zoome sur le premiers schéma 4.2

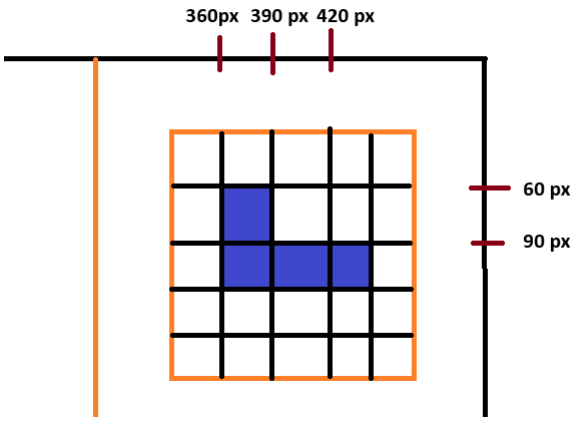

FIGURE 4.8 – Schéma du tetrimino suivant

Nous avons donc utilisé cette méthode d'identification pour chaque tetrimino.
Enfin la dernière boucle permet de dessiner les colonnes du mino en commencent à la ligne en dessous de la diagonal. Donc cette fois ci ce sera les coordonnées des colonnes qu'il faudra incrémenter de 1, et pour bien respecter le premier pixel situé en-dessous de la diagonal, nous affectons la valeur i+1 à z.
Finalement pour obtenir le dessins d'un tetrimino il faut répéter 4 fois le dessin des pixels pour chaque coins supérieurs gauches du mino, en tenant compte des positions des minos, en fonction du type de tetrimino.

TODO n°3 : afficher le score, le niveau et le nombre de lignes eliminées

Dans cette section nous nous sommes beaucoup aider du tutoriel de Cam sur youtube dans la partie 1 et la partie 2 sur l'utilisation de TTF_font pour programmer l'affichage du score, du niveau et du nombre de ligne.

```
1  //Déclaration nécessaire:
2   //Variable de la donnée
3   int score = g->tet->score;
4   //Tableau contenant la donnée
5   char buffer_s[20]={'\0'};
6
7   int level = g->tet->level;
8   char buffer_l[2]={'\0'};
9
10  int nbr_lines = g->tet->nbr_lines;
11  char buffer_nl[10]={'\0'};
12
13  //Tableau contenant la donnée et un texte
14  char final_buffer_s[26]={'\0'};
15  char final_buffer_l[8]={'\0'};
16  char final_buffer_nl[20]={'\0'};
17
18  // Ici on déclare des surfaces qui permetrons d'écrire le texte,
19  // à l'aide de la fonction TTF_RenderText_Solid().
20  SDL_Surface * text_s= NULL;
21  SDL_Surface * text_l= NULL;
22  SDL_Surface * text_nl= NULL;
23
24  //Ici nous déclarons des texture dans lesquelles seront copiées les surfaces.
25  SDL_Texture * texture_s = NULL;
26  SDL_Texture * texture_l = NULL;
27  SDL_Texture * texture_nl = NULL;
28
29  //Il s'agit ici de réctangle représentant la surface dans laquelle il y aura le texte.
30  SDL_Rect rect_s = {300,210,210,30};
31  SDL_Rect rect_l = {300,240,210,30};
32  SDL_Rect rect_nl = {300,270,210,30};
33
34  // On une couleur
35  SDL_Color white = {255,255,255,255};
36
```

```
37
38  // Ici itoa transforme une valeur numérique en une chaine de carractère et on la mets dans
      le tableau.
39  itoa(score,buffer_s,10);
40  itoa(level,buffer_l,10);
41  itoa(nbr_lines,buffer_nl,10);
42
43  //Avec strcpy on copie "score:" dans le buffer final.
44  strcpy(final_buffer_s,"Score:");
45  //Avec strcat on concatène les tableau de carractère.
46  strcat(final_buffer_s,buffer_s);
47
48  strcpy(final_buffer_l,"level:");
49  strcat(final_buffer_l,buffer_l);
50
51  strcpy(final_buffer_nl,"nbr_lines:");
52  strcat(final_buffer_nl,buffer_nl);
53
54
55  //Ici on "met" dans la surface les chaine de caractère et on teste l'erreur.
56  text_s = TTF_RenderText_Solid(g->font, final_buffer_s,white);
57  if(!text_s)
58    {
59      //En cas d'erreur on libert toutes les allocation de mémoire.
60      SDL_FreeSurface(text_s);
61      TTF_CloseFont(g->font);
62      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
63      SDL_DestroyRenderer(g->ren);
64      SDL_DestroyWindow(g->win);
65      TTF_Quit();
66      SDL_Quit();
67      return;
68    }
69
70  text_l = TTF_RenderText_Solid(g->font, final_buffer_l,white);
71  if(!text_l)
72    {
73      SDL_FreeSurface(text_l);
74      TTF_CloseFont(g->font);
75      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
76      SDL_DestroyRenderer(g->ren);
77      SDL_DestroyWindow(g->win);
78      TTF_Quit();
79      SDL_Quit();
80      return;
81    }
82
83  text_nl = TTF_RenderText_Solid(g->font, final_buffer_nl,white);
84  if(!text_nl)
85    {
86      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
87      SDL_FreeSurface(text_nl);
88      TTF_CloseFont(g->font);
89      SDL_DestroyRenderer(g->ren);
90      SDL_DestroyWindow(g->win);
91      TTF_Quit();
92      SDL_Quit();
93      return;
94    }
95
96  //Ensuite on créer des texture à partir des surface, car la texture nous permetra d'avoir l'
      affichage souhaité.
97  texture_s = SDL_CreateTextureFromSurface(g->ren,text_s);
98  SDL_FreeSurface(text_s);
99  //On teste l'erreur.
100 if((!texture_s) || (SDL_QueryTexture(texture_s,NULL,NULL,&rect_s.w,&rect_s.h) !=0) )
101   {
102     SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
103     TTF_CloseFont(g->font);
104     SDL_DestroyTexture(texture_s);
105     SDL_DestroyRenderer(g->ren);
106     SDL_DestroyWindow(g->win);
107     TTF_Quit();
108     SDL_Quit();
109     return;
110   }
```

```
111
112   texture_l = SDL_CreateTextureFromSurface(g->ren,text_l);
113   SDL_FreeSurface(text_l);
114   if((!texture_l) || (SDL_QueryTexture(texture_l,NULL,NULL,&rect_l.w,&rect_l.h) !=0))
115     {
116       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
117       TTF_CloseFont(g->font);
118       SDL_DestroyTexture(texture_l);
119       SDL_DestroyRenderer(g->ren);
120       SDL_DestroyWindow(g->win);
121       TTF_Quit();
122       SDL_Quit();
123       return;
124     }
125
126   texture_nl = SDL_CreateTextureFromSurface(g->ren,text_nl);
127   SDL_FreeSurface(text_nl);
128   if((!texture_nl) || (SDL_QueryTexture(texture_nl,NULL,NULL,&rect_nl.w,&rect_nl.h) !=0))
129     {
130       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
131       TTF_CloseFont(g->font);
132       SDL_DestroyTexture(texture_nl);
133       SDL_DestroyRenderer(g->ren);
134       SDL_DestroyWindow(g->win);
135       TTF_Quit();
136       SDL_Quit();
137       return;
138     }
139
140   //On va afficher le texte en associant la texture au renderer pour avoir un affichage.
141   if (SDL_RenderCopy(g->ren, texture_s,NULL,&rect_s) != 0)
142     {
143       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
144       TTF_CloseFont(g->font);
145       SDL_DestroyTexture(texture_s);
146       SDL_DestroyTexture(texture_l);
147       SDL_DestroyTexture(texture_nl);
148       SDL_DestroyRenderer(g->ren);
149       SDL_DestroyWindow(g->win);
150       TTF_Quit();
151       SDL_Quit();
152       return;
153     }
154
155   if (SDL_RenderCopy(g->ren, texture_l,NULL,&rect_l) != 0)
156     {
157       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
158       TTF_CloseFont(g->font);
159       SDL_DestroyTexture(texture_s);
160       SDL_DestroyTexture(texture_l);
161       SDL_DestroyTexture(texture_nl);
162       SDL_DestroyRenderer(g->ren);
163       SDL_DestroyWindow(g->win);
164       TTF_Quit();
165       SDL_Quit();
166       return;
167     }
168
169   if (SDL_RenderCopy(g->ren, texture_nl,NULL,&rect_nl) != 0)
170     {
171       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
172       TTF_CloseFont(g->font);
173       SDL_DestroyTexture(texture_s);
174       SDL_DestroyTexture(texture_l);
175       SDL_DestroyTexture(texture_nl);
176       SDL_DestroyRenderer(g->ren);
177       SDL_DestroyWindow(g->win);
178       TTF_Quit();
179       SDL_Quit();
180       return;
181     }
182
183   //On affecte aux pointeur de texture les texture d'affichage.
184   g->tex_s=texture_s;
185   g->tex_l=texture_l;
186   g->tex_nl=texture_nl;
```

```
187
188   //On détruit les texture d'affichage inutile.
189   SDL_DestroyTexture(texture_s);
190   SDL_DestroyTexture(texture_l);
191   SDL_DestroyTexture(texture_nl);
192
193   //Affichage du résultat.
194   SDL_RenderPresent(g->ren);
195
196   update = 0;
197
198 }
```

Dans ce programme, nous effectuons 3 fois le même méthode d'affichage. Pour la décrire, nous allons donc d'abord présenter les déclarations qui sont différentes, puis décrire la création de l'affichage du score qui sera différente des autre par ces coordonnées de placement et son contenue.

```
1   //Variable de la donnée
2   int score = g->tet->score;
3   //Tableau contenant la donnée
4   char buffer_s[20]={'\0'};
5
6   int level = g->tet->level;
7   char buffer_l[2]={'\0'};
8
9   int nbr_lines = g->tet->nbr_lines;
10  char buffer_nl[10]={'\0'};
11
12  //Tableau contenant la donnée et un texte
13  char final_buffer_s[26]={'\0'};
14  char final_buffer_l[8]={'\0'};
15  char final_buffer_nl[20]={'\0'};
```

Dans ce début de partie du programme nous initialisons des variabales contenant les valeurs de score, nbr_line et level afin de faciliter le codage. Ensuite nous créons des tableaux nommés "buffer" pour contenir les valeurs respectivement associées. Ensuite nous créons également des tableaux nommés "final_buffer" qui contiendront à la fois les datas associées et le texte d'affichage. Le nombre de valeur de ces tableaux est défini par le nombre de char qu'il contiennent. Pour le buffer de score et de nbr_line l'affichage est limité.

Passons à l'affichage de score.

```
1  int score = g->tet->score;
2  char buffer_s[20]={'\0'};
3  char final_buffer_s[26]={'\0'};
4  SDL_Surface * text_s= NULL;
5  SDL_Texture * texture_s = NULL;
6  SDL_Rect rect_s = {300,210,210,30};
7  SDL_Color white = {255,255,255,255};
8
9  itoa(score,buffer_s,10);
10 strcpy(final_buffer_s,"Score:");
11 strcat(final_buffer_s,buffer_s);
12
13 text_s = TTF_RenderText_Solid(g->font, final_buffer_s,white);
14   if(!text_s)
15     {
16       SDL_FreeSurface(text_s);
17       TTF_CloseFont(g->font);
18       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
19       SDL_DestroyRenderer(g->ren);
20       SDL_DestroyWindow(g->win);
21       TTF_Quit();
22       SDL_Quit();
23       return;
24     }
25
26 texture_s = SDL_CreateTextureFromSurface(g->ren,text_s);
27   SDL_FreeSurface(text_s);
28   if((!texture_s) || (SDL_QueryTexture(texture_s,NULL,NULL,&rect_s.w,&rect_s.h) !=0) )
29     {
30       SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
31       TTF_CloseFont(g->font);
32       SDL_DestroyTexture(texture_s);
33       SDL_DestroyRenderer(g->ren);
34       SDL_DestroyWindow(g->win);
35       TTF_Quit();
```

```
36        SDL_Quit();
37        return;
38      }
39
40  if (SDL_RenderCopy(g->ren, texture_s,NULL,&rect_s) != 0)
41      {
42        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
43        TTF_CloseFont(g->font);
44        SDL_DestroyTexture(texture_s);
45        SDL_DestroyTexture(texture_l);
46        SDL_DestroyTexture(texture_nl);
47        SDL_DestroyRenderer(g->ren);
48        SDL_DestroyWindow(g->win);
49        TTF_Quit();
50        SDL_Quit();
51        return;
52      }
53
54  g->tex_s=texture_s;
55
56  SDL_DestroyTexture(texture_s);
57
58  }
```

Premièrement nous devons initialiser une surface et une texture à NULL afin de pouvoir les utiliser plus tard.
Ensuite nous avons besoin d'initialiser un rectangle pour placer l'affichage à l'endroit voulu, qui est situé deux cases en dessous du rectangle orange du schéma 3 4.3. Donc en coordonnée 300 et 210 de taille 210x30 pour recouvrir la ligne.
Enfin nous initialisons également la couleur du texte qui sera blanc.

Ensuite des lignes 9 à 11 nous créons un tableau affichant "Score :[le score]".
Pour cela nous utilisons la fonction itoa qui permet de convertir des chiffres en lignes de caractères et de stocker le résultat dans un espace de mémoire de type char. Dans notre cas la valeur numérique est "score", la mémoire est "buffer_s" et 10 représente la base numérique, qui est décimal dans notre cas.
Par la suite nous utilisons strcpy qui copie une chaîne de caractère dans un espace de mémoire de type char et donc nous copions "Score :" dans le buffer final.
Et nous ajoutons buffer_s dans final_buffer_s avec strcat.

De la ligne 13 à 24, nous "écrivons" dans la surface le texte en allouant de la mémoire pour cela on utilise la fonction

TTF_RenderText_Solid(TTF_font *font,char *text, SDL_Color color);

On pourra avec ça écrire dans la surface text_s le texte présent dans final_buffer_s, en blanc avec la police pointée par g->font qui est "arial".
Nous testons également si l'allocation de mémoire à bien été effectuée, si l'allocation échoue il faut libérer toutes les allocations de mémoire précédemment allouées, renvoyer une erreur, quitter le SDL et TTF à l'aide des commande respective.

Ensuite ligne 26 nous créons une texture_s depuis une surface. Cela nous permettra de manipuler plus facilement le texte à afficher. Nous utilisons alors la commande :

SDL_CreateTextureFromSurface(g->ren,text_s);

On libère parla suite la mémoire allouée par la surface. Également, si l'allocation de mémoire échoue, il faut agir en conséquence.
Nous allons également charger le texte en mémoire avec la deuxième partie de la condition, ce qui est nécessaire pour son affichage. On utilise pour cela

SDL_QueryTexture(texture_s,NULL,NULL,&rect_s.w,& rect_s.h)

Puis nous affichons le texte avec un if et la commande

SDL_RenderCopy(g->ren,texture_s,NULL,&rect_s);

Ici aussi si le teste est faut il faut bien libérer la mémoire et quitter à la fois la SDL et TTF.

Enfin on affecte au pointeur g->tex_s la texture_s pour pouvoirs libérer la mémoire de la texture_s et une fois le jeu fermé, libérer la mémoire de g->tex_s grâce à la fonction game_del() 4.3.4.

4.3.3 game_new()

La fonction game_new() prend en argument deux int un pour la coordonnée x et l'autre pour la coordonnée y du coin supérieure gauche de la fenêtre. Elle renvoie de façons générale un nouveau jeu en ouvrant une fenêtre, en créent un rendu, un font, un pointeur sur un nouveau Tetris et en obtenant des valeurs nécessaires au bon fonctionnement du jeux.
Initialement nous ne savions pas comment faire pointer g->win, g->ren, g->font, g->freq et g->count. Nous avions laissé ce travail pour plus tard pour avancer et surtout parce que nous pension que c'etait inutile de le faire. Plus tard après l'étude de la SDL, nous avons compris comment le faire et pourquoi le faire, car c'est en réalité dans cette fonction même que le jeux est créé et donc il s'agit de l'unique emplacement de leur création.

```
Game *game_new(int x, int y)
{
  int w=510;
  int h=600;
  Game *g;
  g=malloc(sizeof(Game));

  if(!g)
    return NULL;

  //On affecte les textures à NULL, car elle ne sont utile que dans la fonction
    game_board_update.
  g->tex_s=NULL;
  g->tex_l=NULL;
  g->tex_nl=NULL;

  g->tet_offset_x=x;

  g->tet_offset_y=y;

  //Affectation d'un nouvelle fenêtre.
  g->win = SDL_CreateWindow("Tetris",g->tet_offset_x,g->tet_offset_y , w, h, SDL_WINDOW_SHOWN)
    ;
  //On teste l'erreur.
  if (!g->win)
    {
      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
      SDL_Quit();
      TTF_Quit();
      return NULL;
    }

  //Affectation d'un renderer.
  g->ren = SDL_CreateRenderer(g->win, -1, SDL_RENDERER_ACCELERATED);
  if(!g->ren)
    {
      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
      SDL_DestroyWindow(g->win);
      SDL_Quit();
      TTF_Quit();
      return NULL;
    }

  //Affectation de la bibliothèque "arial" à g->font
  g->font =  TTF_OpenFont("font/arial.ttf",25);
  if(!g->font)
    {
      SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "error : %s\n", SDL_GetError());
      TTF_CloseFont(g->font);
      TTF_Quit();
      SDL_Quit();
      return NULL;
    }


  g->tet = tetris_new();


  g->mino_size=30;

  //Permet d'avoir la valeur du conteur de haute résolution en seconde
  g->freq = SDL_GetPerformanceFrequency();
  //Permet d'avoir la valeur du conteur de haute résolution
  g->count = SDL_GetPerformanceCounter();
```

```
63
64   return g;
65 }
```

Premièrement on définit la hauteur et la largeur de la fenêtre avec l'initialisation de w et h. Ensuite on déclare un pointeur g vers la structure Game 3.1 pour la quelle on alloue de la mémoire. Et nous testons si la mémoire à bien été allouée.

Ligne 12, 13 et 14 on pointe les textures vers NULL car on leur affectera une mémoire dans la fonction game_board_update().

g->tet_offset_x et g->tet_offset_y point respectivement vers x et y pour le coin supérieur gauche de la fenêtre.

g->win pointe vers une fenêtre créer par la commande

SDL_CreateWindow("Tetris",g->tet_offset_x,g->tet_offset_y,w,h,SDL_WINDOW_SHOWN) ;

SDL_WINDOW_SHOWN peut prendre d'autre valeurs mais nous gardons celle-la pour notre projet. On teste aussi l'allocation

On fait également pointer g->ren vers SDL_CreateRenderer(g->win,-1,SDL_RENDERER_ACCELERATED) ; et on teste l'allocation.

Pour g->font on utilise TTF_OpenFont("font/arial.ttf",25) ; qui permet d'aller chercher dans les fichiers local la police arial en taille 25. On teste également l'erreur et agissons en conséquence.

On fait pointer g->tet vers un teris_new() ; 4.2.3 et g->mino_size vers 30 qui sera la taille en pixel d'un mino.

Pour finir g->freq point vers SDL_GetPerformanceFrequency() ; et g->count vers SDL_GetPerformanceCounter() ; Il est important de retrouner g à la fin.

4.3.4 game_del()

Cette fonction prend en argument un pointeur sur la structure Game et permet de libérer toutes les mémoires allouées par ce pointeur et de quitter le SDL et TTF. Elle est utilisée uniquement dans la fonction main.c 2, car elle permet de fermer le jeu une fois la boucle de game_run() "fini". Initialement lors du codage du jeu nous voulions libérer les mémoires et quitter SDL et TTF dans toutes les fonctions les utilisant, cependant après l'étude de la SDL, nous avons compris qu'elles n'étaient nécessaires qu'une fois (sans compter les testes d'échec d'allocation de mémoires), car les différentes commandes de la SDL et TTF doivent tourner en boucle pour que le jeu puisse continuer de fonctionner et donc il ne faut pas les détruire, ou quitter SDL ou TTF.
Le code de la fonction le suivant :

```
1  void game_del(Game *g)
2  {
3    //on libert toute la mémoire allouer par nos fonctions et on quite la TTF puis SDL.
4    //l'ordre semble être important
5
6    tetris_del(g->tet);
7
8    SDL_DestroyTexture(g->tex_s);
9    SDL_DestroyTexture(g->tex_l);
10   SDL_DestroyTexture(g->tex_nl);
11
12   TTF_CloseFont(g->font);
13
14   SDL_DestroyRenderer(g->ren);
15
16   SDL_DestroyWindow(g->win);
17
18   free(g);
19
20   TTF_Quit();
21
22   SDL_Quit();
23 }
```

Comme mentionné dans le programme l'ordre de l'utilisation des commandes semble être important, car en TD nous avons vu que pour une structure contenant un pointeur, il était nécessaire d'abord de libérer la mémoire

des données de la structure puis de libérer la mémoire du pointeur de structure, car il pouvait y avoir une perte de mémoire. Dans cette même logique nous avons préférer libérer les mémoires dans l'ordre de dépendance.
Donc nous commençons par libérer le pointeur sur tetris avec la fonction tetris_del() 4.2.4, puis nous libérons les textures permettant d'afficher le score, le nombre de lignes détruites et le niveau avec la commande

SDL_DestroyTexture(Texture) ;

qui prend une texture en argument et libère la mémoire allouée par celle-ci. Nous commençons par cette mémoire la, car elle est utilisée dans le rendu et donc pour éviter toute possibilité de perte de mémoire, nous l'avons placer en premiers.
Ensuite nous libérons la mémoire allouée par le pointeur de font avec la commande

TTF_CloseFont(g->font) ;

Nous faisons la même chose pour le rendu et la fenêtre

SDL_DestroyRenderer(g->ren) ;

SDL_DestroyWindow(g->win) ;

Nous le faisons dans cet ordre pour éviter une possible perte de mémoire.
A l'issue de ces libérations on peut libérer en dernier la mémoire allouée par g. Et enfin une fois que toutes les mémoires sont libérées nous devons quitter TTF puis SDL.

4.3.5 game_run()

La fonction prend en argument un pointeur de type Game et renvoie des évènements déclenchables manuellement par le joueur et des évènements continues propre au jeu.
Cette fonction fait partie des dernières à avoir été codée car il nous fallait coder les fonctions de teste de tetrimino avant. Dès que nous avions codé ces fonctions nous sommes aussitôt passé a son codage. Cependant il a fallu trouver les notations pour les évènements associés aux flèches directionnelles, à la barre d'espace et à la touche échappe. Nous avions trouvé quelques réponses dans un forum qui utilisait la bibliothèque SDL puis nous avions deviné pour le reste.
Le plus compliqué dans cette fonction a été la fin, car il fallait comprendre où il fallait coder la ligne pour faire descendre le tetrimino en fonction du temps et comment enclencher la suite d'évènements associées à la destruction de ligne. De plus pour cette fonction il à fallu créer une fonction qui n'était pas mentionnée, il s'agit de la fonction

tetris_matrix_update(g->tet) ;

```
1  void game_run(Game *g)
2  {
3    SDL_Event events;
4    int running;
5    int i;
6    int j;
7
8    running =1;
9
10   while (running)
11     {
12       Uint64 c;
13
14       while(SDL_PollEvent(&events))
15   {
16     switch(events.type)
17       {
18       case SDL_QUIT:
19         {
20     running =0;
21     break;
22         }
23       case SDL_KEYDOWN:
24         {
25     switch (events.key.keysym.sym)
26       {
27       case SDLK_q:
28         {
```

```
29          running = 0;
30          break;
31        }
32      case SDLK_ESCAPE:
33        {
34          running = 0;
35          break;
36        }
37      case SDLK_SPACE:
38        {
39          /*descendre immédiatement le tetrimino tout en bas et mettre à jour, si c'est
    possible. */
40          for(i=0;i<20;i++)
41      {
42        if(tetris_can_go_down(g->tet)==1)
43          {
44            g->tet->current_line+=1;
45          }
46      }
47          game_board_update(g);
48          break;
49        }
50      case SDLK_LEFT:
51        {
52          /* déplacer le tétrimino à gauche et mettre à jour, si c'est possible.*/
53          if(tetris_can_go_left(g->tet)==1)
54      {
55        g->tet->current_column -= 1;
56        game_board_update(g);
57        break;
58      }
59          else
60      {
61        break;
62      }
63        }
64      case SDLK_RIGHT:
65        {
66          /*déplacer le tétrimino à droite et mettre à jour, si c'est possible. */
67          if(tetris_can_go_right(g->tet)==1)
68      {
69        g->tet->current_column += 1;
70        game_board_update(g);
71        break;
72      }
73          else
74      {
75        break;
76      }
77        }
78      case SDLK_DOWN:
79        {
80          /* déscendre de une case le tétrimino et mettre à jour, si c'est possible. */
81          if(tetris_can_go_down(g->tet)==1)
82      {
83        g->tet->current_line += 1;
84        game_board_update(g);
85        break;
86      }
87          else
88      {
89        break;
90      }
91        }
92      case SDLK_UP:
93        {
94          /* faire pivoter le tétrimino dans le sens horaire et mettre à jour, si c'est
    possible. */
95          if(tetris_can_rotate_h(g->tet)==1)
96      {
97        if(g->tet->current_rotation +1 == 4)
98          {
99            g->tet->current_rotation =0;
100         }
101       else
102         {
```

```
            g->tet->current_rotation += 1;
          }
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
      case SDLK_n:
        {
          /* faire pivoter dans le sens anti_horaire et mettre à jour, si c'est possible. */
          if(tetris_can_rotate_ah(g->tet)==1)
      {
        if(g->tet->current_rotation -1 == -1)
          {
            g->tet->current_rotation =3;
          }
        else
          {
            g->tet->current_rotation -= 1;
          }
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
      }
    break;
        }
      }
}
      c= SDL_GetPerformanceCounter();

      if (update)
game_board_update(g);

      if ((float)(c - g->count)/g->freq > tetris_get_drop_speed(g->tet))
{
  g->count = c;
  if (tetris_can_go_down(g->tet))
    {
      //Permet de faire desendre le tétriminos chaque seconde.
      g->tet->current_line+=1;
      game_board_update(g);
    }

  //Permet de mettre fin à la boucle infini et donc du jeux.
  else if((!tetris_can_go_down(g->tet) && g->tet->current_line == 0) ||(!tetris_can_go_down(
  g->tet) && g->tet->current_line == 1))
  {
    running=0;
  }

  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
       */
      //Variable de contage
      int count;
      int count_lines=0;
      int count_lvl=1;

      //(Peut aider pour sauvegarder les couleur des tetriminos)
      //Permet de sauvegarder les tetrimino dans matrix.
      tetris_matrix_update(g->tet);

      //Compte si un ligne est remplie de mino.
      for(i=-2;i<=2;i++)
```

```
    {
      count=0;
      for(j=0;j<10;j++)
        {
          if(g->tet->matrix[g->tet->current_line-i][j] =='1')
      {
        count+=1;
        if(count==10)
          count_lines+=1;
      }
        }
    }

        if(count_lines==0)
    {
      //Label vers la fin de la fonction
      goto exit;
    }
        //Si des lignes sont remplies
        else if(count_lines!=0)
    {
      //Mise à jour du score
      if(count_lines == 1)
        {
          g->tet->score += 100*g->tet->level;
        }
      else if(count_lines == 2)
        {
          g->tet->score += 300*g->tet->level;
        }
      else if(count_lines == 3)
        {
          g->tet->score += 500*g->tet->level;
        }
      else if(count_lines == 4)
        {
          g->tet->score += 800*g->tet->level;
        }

      //Mise à jour de nbr_lines
      g->tet->nbr_lines += count_lines;

      //Boucle pour détécter le niveau
      for(i=1;i<=15;i++)
        {
          if(g->tet->nbr_lines >= i*10)
      count_lvl+=1;
        }
      //Mise à jour du niveau
      g->tet->level = count_lvl;
    }

        // actualisation du plateau de jeux après l'élimination des lignes
        tetris_shift_board(g->tet);
      exit:
        /*nouveau tetrimino */
        tetris_reset(g->tet);
        game_board_update(g);
      }
  }
    }
}
```

Pour décrire cette fonction nous allons nous intéresser aux évènementx souhaité.

Descendre immédiatement le tetrimino tout en bas et mettre à jour

```
case SDLK_SPACE:
        {
          for(i=0;i<20;i++)
      {
        if(tetris_can_go_down(g->tet)==1)
          {
            g->tet->current_line+=1;
          }
      }
```

```
          game_board_update(g);
          break;
        }
```

Dans cette partie du switch, si la barre d'espace est utilisée, elle sera reconnue à l'aide la commande SDLK_SPACE, une boucle for se lance pour parcourir toutes les lignes de la matrice et vérifier si la fonction tetris_can_go_down(g->tet) 4.2.10 est vrai, si c'est le cas le pointeur g->tet->current_line qui pointe sur la ligne actuelle du tetrimino est incrémenté de 1, la boucle se répète donc tant que la condition est vrai. Une fois que la condition est fausse on met a jour l'affichage et on sort du switch, se qui provoquera l'effet escompté.

Déplacer le tétrimino à gauche et mettre à jour, si c'est possible.

```
case SDLK_LEFT:
        {
          if(tetris_can_go_left(g->tet)==1)
      {
        g->tet->current_column -= 1;
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
```

Dans ce bout de code, si la touche "flèche gauche" est actionnée, et que la fonction tetris_can_go_left(g->tet) 4.2.6 est vrai alors le pointeur de la colonne actuelle du tetrimino diminue de 1 ce qui décale le tetrimino à gauche, puis la fenêtre est mise à jour et on sort du switch, sinon rien ne se passe.

Déplacer le tétrimino à droite et mettre à jour

```
case SDLK_RIGHT:
        {
          if(tetris_can_go_right(g->tet)==1)
      {
        g->tet->current_column += 1;
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
```

Ici comme pour l'évènement précédent si la flèche de droite est actionnée et que la condition de liberté de mouvement à droite tetris_can_go_right() 4.2.10 est vrai le pointeur vers la colonne actuelle du tetrimino est incrémenté de 1, on met ensuite à jour l'affichage et enfin on sort du switch, et sinon rien ne se passe.

Descendre de une case le tetrimino et mettre à jour

```
case SDLK_DOWN:
        {
          if(tetris_can_go_down(g->tet)==1)
      {
        g->tet->current_line += 1;
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
```

Ici si la flèche du bas est actionnée et que la condition tetris_can_go_down est vrai, alors le pointeur vers la ligne actuelle du tetrimino augmente de 1, puis on actualise la fenêtre et on sort du switch, sinon rien ne se passe.

Faire pivoter le tetrimino dans le sens horaire et mettre à jour

```
case SDLK_UP:
        {
          if(tetris_can_rotate_h(g->tet)==1)
      {
        if(g->tet->current_rotation +1 == 4)
          {
            g->tet->current_rotation =0;
          }
        else
          {
            g->tet->current_rotation += 1;
          }
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
```

Dans cet évènement nous avons dû passer par deux conditions car tout d'abord il fallait tester si nous pouvons faire une rotation horaire avec tetris_can_rotate_h 4.2.8 et ensuite en fonction de la valeur du pointeur de rotation actuelle modifier la valeur de ce pointeur en question.
Donc si la rotation horaire est possible dans ce cas il faut vérifier si le pointeur de rotation actuelle est égale à 4 ou non, car si c'est le cas il faut affecter à g->tet->current_rotation la valeur 0 pour rester dans le tableau tetrimino et également pour pouvoir effectuer de nouvelles rotations, sinon tout simplement incrémenter de 1 le pointeur de rotation. Au quel cas si nous ne pouvons pas effectuer de rotation, rien ne se passe.

Faire pivoter dans le sens anti-horaire et mettre à jour

Pour cette évènement la, il a fallu programmer une fonction supplémentaire pour le teste de rotation anti-horaire, la fonction tetris_can_rotate_ah() 4.2.9

```
case SDLK_n:
        {
          if(tetris_can_rotate_ah(g->tet)==1)
      {
        if(g->tet->current_rotation -1 == -1)
          {
            g->tet->current_rotation =3;
          }
        else
          {
            g->tet->current_rotation -= 1;
          }
        game_board_update(g);
        break;
      }
          else
      {
        break;
      }
        }
```

La touche associée à l'évènement est la touche n, si le test de rotation anti-horaire est vrais alors nous avons également ici deux conditions sur le pointeur de rotation actuel. De ce fait, si le pointeur est décrémenté de 1 est égale à -1 on lui affecte la valeur 3 pour rester dans le tableau tetrimino et pourvoir effectuer de nouvelles rotations, sinon on le décrémente normalement, puis on met à jours la fenêtre et on sort du switch. Et si le teste de rotation anti-horaire est faux rien en se passe

L'utilité de la boucle infini

```
    if (update)
  game_board_update(g);
```

Ces deux lignes de code permettent à la fenêtre de continuellement se recharger et ainsi de ne pas quitter le jeux. La condition se fait sur update qui est initialisé à 1 comme variable global 4.3.1, si elle reste vrai alors le jeu continue de tourner, c'est un peu plus loin qu'un condition pourrait provoquer le changement booléen de update et causer la fermeture du jeu.

Déroulement du jeux

Le jeux doit avoir un déroulement propre qui est le suivant : si le tetrimino peut descendre il descend, si il ne peux pas il s'arrête à la case actuelle. Si le tetrimino remplit une ligne en s'arrêtent, la ligne se détruit, le score est augmenter, ainsi que le nombre de ligne détruite et le niveau si le nombre de ligne détruite atteint un certain nombre, puis un nouveau tetrimino apparaît et le schéma continue. Si aucune ligne n'est remplie à l'issue de l'arrêt du tetrimino, dans ce que il n'y a pas de change et un nouveau tetrimino apparaît pour continuer la boucle. Et c'est à cela que sert la fin de la fonction.

```
if ((float)(c - g->count)/g->freq > tetris_get_drop_speed(g->tet))
  {
    g->count = c;
    if (tetris_can_go_down(g->tet))
      {
        //Permet de faire desendre le tétriminos chaque seconde.
        g->tet->current_line+=1;
        game_board_update(g);
      }

    //Permet de mettre fin à la boucle infini et donc du jeux.
    else if((!tetris_can_go_down(g->tet) && g->tet->current_line == 0) ||(!tetris_can_go_down(
    g->tet) && g->tet->current_line == 1))
    {
      running=0;
    }

    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
         */
        //Variable de contage
        int count;
        int count_lines=0;
        int count_lvl=1;

        //(Peut aider pour sauvegarder les couleur des tetriminos)
        //Permet de sauvegarder les tetrimino dans matrix.
        tetris_matrix_update(g->tet);

        //Compte si un ligne est remplie de mino.
        for(i=-2;i<=2;i++)
    {
      count=0;
      for(j=0;j<10;j++)
        {
          if(g->tet->matrix[g->tet->current_line-i][j] =='1')
           {
        count+=1;
        if(count==10)
          count_lines+=1;
           }
        }
    }

        if(count_lines==0)
    {
      //Label vers la fin de la fonction
      goto exit;
    }
        //Si des lignes sont remplies
        else if(count_lines!=0)
    {
      //Mise à jour du score
      if(count_lines == 1)
        {
          g->tet->score += 100*g->tet->level;
        }
      else if(count_lines == 2)
        {
          g->tet->score += 300*g->tet->level;
        }
      else if(count_lines == 3)
        {
```

```
67            g->tet->score += 500*g->tet->level;
68          }
69        else if(count_lines == 4)
70          {
71            g->tet->score += 800*g->tet->level;
72          }
73
74        //Mise à jour de nbr_lines
75        g->tet->nbr_lines += count_lines;
76
77        //Boucle pour détécter le niveau
78        for(i=1;i<=15;i++)
79          {
80            if(g->tet->nbr_lines >= i*10)
81        count_lvl+=1;
82          }
83        //Mise à jour du niveau
84        g->tet->level = count_lvl;
85      }
86
87          // actualisation du plateau de jeux après l'élimination des lignes
88          tetris_shift_board(g->tet);
89        exit:
90          /*nouveau tetrimino */
91          tetris_reset(g->tet);
92          game_board_update(g);
93        }
94       }
95      }
```

De la ligne 4 à la ligne 9 nous effectuons un test sur le tetrimino pour vérifier qu'il puisse descendre, si c'est le cas alors le pointeur de ligne est incrémenté de 1 et la fenêtre est rechargée, se qui va faire descendre le tetrimino.

Ensuite On effectue un teste à la ligne 12 pour vérifier si le jeu doit se terminer. Pour cela on vérifie que le tetrimino ne peux pas descendre et que le tetrimino est à la ligne 0 pour les tretriminos I et O , ou que le tetrimino ne peut pas descendre et que le pointeur de ligne du tetrimino est à la ligne 1 pour les autres tetriminos. Alors si la condition est respectée running = 0 et cela causera la fermeture de la fenêtre.

Le dernier else ligne 17, concerne le moment ou le tetrimino ne peut plus descendre, et la potentielle destruction de ligne.
Il nous faut tout d'abord déclarer des variables de contage, count pour le nombre de cases remplies dans la matrice, count_lines pour le nombre de lignes remplis et count_lvl pour le niveau actuel.
Ensuite il est nécessaire d'utiliser la fonction tetris_matrix_update() 4.2.12, créée spécifiquement pour sauvegarder le tetrimino présent dans le buffer, dans la matrice. Puis cela étant fait nous passons à la création de deux doubles boucles for pour parcourir les lignes situées dans la zone du tetrimino.
Nous parcourons les lignes de g->tet->current_line - 2 à g->tet->current_line + 2, et les colonnes iront de 0 à 10, cela nous permettra de parcourir les lignes sur lesquelles se posera le tetrimino. Ensuite la ligne 39 est une condition pour vérifier que la case est bien remplie, si c'est le cas, count est incrémenté de 1, et si la ligne est remplie count sera alors égale à 10, se qui provoquera une incrémentation de count_lines et donc augmentera le nombre de lignes à détruire par le code.
A l'issue de ce comptage, si le nombre de ligne à détruire est égal à 0 un label goto vers exite (ligne 51) nous amène à la fin du programme pour recommencer la boucle. Sinon il y aura plusieurs cas en fonction du nombre de ligne. Soit un multiplicateur de score noté m si le nombre de ligne est égale à 1 m=100, si le nombre de ligne est égale à 2 m=300, si le nombre de ligne est égale à 3 m=500, si le nombre de ligne est égale à 4 m=800. Alors en fonction de la valeur de count_lines on aura l'effet suivant :

$$\text{g->tet->score} += m \times \text{g->tet->level}$$

Ce qui aura pour effet d'augmenter le score actuel.
Ensuite il faut aussi ajouter le nombre de ligne détruite au nombre de ligne détruite actuel.
Nous avons également du faire une boucle for pour savoir à quel niveau nous sommes. Pour cela on teste si le pointeur de nombre de ligne est supérieur à 10×i, où i varie de 1 à 15. Si c'est le cas alors on incrémente le niveau et ensuite on affecte ce niveau au pointeur de niveau actuel.
Après on utilise la fonction tetris_shift_board() 4.2.13 pour faire descendre les minos jusqu'à la zone libre.
Puis on utilise tetris_reset() 4.2.5 pour amener un nouveau tetrimino et enfin on met à jour la fenêtre.

Chapitre 5

Conclusion

Nous avons réussit à programmer le jeu Tetris. Nous avons bien respecter toutes les contraintes du projet et nous avons créé une jeu fonctionnel sans bug. Le tetrimino descend correctement. Le score, le nombre de ligne et le niveau sont correctement affichés et évoluent de la bonne façons. Les vitesses sont respectées, les évènements sont fonctionnelles et le jeu se termine si le tetrimino atteint le haut du plateau. Le seul défaut est sur l'affichage des minos présent dans la matrice qui sont du même type que le tetrimino actuel, cela créer une certaine homogénéité, mais se n'ai pas l'effet initialement escompter.
L'élaboration du projet à été long et nous a demandé de nous adapter au programme. Il nous a fallu faire des recherches et revoir nos acquis en langage C. Parfois nous étions contraint de sauter des parties de ficher et parfois des fonctions entières pour pouvoir avancer. Bien que le langage C est un langage considéré comme compliqué pour créer un jeu, se projet nous à permis nous familiariser d'avantage avec l'utilisation des pointeurs et de la SDL.

Bibliographie

[1] **Cam**, COMMENT UTILISER LA SDL_TTF pour Afficher un SCORE en LANGAGE C - Partie 1(Expliqué en FRANÇAIS), *youtube*, 16 mai 2023, https://www.youtube.com/watch?v=NQZNHUoba-8&t=468s

[2] **Cam**, COMMENT UTILISER LA SDL_TTF pour Afficher un SCORE en LANGAGE C - Partie 2(Expliqué en FRANÇAIS), *youtube*, 3 juin 2023, https://www.youtube.com/watch?v=LS_eeI-9-pA&t=11s

[3] _itoa ()- Convertir un entier en chaîne, *IBM*, https://www.ibm.com/docs/fr/i/7.5?topic=functions-itoa-convert-integer-string

[4] strcat ()- Concaténer des chaînes, *IBM*, https://www.ibm.com/docs/fr/i/7.5?topic=functions-strcat-concatenate-strings

[5] strcpy ()- Copier des chaînes , *IBM* https://www.ibm.com/docs/fr/i/7.5?topic=functions-strcpy-copy-strings

[6] 3.10 Écrire du texte avec SDL_ttf, https://lappweb.in2p3.fr/~paubert/introductioncplusplus/4-0-3-10-0_2250.html

[7] [C] MasterMind avec la SDL2, https://perso.isima.fr/loic/unixc/tpc-sdlmaster.php#:~:text=1.6.-,%C3%89crire%20du%20texte%20%C3%A0%20l'%C3%A9cran,que%20l'on%20veut%20%C3%A9crire.

[8] **Karnaj**et **Pouet_forever** Utiliser la SDL en langage C, *zestedesavoir*, dernière mise à jour le jeudi 27 décembre 2018 à 21h01, https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/

[9] X11 color names, *Wikipédia*, dernière modification le 2 janvier 2025, https://en.wikipedia.org/wiki/X11_color_names#Derived_lists

[10] MESSAGE D'ERREUR, *Openclassrooms*, 9 mars 2016, https://openclassrooms.com/forum/sujet/message-d-erreur-29

[11] Générateur de nombre aléatoire entre 1 et 9, *Openclassrooms*, 16 novembre 2013, https://openclassrooms.com/forum/sujet/generateur-de-nombre-aleatoire-entre-1-et-9

[12] **Nicolas Joseph**, Les nombres aléatoires en C, *developpez*, 10 octobre 2005, https://nicolasj.developpez.com/articles/libc/hasard/

[13] **Lucas-84**, **Taurre** et **informaticienzero**, Les tableaux, Les tableaux multidimensionnels, 24 août 2024, https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1043_aggregats-memoire-et-fichiers/4281_les-tableaux/#3-14094_les-tableaux-multidimensionnels

[14] **Boris ('PrimFX')**, TUTOS C, *youtube*, https://www.youtube.com/playlist?list=PLEagTQfI6nPOWS4JPnxW5pRVgeyLuS5oC

[15] **Auteur externe**, La vérité sur les tableaux et pointeurs en C, *zestedesavoir*, 16 octobre 2018, https://zestedesavoir.com/tutoriels/2787/la-verite-sur-les-tableaux-et-pointeurs-en-c/#5-char-tab-et-char-p

[16] Rédaction d'un Rapport de Projet, https://web.umons.ac.be/app/uploads/sites/37/2018/06/Guide-El%C3%A9mentaire-pour-la-R%C3%A9daction-dun-Rapport-de-Projet.pdf