Pierrick Delrieu

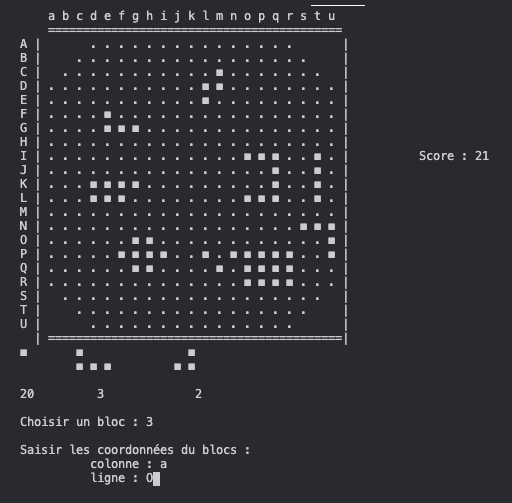
Romain Joreau

L1 - Groupe B

Projet de programmation en C

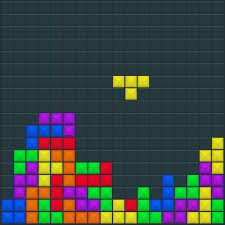
2019-2020

**COMME UN AIR DE TETRIS**



**SOMMAIRE**

1. L'architecture du projet
2. Le menu de présentation et de paramétrage du jeu
3. Les plateaux de jeu (cercle, losange et triangle)
4. Gestion de la création des blocs
5. Affichage des blocs en fonctions de la politique choisie et choix du bloc
6. Saisie des coordonnées et placement du bloc choisi
7. Gestion du plateau lors de l’annulation d’une ligne et calcul du score
8. L’architecture de la fonction main



**Le jeu Tetris** est un jeu vidéo sortie il y a plus de 40 ans. C’est un jeu de puzzle. C’est puzzle soumis à la gravité tombe de manière aléatoire et plus en plus fréquente. Le but du jeu et de survivre le plus longtemps possible en gardant le plateau non- remplit. Les puzzles peuvent être contrôlé de droite à gauche le temps de leur descente. Lorsqu’une ligne est pleine celle-ci est détruite. Le but du jeu est donc de placer les blocs de telle sorte à remplir les lignes.

L’objectif du projet était de réaliser un jeu similaire à celui-ci avec des conditions de jeu légèrement différentes. Le jeu que nous avons créé porte le nom de **"Comme un air de Tetris"**.

**Les règles du jeu :**

Le jeu "Comme un air de Tetris" se joue sur un plateau en cercle, en losange ou en triangle de minimum 19 cases de largeur. Le but du jeu est de placer des formes sur un plateau afin d'annuler des lignes et des colonnes. Contrairement au jeu Tetris classique, les blocs ne subissent pas la gravité. Si une ligne est annulée alors toutes les cases pleines au-dessus de la ligne descende d'un cran (il est possible d'annuler plusieurs lignes en même temps). Une ligne ou une colonne est annulée si et seulement si suite au placement d'une forme la ligne est complète (attention si la descente des blocs remplis une ligne ou une colonne, celle-ci n'est pas annulé). Le score correspond donc au nombre de cases annulées. Vous aurez seulement trois essais pour saisir des coordonnés valides pour le bloc choisi. Une fois ces trois chances épuisé la partie se finir et vous aurez atteint votre score maximal.

1. L'architecture du projet

Une image contenant capture d’écran, moniteur, téléphone, noir

Description générée automatiquementAu fur et à mesure du projet nous avons dû placer avec soin nos différents fichiers.

Le dossier source contient tous les fichiers de code C. Chaque fichier contient les fonctions qui lui corresponde. Le fichier "main.c" contient la fonction principale du programme.

Le dossier blocs contient trois fichiers de type ".csv" qui contiennent dans un tableau les différents blocs en fonctions de la forme du plateau (expliqué dans la suite du rapport).

Le fichier Makefile permet de faciliter la compilation d'un fichier exécutable en tapant simplement make dans le terminal car il est nécessaire de compiler en même temps les 5 fichiers .c. Le fichier exécutable main se situe alors au même endroit que ce dernier.

Le Readme.md est un dossier git qui permet de mettre quelques informations nécessaires au programme.

Le fichier "tetris.code-worspace" est un fichier propre a Visual Studio Code afin d'ouvrir plus facilement le projet sur cet éditeur.

1. Le menu de présentation et de paramétrage du jeu

Tout d’abord, nous avons réalisé un menu de présentation lorsque le joueur commence le jeu.

Le menu de présentation se situe à chaque début de partie et contient :

* Le nom du jeu avec le choix d'affichage des règles du jeu
* Le choix de la forme du plateau
* Le choix de la taille du plateau
* Le choix de la politique de suggestion des blocs

Chacun des quatre affichages ci-dessus contient une fonction dédiée :

void ecran\_accueil(void);

void choix\_plateau(int \* choix\_forme);

void saisie\_taille(int\* taille, int choix\_plateau);

void choix\_politique\_suggestion\_blocs(int\* choix);

L'affichage au niveau de la console se fait de manière brute avec des printf.

Les fonctions de retourne rien. La valeur saisie par l'utilisateur est enregistrée dans le paramètres modifié de la fonction. Ceci permet de réduire le nombre de variable nécessaire au programme. La saisie de l'utilisateur est sécurisé grâce a des boucles "faire tant que" (do while) et grâce a la réinitialisation du buffer d'entrée (stdin) par la fonction prédéfinie fflush(stdin);. Si l'utilisateur échoue la saisie alors un message d'erreur est affiché et une nouvelle saisie lui ai demandé.

On supprime la console à la fin de chaque fonction pour que le programme soit plus "esthétique". Afin de rendre le projet portable, il faut d'abord connaitre le système d'exploitation dans lequel on se situe. Ceci se réalise dans le fichier "fonction.h" à l'aide de condition de préprocesseur :

#ifdef \_WIN32

#define SYSTEME\_EXPLOITATION "win" *//windows*

#elif \_\_linux\_\_

#define SYSTEME\_EXPLOITATION "linux"

#elif \_\_APPLE\_\_

#define SYSTEME\_EXPLOITATION "mac"

#else

#define SYSTEME\_EXPLOITATION "unk" *//systéme inconnu*

#endif

La fonction "supr\_console" utilise donc la macro-constante définie en fonction du système d'exploitation pour effectuer la bonne saisie dans le terminal pour effacer la console :

void supr\_console(void)

{

system(SYSTEME\_EXPLOITATION == "win" ? "cls" : "clear");

}

1. Les plateaux de jeu

Le plateau de peut prendre trois formes possibles : cercle, losange, triangle. Il est prédéfini que la largeur du plateau doit être au minimum de 21 cases avec les affichages des numéros de colonnes compris. Soit le plateau réel de jeu doit avoir une taille de 20 cases de large. Cependant, l'affichage de ces trois types de formes demande une largeur impaire pour pouvoir être afficher correctement. Nous avons donc établi la taille minimale du plateau serait 19. Cette taille minimale fait l'objet du macro-constante dans le fichier "fonction.h".

Nous avons créé trois fonctions de création de plateau, une pour chaque forme :

int\*\* creation\_plateau\_losange(int taille);

int\*\* creation\_plateau\_triangle(int taille);

int\*\* creation\_plateau\_cercle(int taille);

En fonction de la forme du plateau choisie par l'utilisateur une fonction seulement parmi ces trois sera appeler. Le plateau est alloué de manière dynamique pour chacune des formes. La variable de retour est un pointeur sur la zone mémoire alloué (un int\*\*). Dans la fonction, le plateau est alloué de manière dynamique et initialisé en fonction de sa forme.

L'initialisation des plateaux se fait par des valeurs entières. Des macro-constantes sont définies dans le fichier "fonctions.h" afin de déterminer les cases pleines, les cases vides jouables et les cases vides injouables.

Lors du projet, de nombreuse allocation dynamique seront nécessaire pour un tableau 2D et ainsi de nombreuses désallocations auront lieu aussi. Nous avons donc ainsi créé deux fonctions dans le fichiers "fonctions\_usuelles.c" qui alloue et désaloue un tableau 2D de taille demandée :

void creation\_tableau\_2D(int\*\*\* tableau, int ligne, int colonne); *//allocation dynamique*

void desalocation\_tableau2D(int\*\* tableau, int ligne); *//dealocation dynamique*

La largeur et la hauteur de plateau pouvant être différentes, nous avons créé une structure contenant la hauteur, la largeur et le pointeur sur le tableau 2D du plateau.

Une image contenant ordinateur

Description générée automatiquementtypedef struct

{

int\*\* tableau;

int hauteur;

int largeur;

} Tableau2D;

Cette structure sera réutilisée pour les blocs.

Une image contenant blanc

Description générée automatiquement

L'affichage des plateaux se fait avec les codes ascii pour les espaces, les points, les lettres …

Les codes ascii étant prédéfinie dans des macro-constantes.



Cependant, sur mac n'étant pas possible d'afficher les codes ascii au-dessus de 127, on affiche un carré brut pour les cases pleines :

printf("◼︎");.

4. Gestion de la création des blocs

Dans le projet, la création des blocs est demandée d'être réalisé bloc par bloc avec une fonction indépendante pour chaque bloc. Chaque tableau 2D correspondant a un bloc devrait être contenue dans un tableau 1D ce qui ferai un tableau 3D. Or, le projet contient au totale plus de 50 blocs différents. Nous avons donc décidé de créer trois fichier Excel, un fichier Excel pour chaque forme de plateau car les blocs diffèrent en fonction de la forme.

Le fichier Excel est organisé de telle manière à ce que les blocs soient de taille 5x5 et qu'il y en ait 10 par lignes. Le fichier contient donc maximum 4 ranger de 10 blocs. Une fois le fichier Excel créer, il est converti en fichier csv pour faciliter sa lecture. Un fichier csv sépare les valeurs par des ";". Comme vu précédemment, les trois fichiers csv se situe dans un dossier "blocs" du projet.

Une image contenant fenêtre, grand, blanc

Description générée automatiquement

Voici ci-contre, le fichier csv correspondant aux blocs pour la forme cercle. Les blocs sont représentés par des 1 et les autres valeurs pour compléter la taille des blocs de 5x5 par des 0. Les "#" délimitent la taille réelle des blocs.

En construisant le fichier csv ainsi, il est facile de retrouver les coordonnées dans blocs dans ce tableau simplement avec son numéro :

x = ((num\_bloc-1) % 10) \* 5;

y = 4 + (5\*rang) ;

(Le rang correspond à sa ligne. Il est calculé sachant que chaque ligne contient 10 blocs).

A noter : les trois fichiers csv sans compter les ";" ont une taille de 50 x 20. Ces deux valeurs sont définies par des macro-constante dans le fichier "fonctions.h".

Nous avons donc eu besoin de créer une fonction qui lie le fichier est créer un tableau 2D correspondant. Cette fonction a besoin de connaitre la forme choisie par l'utilisateur pour ouvrir le bon fichier.

void creation\_tableau\_blocs(int tableau[HAUTEUR\_TAB\_BLOCS][LARGEUR\_TAB\_BLOCS], int forme);

Le tableau sera enregistré dans le tableau reçu en paramètre.

Nous devons d'abord créer un pointeur sur le fichier puis l'ouvrir. Avant la lecture du fichier, nous vérifions si le fichier est bien ouvert afin d'éviter les erreurs. Ensuite on lit le fichier et remplace dans le plateau en fonction de la valeur lit. Puis enfin, on ferme le fichier. Le fichier ne sera plus ouvert par la suite. Ici, on observe l'importance de l'architecture du projet vu précédemment.

La création d'un bloc se fait par la création d'une structure contenant sa hauteur et sa largeur réelle et un pointeur sur un tableau 2D alloué dynamiquement. On alloue le plateau dynamique pour pouvoir le créer quand on en a besoin et le supprimer quand on en a plus besoin. Le calcul de la hauteur et de la largeur réel du bloc se fond par deux fonctions distinctes qui repère le bloc dans le tableau créé précédemment et qui repère les "#". :

int hauteur\_bloc(int num\_bloc, int tab\_blocs[HAUTEUR\_TAB\_BLOCS][LARGEUR\_TAB\_BLOCS]);

int largeur\_bloc(int num\_bloc, int tab\_blocs[HAUTEUR\_TAB\_BLOCS][LARGEUR\_TAB\_BLOCS]);

Tout ceci fait l'objet d'un fonction : Tableau2D creation\_struc\_bloc(int num\_bloc, int tab\_blocs […][…]);

Cette fonction initialise les valeurs d'une structure en fonction du numéro du bloc. et retourne donc une structure correspondante au numéro du bloc demandé en paramètre.

1. Affichage des blocs en fonction de la politique choisie et choix du bloc par l'utilisateur

Au début du jeu, lors du menu de paramétrage du jeu, l'utilisateur a le choix entre de politique de suggestion des blocs :

* Affichage de tous les blocs possibles à chaque tour
* Affichage de trois blocs aléatoires

L'affichage de tous les blocs en une seul fois étant volumineux, nous avons décidé de les afficher trois par trois. L'utilisateur peut choisir d'afficher les trois blocs suivants ou de revenir aux trois blocs précédents.

Nous avons donc réalisé une fonction qui affiche trois blocs côte à côte avec leur numéro correspondant :

void affichage\_3\_blocs(int num\_bloc1, int num\_bloc2, int num\_bloc3, int tab\_blocs[…][…]);

La fonction créer les trois blocs en fonction de leur numéro, les affiches à l'aide de plusieurs boucles pour. A la fin de la fonction les tableaux alloués dynamiquement pour les blocs sont désaloué.

Cet affichage est essentiel pour l'affichage des blocs en fonction de la politique :

Pour i allant de 0 à 4

| Pour j allant de 0 à taille\_bloc\_1

| | …affichage avec les codes ascii…

| Fin pour

| Pour j allant de 0 à taille\_bloc\_2

| | …affichage avec les codes ascii…

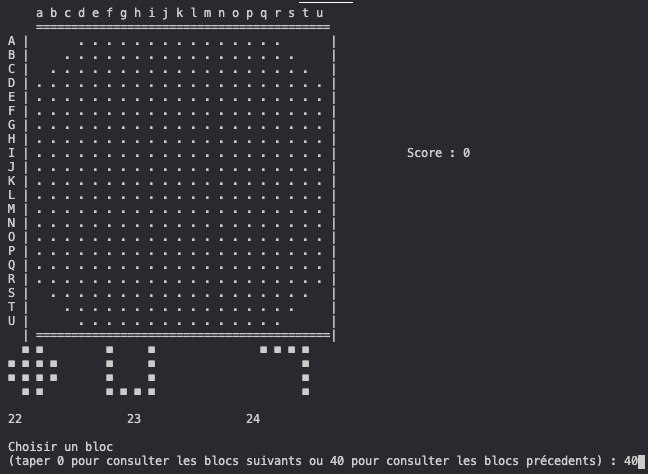
| Fin pour

| Pour j allant de 0 à taille\_bloc\_3

| | …affichage avec les codes ascii…

| Fin pour

Fin pour



L'affichage de la politique 1 (affichage de tous les blocs) correspond à un affichage du plateau et des blocs suivis d'une saisie. Les trois premiers blocs afficher sont les 1, 2 et 3. Si l'utilisateur saisie 0 alors la console est réinitialisée et l'affichage se répète avec les trois blocs suivants. Et inversement si l'utilisateur saisie, 40. La saisie de l'utilisateur est sécurisée, c’est-à-dire l'utilisateur peut saisir 0 ou 40 ou un des blocs afficher a la console. S'il n'y a plus de blocs disponibles alors l'utilisateur ne pourra plus saisir 0 ou 40 en fonction de l'extrémité à laquelle il se situe.

Une image contenant micro-ondes, ordinateur

Description générée automatiquement

L'affichage de la politique 2 (affichage de trois blocs aléatoires) correspond à un affichage du plateau et de trois blocs aléatoires. Mais, il est aussi composé d'une saisie sécurisée d'un des trois numéros de blocs aléatoire.

1. Saisie des coordonnées et placement du bloc

La saisie des coordonnés se fait sous forme de la saisie d'un caractère minuscule pour les colonnes et d'un caractère majuscule pour les lignes. Il est ensuite converti sous forme d'un entier correspondant à l'indice du tableau. Si l'indice de la ligne est supérieur à la largeur du plateau – la largeur du bloc alors l'indice prendra la valeur de 1. Si l'indice de la colonne est supérieur à la hauteur du tableau alors l'indice prendra la valeur de -1.

Ces deux indices sont enregistrés dans une structure "Coord".

La vérification de la saisie des coordonnées en fonction du bloc se réalise tout d'abord en vérifiant si un des deux indices précédents a la valeur de -1, ce qui signifie que les coordonnés ne sont pas valides. Si ces derniers ont une valeur valide, c’est-à-dire qu'il est possible de placer le bloc sur le plateau. Nous réalisons donc ensuite une deuxième vérification pour voir si à l'endroit où le bloc doit être poser se situe des cases vide jouables (1) où il est nécessaire.

Pour j allant de 0 à bloc.largeur-1

| Pour i allant de 0 à bloc.hauteur-1

| | Si (bloc.tableau[i][j] == 1) //vérification qui si la case du bloc est pleine

| | | Si (plateau.tableau[choix\_coord.y - bloc.hauteur + 1 + i][choix\_coord.x + j] != 1)

| | | | valide = 0;

| | | Fin si

| | Fin si

| Fin pour

Fin pour

La fonction retourne 1 si les coordonnées sont valides et 0 sinon.

Le fonction placement de bloc permet de modifier le plateau de jeu. Elle place le bloc sur le plateau sans regarder les coordonnées. La vérification des coordonnées dans la fonction principale se fait donc avant.

Le prototype de la fonction est :

void placement\_bloc (int num\_bloc, int tableau\_blocs[…][…], Tableau2D plateau, Coord choix\_coord);

Le placement du bloc se fait grâce au morceau de programme ci-dessous :

Pour i allant de 0 à bloc.hauteur – 1

| Pour j allant de 0 à bloc.largeur – 1

| | Si(bloc.tableau[i][j] == 1)

| | | plateau.tableau[choix\_coord.y - bloc.hauteur + 1 + i][choix\_coord.x + j] = 1;

| | Fin si

| Fin si

Fin si

Avec coord.x et coord.y les coordonnés saisie par l'utilisateur convertis en des nombres entiers.

1. Gestion du plateau lors de l'annulation d'une ligne et calcul du score

Lorsque l'utilisateur remplit une ligne, la ligne est supprimée et les blocs du dessus descende d'un cran. Cependant, lorsqu'une colonne est pleine, celle-ci est simplement supprimée.

Nous avons donc réalisé trois fonctions pour gérer ceci et à travers ces fonctions calculer le score :

* int etat\_ligne(Tableau2D plateau, int num\_ligne);
* int etat\_colonne(Tableau2D plateau, int num\_colonne);
* void annuler\_ligne(Tableau2D plateau, int num\_ligne, Score\* score);
* void annuler\_colonne(Tableau2D plateau, int num\_colonne, Score\* score);
* void empilement\_lignes(Tableau2D plateau,int num\_ligne, Score\* score);

Les fonctions "etat\_ligne" et "etat\_colonne" prennet en paramètre un numéro et retourne 1 si la ligne ou la colonne est pleine et 0 sinon.

Une fois l'état de la ligne ou la colonne vérifié, il faut donc supprimer cette dernière si nécessaire. Ceci ce fait grâce aux deux fonctions d'annulation.

Les fonctions états plus annulation se situe donc dans une boucle pour dans le programme principal qui parcours tout le tableau.

La gestion de la descente des lignes lors de l'annulation d'une ligne se fait grâce à la fonction "empilement\_lignes". On part donc de la ligne qui a était annulé puis on copie sur la ligne du dessus la ligne encore au-dessus et on supprime la ligne qui vient d'être remplacer. On fait ça jusqu’au haut du plateau.

Cependant, un problème arrive si une case pleine du plateau ne peut pas descendre car en dessous il se situe une case vide mais injouable. Afin de résoudre cela, lorsque nous rentrons dans ce cas particulier on remplace la case qui vient de descendre pas une autre valeur (3) pour pouvoir la différencier des autres et afin que les cases du dessus ne descende pas également. A la fin de la fonction, on parcours le tableau est on réhabilite au 3 la valeur d'une case pleine (1).

Pour i allant de num\_ligne à 0 par pas de -1

| annuler\_ligne(plateau, i, score); *//annulation de la ligne + sans calcul score*

| Si (i != 0) *//si =0 la ligne doit simplement être annulée*

| | *//copie de la ligne précédente dans la ligne actuelle*

| | Pour j allant de 0 à plateau.largeur

| | | Si (plateau.tableau[i-1][j] == 1)

| | | | Si (plateau.tableau[i][j] == 0)

| | | | | plateau.tableau[i][j] = 1;

| | | | else

| | | | | plateau.tableau[i-1][j] = 3;

| | | | Fin si

| | | Fin si

| | Fin pour

| Fin si

Fin si

*//réhabilitation des trois en case pleine* …

Le calcul du score se réalise lors de l'annulation de la première ligne qui était pleine. Lors de cette annulation, on compte le nombre de case qui viennent d'être annulé. C'est ainsi que le score est calculé.

Une fois ce dernier programme terminer, le projet était fonctionnel. En le testant durant une semaine, nous nous sommes rendu compte de quelques erreurs au niveau des indices du tableau … que nous avons pu modifier.

1. L'architecture de la fonction main

**Algorithme : main ()**

Variables locales : Tableau2D plateau, structure, plateau de jeu

taille, entier, taille du plateau

choix\_forme, entier, saisie utilisateur forme plateau

choix\_politique\_blocs, entier, 1 ou 2

nombre\_blocs, entier, nombre de blocs en fonction de la forme choisie

tab\_blocs[20][50], tableau 2D d'entier, tableau correspondant au fichier csv

choix\_num\_bloc, entier, saisie utilisateur choix numéro du bloc à placer

Coord, structure, saisie utilisateur coordonnées bloc

continuer, entier, condition arrêt de la boucle de jeu

score, structure

cpt, entier, compteur

Début

| ***//Menu***

| supr\_console();

| ecran\_accueil() //affichage commencer à jouer ou règles du jeu

| choix\_forme = choix\_plateau() //choix de la forme du plateau de jeu

| taille = saisie\_taille) //saisie de la taille du plateau de jeu

| choix\_poltique\_blocs = choix\_politique\_suggestion\_blocs); //choix de la taille du plateau

|

| ***//Initialisation en fonction des paramètres choisis***

| //Allocation et initialisation en fonction du choix de la forme du plateau

| Si (choix\_forme == 1) //cercle

| | plateau.tableau = creation\_plateau\_cercle(taille);

| | plateau.hauteur = taille;

| | plateau.largeur = taille;

| | nombre\_blocs = 32;

| sinon si (choix\_forme == PLATEAU\_LOSANGE) //losange

| | plateau.tableau = creation\_plateau\_losange(taille);

| | …

| sinon //triangle

| | plateau.tableau = creation\_plateau\_triangle(taille);

| | …

| Fin si

|

| creation\_tableau\_blocs(tableau\_blocs, choix\_forme); //creation du tab blocs (fichier csv)

|

| ***//Jeu***

| score.valeur = 0; //initialisation du score

|

| faire

| | continuer = 0; //initialisation de la condition d'arret

| |

| | //affichage des blocs + plateau + saisie choix\_bloc en fonction de la politique de suggestion choisie

| |

| | //Saisie coordonnée bloc

| | faire

| | | continuer ++;

| | | saisie\_coord\_bloc(…);

| | tant que ((continuer < 3) && (validite\_coord\_bloc(…) == 0))

| |

| | //Placement du bloc

| | Si (validite\_coord\_bloc(choix\_coord, choix\_num\_bloc, tableau\_blocs, plateau) == 1)

| | | placement\_bloc(choix\_num\_bloc, tableau\_blocs, plateau, choix\_coord);

| | Fin si

| |

| | //gestion de la descende et la suppresion des lignes

| | Si (etat\_ligne(plateau, cpt) == 1) //la ligne cpt est pleine

| | | empilement\_lignes(plateau,cpt,&score); //empilement + annulation de la ligne

| | Fin si

| |

| | //gestion de la suppression des colonnes

| |

| tant que (continuer <= 3)

|

| //fin partie affichage score

Fin

**CONCLUSION**

Le projet nous a tout de suite intéressé. Nous avons ainsi pris par la suite du plaisir à le réaliser.

Ce projet d’équipe nous a initié au monde de l’entreprise et a l'utilisation des commandes git. La mise en relation des différentes fonctions dans le projet principal nous a permis de travailler notre esprit de synthèse et notre réflexion informatique.

Les heures de travail consacrées à ce projet nous ont permis d’approfondir notre apprentissage de la programmation en C ainsi que nos connaissances, telles que les tableaux 2D, les différents affichages complexes, l’optimisation des fonctions, les structures, la lecture de fichier …

Ce projet nous a permis de comprendre que la programmation est aujourd’hui derrière toutes nos utilisations informatiques, nous sommes donc fiers d’avoir pu créer, de nos propres mains, un jeu similaire au jeu Tetris.