QUACH Kevin
Tuteur: FANG Wenjie

Université Gustave Eiffel 2021-2022
L3 Informatique

Rapport de Projet Tutoré: Tetris

Code couleur:
En rose: des commandes du terminal
En vert: les noms de fonction
En rouge: les types

Sommaire:

Sommaire: 1

Explication du sujet 2

Manuel d'utilisateur 2
- **Règles du Tetris** 3
- **L'interface** 4

Modularisation 5

Modèle MVC 6

Cahier des charges 6

Étape de conception 10

Création des structures de bases 10

Les tetrominos 11

Découverte de la librairie SDL 12

Représentation graphique 12
- **ASCII** 12
- **Graphique** 13

Implémentation du mode versus 16

Implémentation de l'IA 17
- **Création de l'IA** 17
- **Amélioration de l'IA** 19

Features prévus non implémentées faute de temps 19

Amélioration envisageable 20

Difficultés rencontrées 20

Ce que j'ai appris 21

Explication du sujet

Le sujet pour mon projet tutoré est un Tetris dans lequel je propose un mode solo ainsi qu'un mode versus. Le mode versus proposera la possibilité d'affronter un joueur ou une IA. Plusieurs niveaux d'IA peuvent être sélectionnés. J'ai choisi ce sujet car j'aime Tetris et je suis intéressé par le fonctionnement des IA. Les encouragements de Mr. Giraudo à choisir le projet tutoré ce semestre ont aussi joué un rôle dans ma décision de prendre cette option. De plus, je pensais que ça serait un bon challenge et une bonne expérience à avoir.

Manuel d'utilisateur

Pour compiler le programme, tapez "make" dans le terminal. Pour lancer le programme, tapez "./bin/Tetris".
Une fois le programme lancé, vous pouvez naviguer dans le menu à l'aide de la souris. Sélectionnez le mode de jeu que vous voulez jouer. Les modes de jeux disponibles sont les suivants:

- Solo : Jouez indéfiniment à un joueur, le seul but étant de ne pas perdre
- Versus:
    - Versus Player: Jouez contre un autre joueur en local
    - Versus AI: Jouez contre l'ordinateur

Une fois un mode de jeu sélectionné, le jeu débute. Les commandes sont les suivantes:

Joueur 1:

- Gauche: q
- Droite: m
- Soft drop: s
- Hard drop: espace
- Rotation Droite: z
- Rotation Gauche: f
- Hold: shift gauche

Joueur 2:

- Gauche: j
- Droite: l

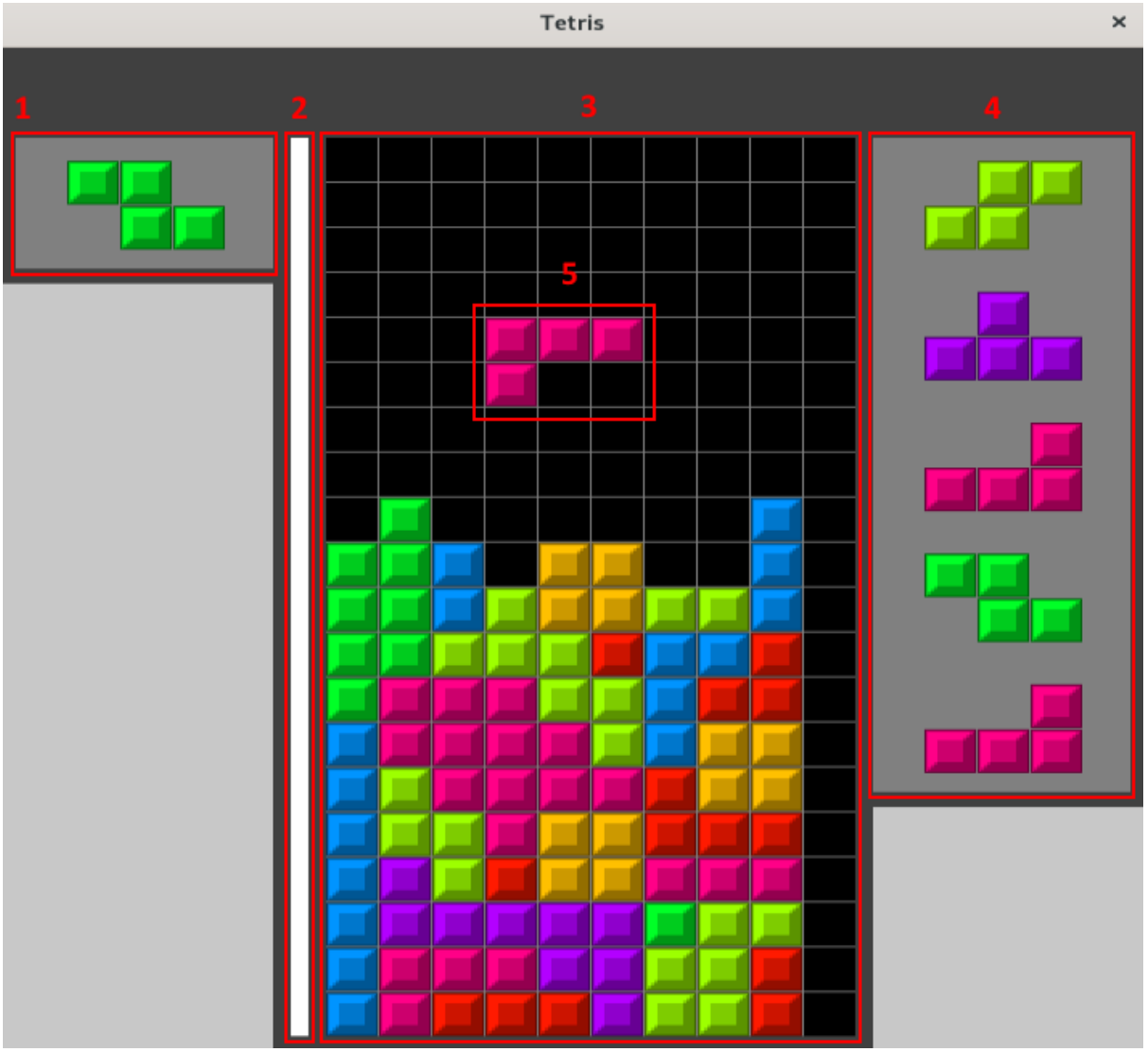
- Soft drop: k
- Hard drop: entrée
- Rotation Droite: i
- Rotation Gauche: m
- Hold: shift droit

Règles du Tetris

Tetris est un jeu dans lequel le joueur peut placer des pièces dans un plateau, les pièces sont posées le plus bas possible dans le plateau. Les pièces s'empilent l'une sur l'autre et le joueur perd si un bloc dépasse la hauteur maximale. Pour éviter ça, le joueur peut nettoyer des lignes en faisant des lignes de blocs complètes, sans aucun trou, cela fera disparaître cette ligne et fera descendre les blocs au-dessus de celle-ci. Pour avoir plus d'options à sa disposition, le joueur peut maintenir une pièce en "hold", cela va placer sa pièce actuelle dans la case hold et piocher la prochaine pièce de la file si la case hold est vide. Dans le cas où elle contient déjà une pièce, la pièce actuelle et la pièce en hold seront échangées et la position de la pièce sera réinitialisée au sommet du plateau. Le joueur ne peut utiliser cette action qu'une fois par nouvelle pièce piochée, après avoir utilisé un hold, il doit obligatoirement poser la pièce avant d'avoir à nouveau accès à cette action.

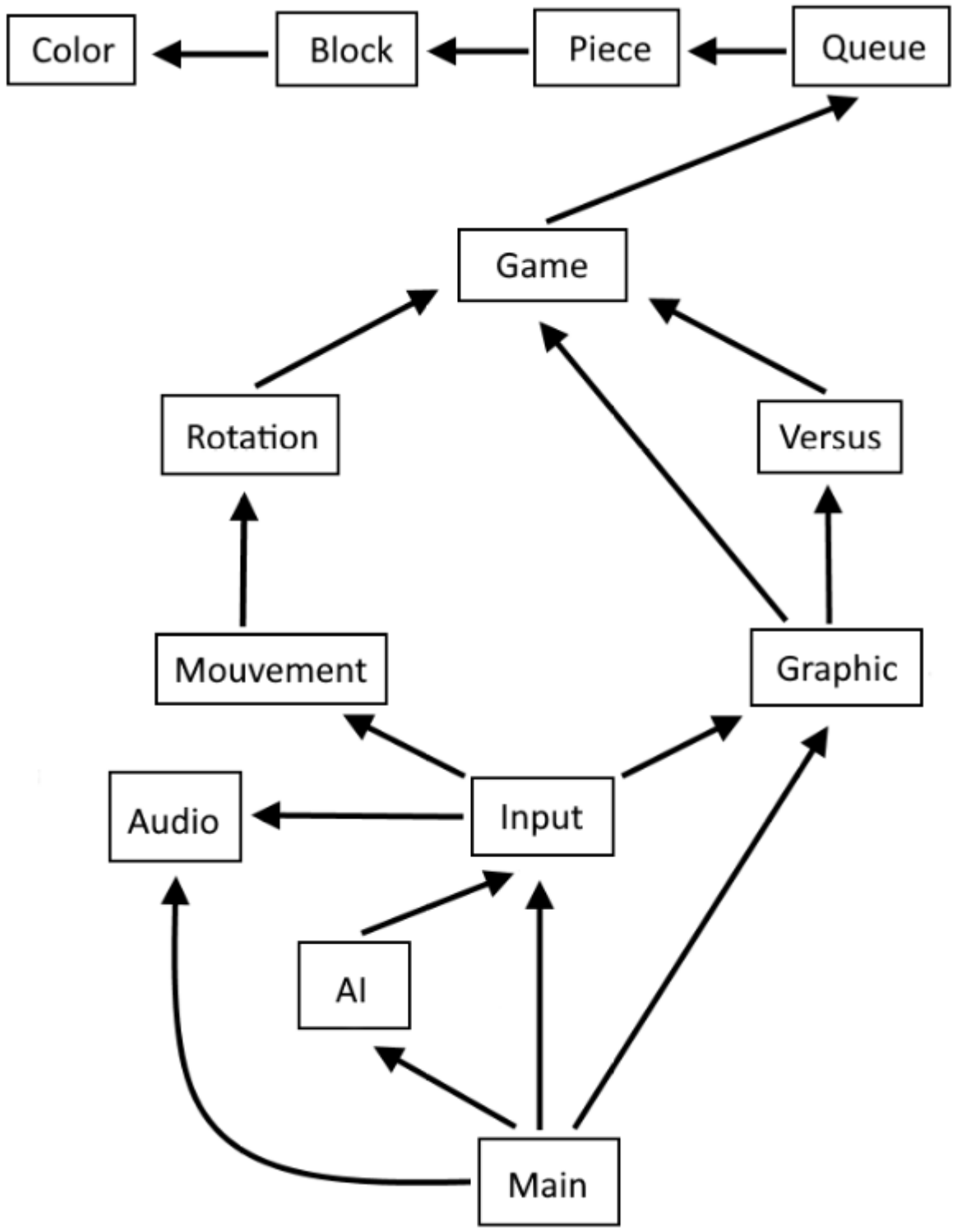
En mode versus, chaque joueur essaye d'éliminer sont ou ses adversaires en leur envoyant des déchets. Ces déchets s'accumulent dans une barre dans le camp de l'adversaire. Lorsque l'adversaire pose une pièce sans nettoyer de ligne, les déchets accumulés apparaissent sur le plateau sous forme de lignes avec un unique bloc vide sur la ligne. Ces lignes apparaissent au fond du plateau, poussant vers le haut les blocs déjà présents, augmentant ainsi les chances de défaite de l'adversaire. Faites attention cependant car ces lignes peuvent permettre à l'adversaire de contre-attaquer ! Pour envoyer des déchets à l'adversaire ou pour baisser la quantité de déchets accumulée dans votre barre, nettoyez plus d'une ligne d'un coup ou faites des combos. Pour plus de détails, voir la table de combo dans la partie "Implémentation du mode versus". Vous ne pouvez envoyer des déchets à l'adversaire que si votre barre de déchet est vide. Dans le cas contraire, la quantité de déchets dans votre barre sera réduite de la même quantité qui aurait dû être envoyée à l'adversaire.

L’interface

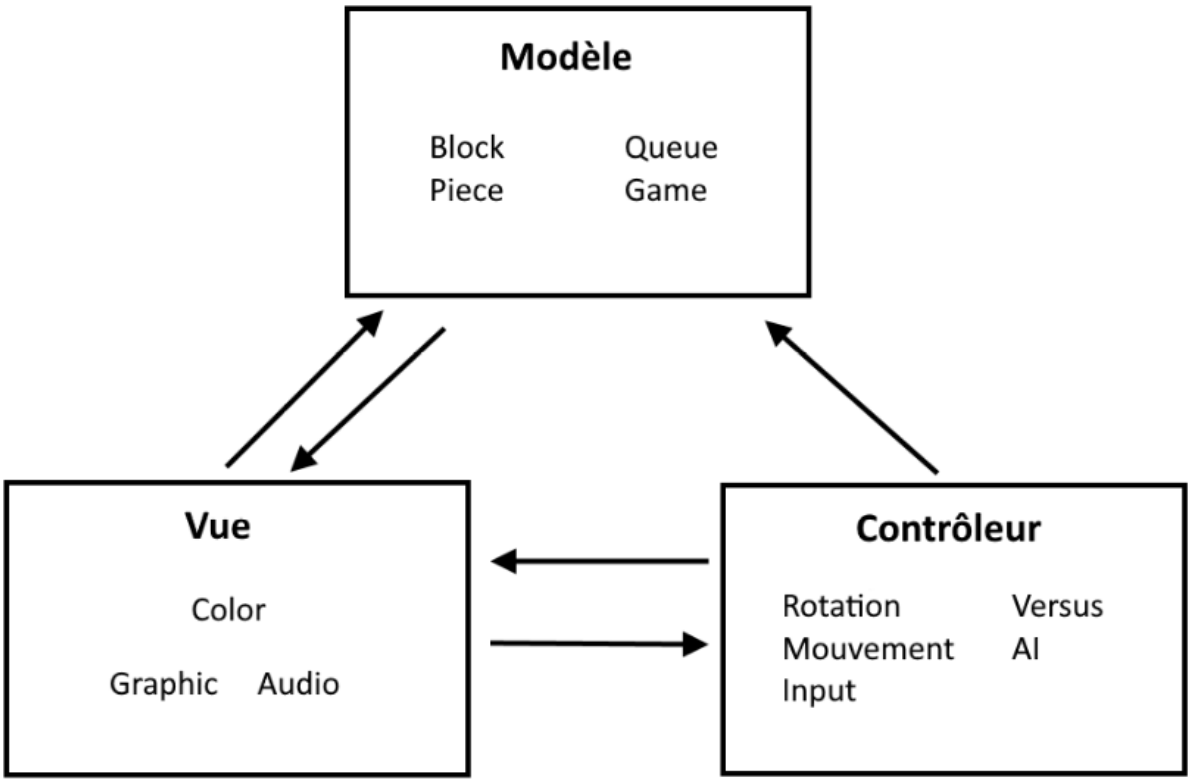


1. La case hold et la pièce hold
2. La barre de déchet
3. Le plateau
4. La file
5. La pièce actuelle

Modularisation



Modèle MVC



Cahier des charges

Color:

Ce petit module contient le type énuméré Color dont je me sers pour affecter une couleur aux blocs dans le module Block.

Block:

C'est dans ce module que je définis la structure Block, comme le nom l'indique, cette structure représente un bloc du plateau du jeu. Une fonction initiate_block permet de créer un Block vide ou rempli et de la couleur donnée en paramètre.

Piece:

Je définie la structure énuméré Type qui sera utilisé dans la structure Piece également définie dans ce module. Type désigne quel type de tetromino est la pièce. Le type Piece contient un champ block, tableau à deux dimensions, décrivant la configuration des blocs et un champ type décrivant le type de tetromino. Le champ block est nécessaire ici car

après une rotation, la configuration de la pièce change. Ce module contient de plus des fonctions pour générer des pièces aléatoirement parmi les 7 types ainsi qu'une fonction de copie de structure pour faire des copies profondes de la structure.

Queue:

Ce module gère la file des prochaines pièces. Elle contient une fonction pour générer les pièces de la file et une fonction permettant de renvoyer la première pièce de la file, décaler les pièces, et générer une pièce à la fin de la file.

Game:

Je définie dans ce module la structure Game représentant une partie pour un joueur. Il faut donc un Game par joueur. Cette structure stocke toutes les informations de la partie du joueur:

- le numéro de partie
- l'état du plateau (un tableau de blocs à deux dimension)
- la file
- la pièce en hold
- la pièce actuelle
- les coordonnées de la pièce actuelle
- le score
- un compteur de déchet
- un compteur de combo
- un flag pour savoir si le hold est disponible
- un flag pour savoir si la partie est perdue

Le module contient les fonctions permettant de vérifier et nettoyer les lignes remplies, ajouter le score au joueur, gérer le compteur de combo, et celle qui pioche la prochaine pièce de la file tout en générant une nouvelle pièce en fin de file. Il existe aussi une fonction permettant de vérifier s'il y a un conflit de position entre la pièce actuelle et le plateau, utile pour les rotations.

C'est aussi dans ce module que je définis des macros décrivant la taille du plateau et la position initiale des pièces. À noter que la taille réelle du plateau et la taille du plateau visible sont différentes. En effet, je permet au joueur de placer des blocs au-dessus de la limite visuelle du plateau servant de marge par rapport à la hauteur limite où le joueur perd. Entre

autres, la taille verticale visible du plateau est de 20 mais la taille réelle est de 22.

Rotation:

Comme son nom l'indique, ce module gère les rotations de pièces. Elle contient des fonctions modifiant configuration de la pièce passée en paramètre pour faire des rotations. Ces fonctions permettent de faire des rotations droite ou gauche.
Lorsqu'on fait la rotation d'une pièce, il est possible qu'un bloc remplis de la pièce atterrisse à la même position qu'un bloc également remplis du plateau, ou qu'un bloc de la pièce finisse en dehors du plateau, Il y a alors conflit. Pour gérer ces conflits, le module contient aussi une fonction d'ajustement. Cette fonction essaye de gérer les possibles conflits liés à la rotation d'une pièce en déplaçant la pièce de un ou deux blocs autour de la position actuelle. Si elle trouve une position ou la rotation ne cause pas de conflit, la rotation est effectuée et la position de la pièce est décalée à cette position. Si aucune position sans conflit n'est trouvée, la rotation est impossible, la pièce ne tourne pas.

Mouvement:

Ce module permet diverses actions que le joueur peut effectuer et gère d'autres mécanismes liés au mouvement de la pièce. Il contient les fonctions pour déplacer la pièce vers la gauche ou vers la droite et celles pour les soft drop et les hard drop ainsi que celle pour mettre une pièce en hold. Parmi les mécanismes de jeu, la tombée progressive de la pièce et le dépôt automatique lorsque la pièce est au fond du plateau est aussi gérer par des fonctions de ce module. Pour poser une pièce sur le plateau, il faut copier les blocs remplis de la pièce sur le plateau à sa position actuelle. Ensuite il faut tirer la prochaine pièce de la file. C'est ce que fait la fonction place_piece.

Versus:

Ce module gère les interactions entre deux plateaux dans le mode versus, notamment l'envois de déchet à l'adversaire, la réduction de sa propre ligne de déchet et l'apparition des déchets sur le plateau en cas de dépôt d'une pièce sans nettoyer de ligne.

Graphic:

Ce module s'occupe de l'affichage graphique du jeu. Pour cela, j'utilise la librairie SDL2, plus spécifiquement, SDL_Image. Les graphismes sont gérés par série d'image que je charge dans une liste et les affiches à des positions définis par des macros grâce à des fonctions de la librairies SLD. C'est ici que je définie de nombreuses macros décrivant la position sur l'écran de chaque partie du plateau et la taille de la fenêtre. J'y définit également quelque type énuméré correspondant aux images. Cela permet de rendre le code plus lisible en faisant référence aux images par leur nom plutôt que par leur numéro d'index dans le tableau où elles sont stockées.
Enfin, une structure Button permet de stocker les informations liées aux boutons du menu principal. Elle contient les coordonnées du bouton, le mode auquel il correspond et l'image correspondante.

Input:

Gère les saisies clavier du joueur. J'y définit un type énuméré KeyPress correspondant aux touches utilisées par le joueur. C'est aussi là que je définis les touches des joueurs 1 et 2 à l'aide d'un tableau de constante contenant des SDL_Keycode, type de la librairie SDL faisant référence aux touches du clavier. Ce module contient des fonctions faisant le lien entre les touches sur lequel le joueur appuie et les actions correspondantes dans le jeu (et dans le menu principal).

AI:

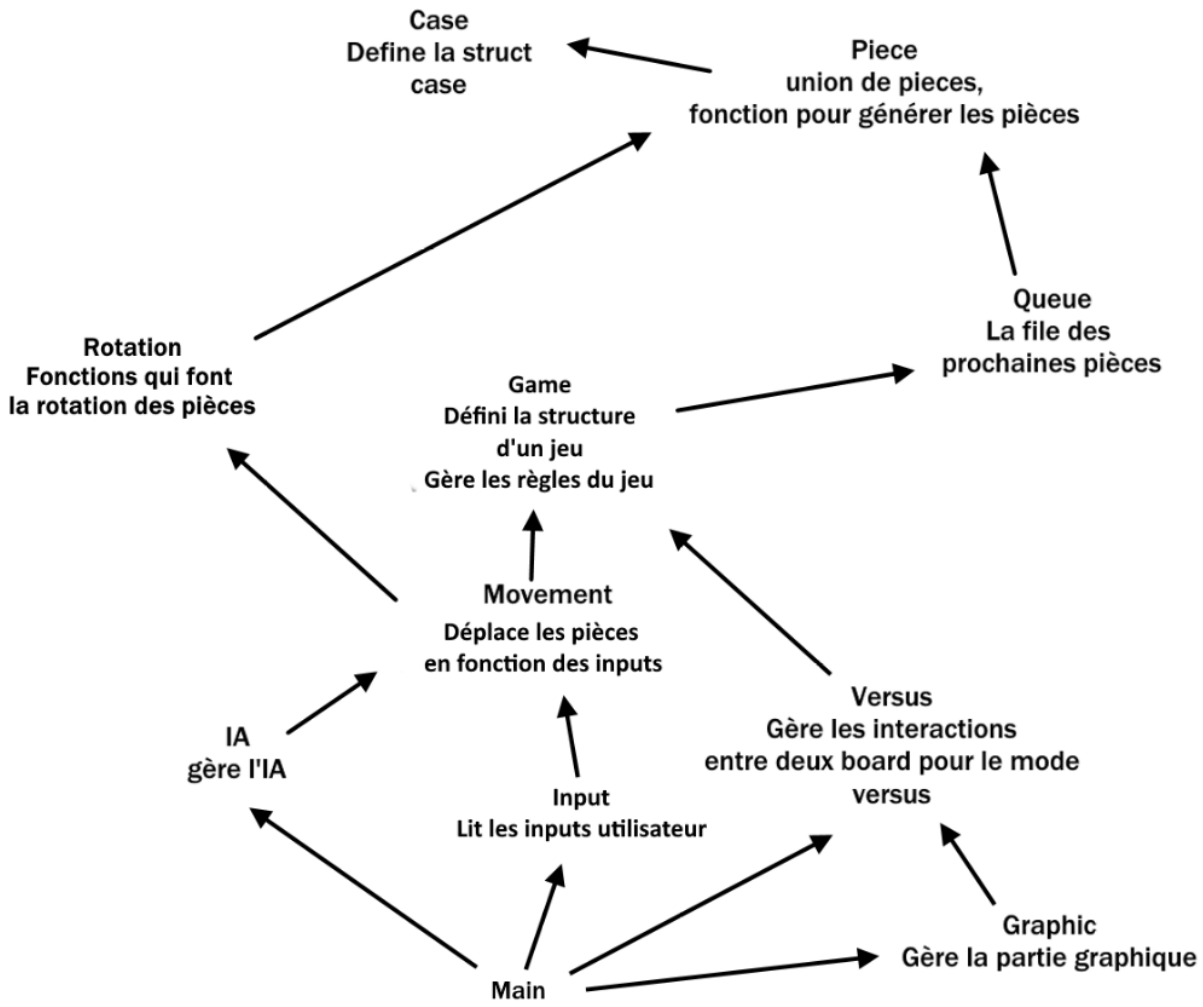
Le plus gros module du projet, ce module s'occupe de l'IA. Elle contient les fonctions de recherche et les fonctions d'évaluations de l'IA ainsi que les fonctions qui gère les actions de l'IA. Pour une explication plus détaillée du fonctionnement de l'IA, voir la partie correspondante.

Audio:

Ce module gère l'audio du jeu. Au moment de l'écriture de ce projet, ce module est encore vide. J'ai rencontré des problèmes lors de l'utilisation du module SDL_mixer m'empêchant de continuer le développement de ce module.

Étape de conception

Dans un premier temps, j'ai dû réfléchir à un découpage efficace du projet qui permettrait de coder chaque module de façon indépendante l'une de l'autre. Le modèle initiale était le suivant:



La modularisation initiale

Ce modèle changera légèrement au cours du développement du projet pour enfin arriver à la modularisation montrer plus haut.

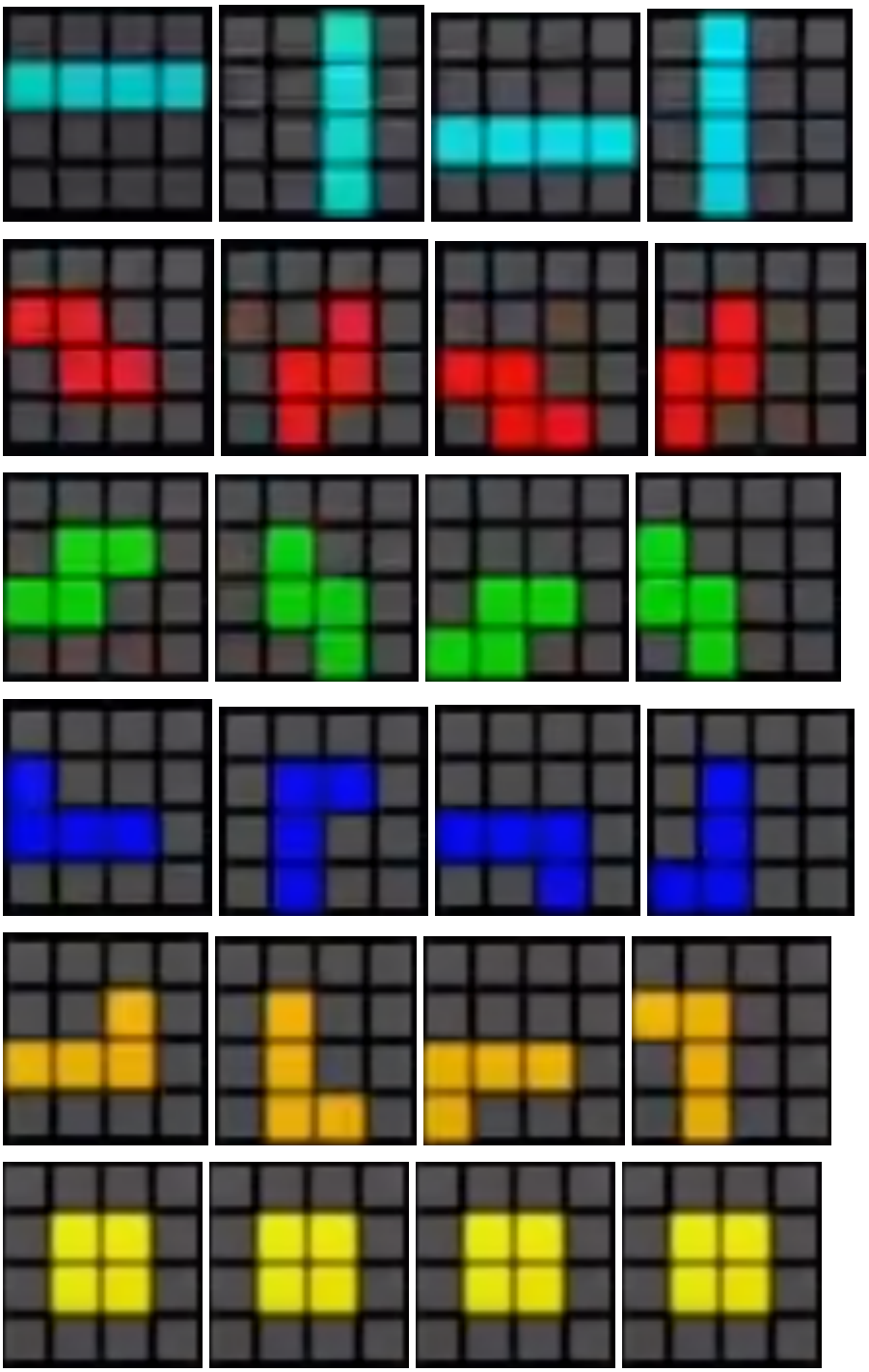
Création des structures de bases

Après avoir décidé du plan, J'ai commencé par créer les différentes structures dont je me servirais pour concevoir le Tetris. Une structure Block (initialement nommée Case) pour les blocs, une structure Piece qui est un tableau de bloc de taille 4 x 4. Une file Queue et la structure

Game. Les coordonnées de la pièce actuelle sur le plateau sont celles du bloc en haut à gauche de la pièce.

Les tetrominos

Pour la configuration des tetrominos et leurs rotations, j'ai utilisé pour référence les images montrées durant le chargement de Tetris 99:



Les Tetrominos et leurs différentes configurations

Chaque tetrominos sont des tableaux de Block de taille 4 x 4 dans lesquels les blocs sont soit vide soit remplis.

Découverte de la librairie SDL

Après discussion avec monsieur FANG, j'ai décidé d'utiliser la librairie MLV pour coder ce projet. N'ayant jamais utilisé cette librairie auparavant, j'ai dû aller me documenter sur le site SDL Wiki et me suis servi de quelques tutoriels sur le site Lazy Foo' Productions.
Pour gérer le déplacement et rotation de pièces, il était nécessaire de pouvoir récupérer les saisis du joueur. C'est grâce au type SDL_Event et la fonction SDL_PollEvent que je fais ça. Ensuite il fallait coder les fonctions exécutant les actions correspondantes à chaque touche et en faire le lien.

Représentation graphique

ASCII

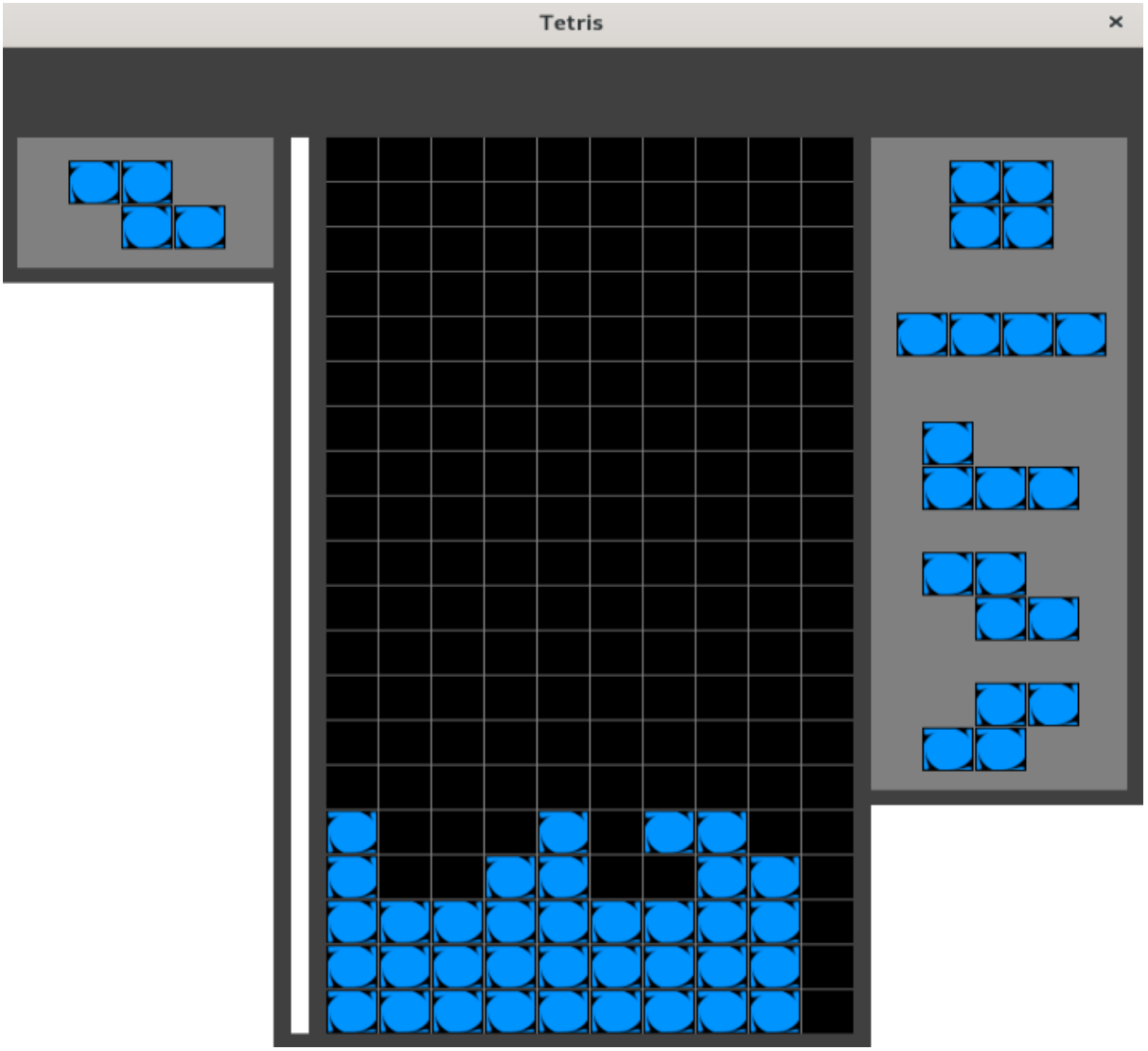
Pour pouvoir jouer au jeu, il fallait bien sûr pouvoir voir le plateau et les pièces. J'ai donc commencé par coder des fonctions pour les dessiner en ASCII dans le terminal. Dans cette version, la pièce hold et la file de pièces ne sont pas affichées.

Représentation ASCII du plateau

Graphique

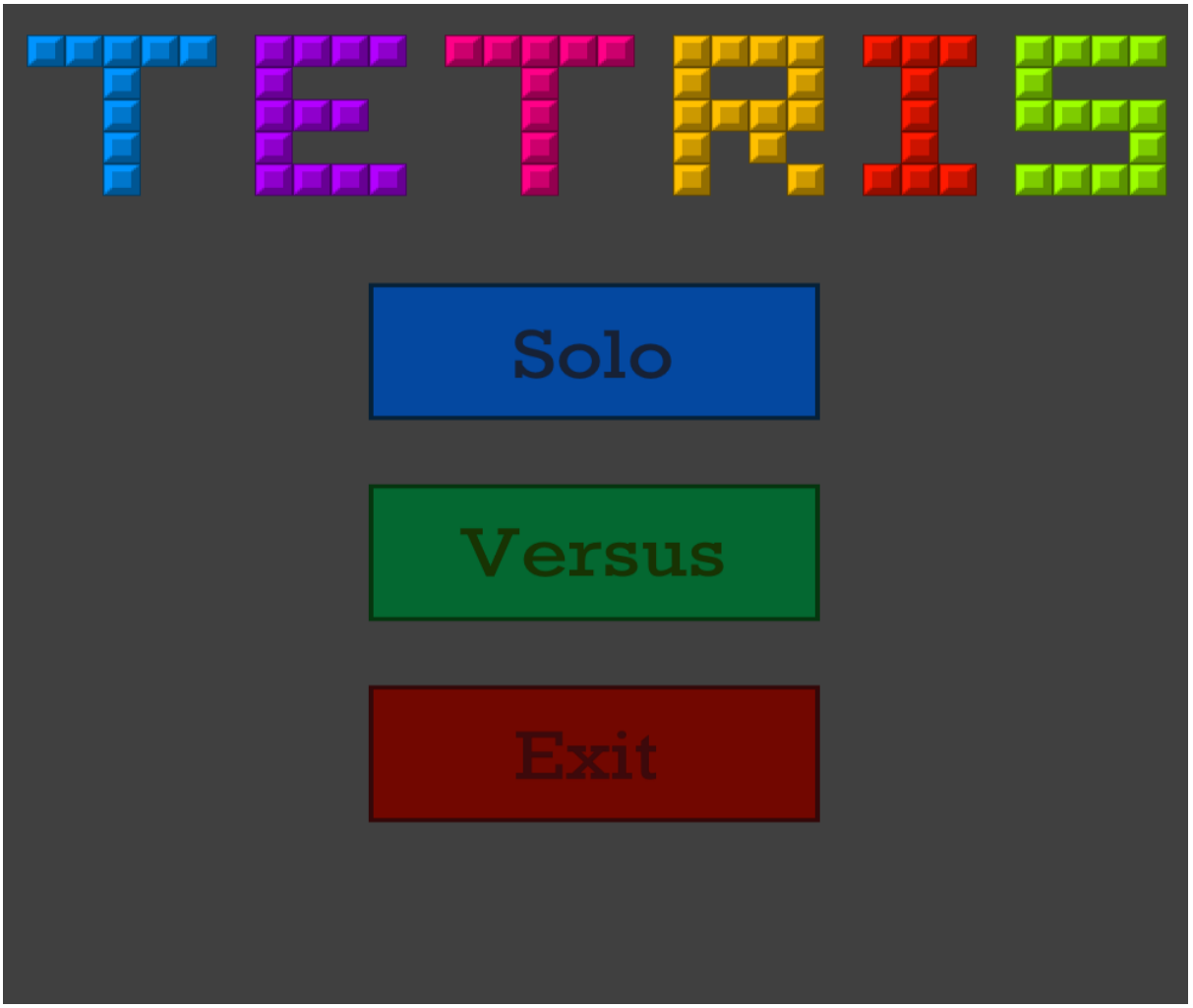
Après avoir vérifié que tout fonctionnait comme prévu et corriger les éventuels problèmes, j'ai travaillé sur une version graphique utilisant la librairie SDL_Image.

Pour cela, il a d'abord fallu dessiner les images que j'allais utiliser avec paint.net . Je me suis aussi documenté sur SDL_Image et, à l'aide des fonctions de la librairies, écris mes fonctions dessinant les différentes parties de l'interface. Ces fonctions font usages de macro que j'ai définis pour connaître la position des images à afficher.



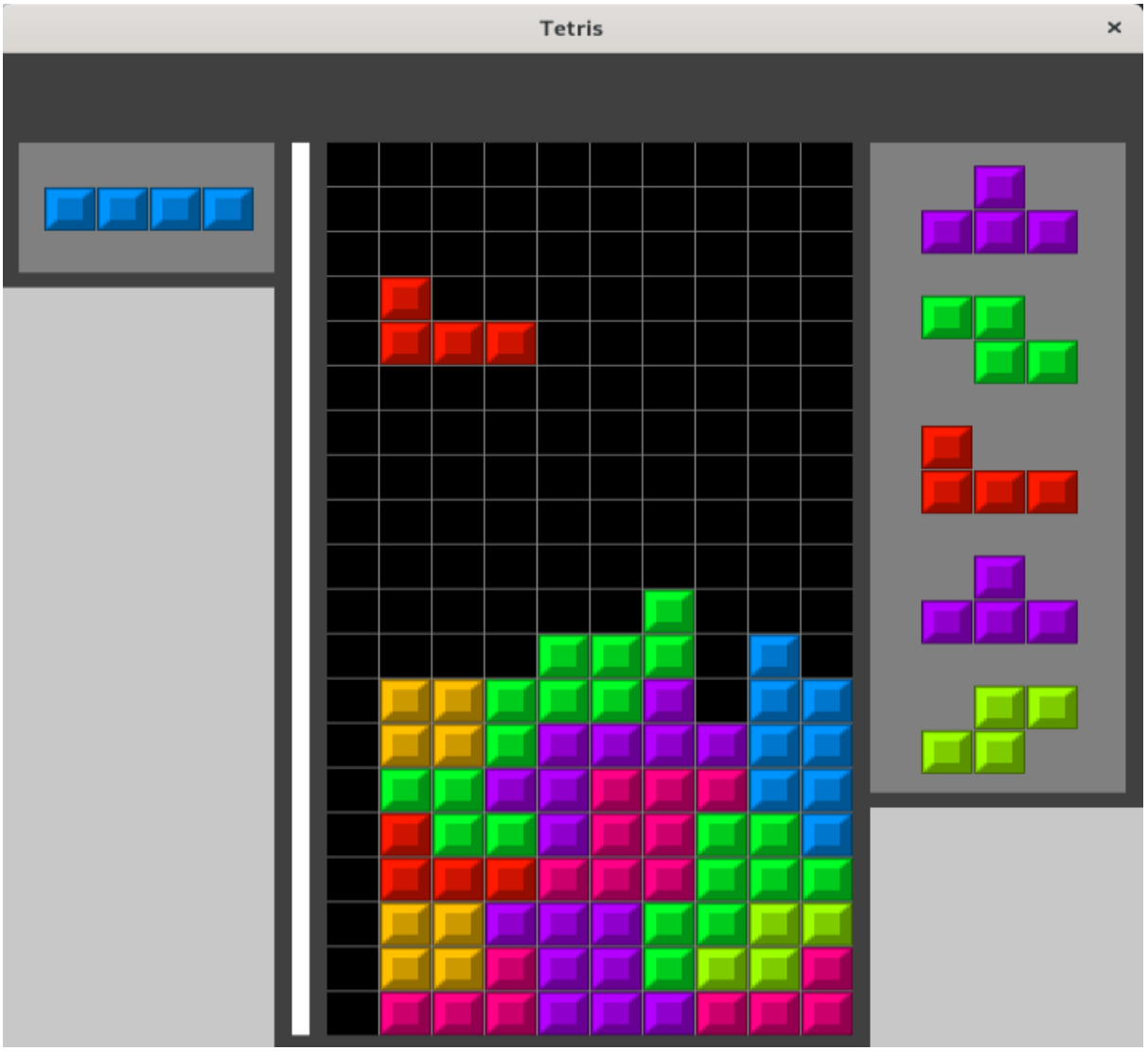
La toute première représentation graphique de mon Tetris

J’ai ensuite créé le menu principal du jeu dont les images ont également été dessinées via paint.net. Au départ, pour savoir sur quel bouton le joueur appuyait, je récupérais les coordonnées des clics et vérifiais si ces coordonnées correspondaient à celles d’un bouton, dans quel cas je lance le mode correspondant. Après discussion avec Mr. Fang, j’ai changé cette implémentation pour faire des boutons une structure dans laquelle se trouvait les coordonnées du bouton, le mode qu’il représente et l’image du bouton.



Le menu principal

Enfin, n'étant pas satisfait avec l'apparence de l'interface du jeu, j'ai ajouté un module Color et dessiné des blocs de différente couleur pour rendre le jeu plus coloré. Avec quelque modification supplémentaire, l'apparence final du jeu est la suivante:



L’apparence finale de l’interface de jeu

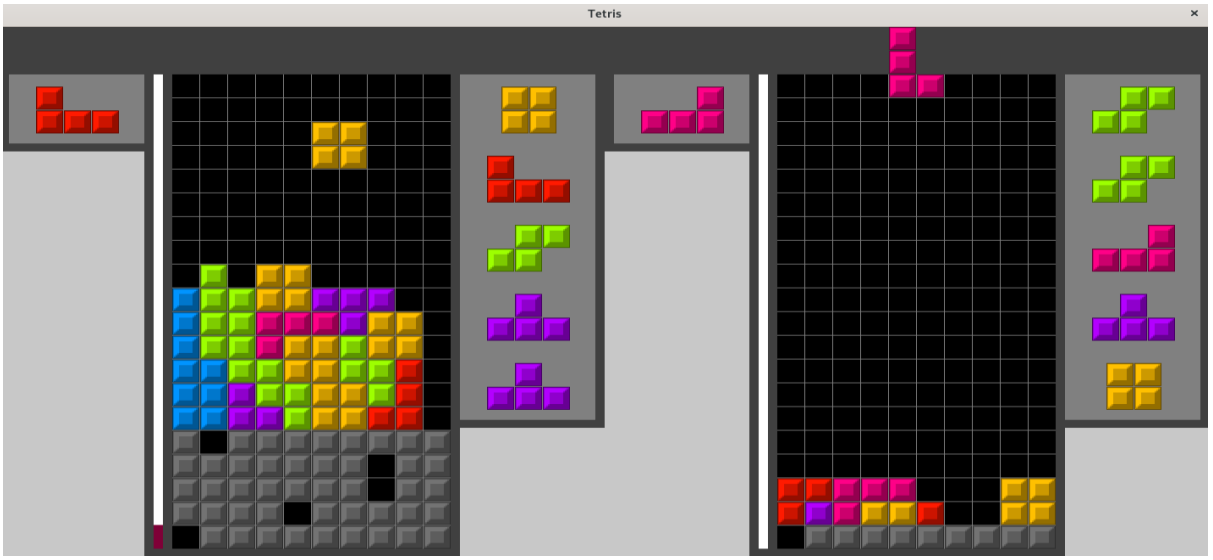
Implémentation du mode versus

Pour implémenter le mode versus. Il fallait définir des règles supplémentaires pour l'envoi et la réduction des déchets. Je me suis inspirée de la [table de combo de TETR.IO] pour définir ces règles, cependant bien plus simplifiée.

	Garbage sent at combo							
Attack Type	1	2	3	4	5	6	7	8
Single	0	1	1	1	2	2	2	3
Double	1	1	1	2	2	2	3	3
Triple	2	2	3	3	4	4	4	4
Quad	4	5	5	5	5	5	5	5

Ma table de combo

Le mode versus nécessitait également des fonctions différentes du mode solo pour gérer les jeux et leurs interactions. En mode versus, après chaque modification du plateau via dépôt de pièce, il fallait vérifier si des lignes devaient être nettoyées, ce qui est le cas également en mode solo. La différence étant de savoir s'il y avait des déchets à envoyer à l'adversaire ou à réduire de sa propre barre et, si oui, combien. Il fallait aussi écrire une fonction qui ferait apparaître les déchets sur un plateau.



Apparence du mode versus

Implémentation de l'IA

Mr Fang m'a expliqué le fonctionnement de l'IA au cours d'un rendez-vous. Je suis donc allé me documenter sur les game tree, des graphes sous formes d'arbre représentant tous les états possibles qu'on peut atteindre à partir d'un état initial.

Création de l'IA

Pour faire une IA, une fonction de recherche et une fonction d'évaluation sont nécessaires. La fonction de recherche cherche toutes les configurations de plateau atteignable à partir du plateau qu'on lui donne. On applique à chacune de ces possibles configurations la fonction d'évaluation qui va donner un score lui donner un score en fonction de différents critères. On garde en mémoire la configuration avec le score le

plus élevé ainsi que les actions nécessaires pour atteindre cette configuration. Après avoir trouver et évaluer toutes les configurations possibles, l'IA exécute les actions pour atteindre la configuration avec le meilleur score. Cette implémentation initiale de mon IA suivrait alors un algorithme glouton.

J'ai passé un certain temps à essayer de coder une fonction d'évaluation correcte, cependant, je n'évaluais pas les bon critère et je n'avais pas non plus une bonne combinaison de points pour ces critères. Le résultat étant que l'IA ne prenait pas la meilleure décision possible car la meilleure configuration a obtenu un score inférieur à une autre configuration moins idéale. J'ai fini par me documenter sur des sites qui m'ont beaucoup aidé pour avoir les bons critères et combinaison de points pour l'IA. J'ai éventuellement modifier légèrement ces points pour modifier le comportement de l'IA, notamment pour qu'elle évite le plus possible les trous et pour qu'elle nettoie les lignes plus souvent.

La fonction d'évaluation évalue les critères suivants:

- La somme de la hauteur de chaque colonne
- La bosselure
- Le nombre de trou dans la configuration
- Le nombre de ligne pleine
- Le coup fait perdre la partie

Explication des critères:

- Plus la hauteur totale de la configuration est haute, plus le joueur est proche de perdre. On réduit donc le score proportionnel à la hauteur de la configuration.
- Une configuration très bosselée, par exemple, si une colonne de hauteur 2 est voisine à une colonne de hauteur 10, est une configuration dans laquelle il est difficile de faire des coups optimals. De ce fait, plus la différence de hauteur entre chaque colonne est grande, plus on réduit le score.
- Des trous dans la configuration rendent la construction de ligne pleine plus difficile. Je définis un trou comme un bloc vide qui a, sur sa même colonne, un bloc plein situé à une hauteur supérieure à ce dernier. J'enlève une quantité fixe de point pour chaque trou dans la configuration.

- Lorsqu'une ligne est pleine, elle est nettoyée et la hauteur de la configuration diminue, c'est le but du Tetris ! Pour cette raison, nettoyer des lignes donne des points.
- Un coup qui fait perdre la partie retire un gros nombre de points, le but étant bien évidemment d'éviter de perdre.

Les sites que j'ai utiliser:

- https://hal.inria.fr/hal-00926213/document
- https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/

Amélioration de l'IA

Pour rendre l'IA plus performante, une recherche plus profonde dans le game tree est nécessaire. La raison est que, le meilleur coup pour une pièce n'est pas forcément le coup le plus optimal pour les pièces qui suivent. Une recherche plus en profondeur permettrait de prévoir les coups sur plusieurs pièces donnant ainsi à l'IA la capacité de "planifier" ses coups en explorant des solutions plus optimales.

Afin de permettre une recherche plus en profondeur, il a fallu garder dans un tableau les meilleures configurations et les mouvements pour l'atteindre après une première recherche puis relancer à nouveau une recherche sur ces configurations, ainsi de suite. J'ai créé une structure ConfigData pour stocker la configuration, son score et les mouvements afin de faciliter la garde en mémoire des configuration à chaque niveau de recherche. Pour éviter que la recherche ne soit trop gourmande pour le processeur, je ne garde que les 5 meilleures configurations à chaque niveau de recherche. Ce nombre est facilement modifiable car c'est une macro. Au moment de l'écriture de ce rapport, j'ai fait les préparations nécessaires pour la recherche en profondeur mais je n'ai pas encore écrit les fonctions nécessaires pour effectuer ces recherches.

Features prévus non implémentées faute de temps

- Du réseau, permettant à des joueurs de jouer en ligne.
- Un mode 40 ligne pour le mode solo, une course contre la montre où le joueur doit nettoyer 40 lignes le plus vite possible. Coder ça

ne serait pas trop compliqué, cependant, il faudra aussi dessiner les boutons et faire un autre menu pour les sous-modes du mode solo. Il serait aussi intéressant de sauvegarder les meilleurs records dans un fichier.
- De l'animation et petits effets de particules pour les pièces
- De l'audio, musique et sound effects. Malheureusement je n'ai pas réussi à implémenter l'audio avec SDL_Mixer, mais j'envisage de le faire avec une autre librairie comme FMod par exemple.

Amélioration envisageable

- En priorité, finir les fonctions pour la recherche en profondeur de l'IA.
- Permettre à l'IA de faire une nouvelle recherche à chaque position horizontale du plateau, ce qui lui permettrait de décaler des pièces à l'horizontale pour remplir les trous causés par des pièces S ou Z par exemple. Cependant cela risque d'être très gourmand car il faut faire une recherche pour chaque hauteur du plateau.
- Implémenter les T-spins dans la table de combo.
- Permettre de faire des parties à plus que 2 joueurs.
- Ajouter un "contour" à l'endroit où la pièce sera posée pour que le joueur puisse mieux visualiser l'endroit où la pièce va se poser (Je pose parfois la pièce une colonne trop loin de là où je voulais la poser).
- Accélérer la vitesse de tombée des pièces au cours de la partie, ne semble pas très difficile à coder.

Difficultés rencontrées

- Gérer la file des pièces était un peu difficile, je l'avais implémenté comme une liste chaînée au départ, pensant que ça aurait été plus efficace. Cela causait des problèmes de shallow copy lors de la recherche de configuration de l'IA. J'ai fini par changer la file en une liste simple.
- Gérer les propriétés de rotations (sur les bords ou quand il y a des pièces juste en dessous)

- Le changement d'implémentation des boutons vers une représentation par structures a nécessité la modification et adaptation de quelque fonction.
- Beaucoup de problèmes avec l'IA, notamment la fonction d'évaluation qui faisait jouer l'IA de façon non optimale. L'ordre des actions de l'IA est aussi important, ce dont je n'ai pas fait attention au départ. Par exemple, si l'IA fait d'abord les mouvements à l'horizontale avant les rotations, il était possible que, si la pièce devait être déplacée au bord du plateau, la rotation pouvait causer la pièce à ajuster sa position et donc être décalée par rapport à position prévue.
- Trouver une bonne combinaison de score pour l'IA. Sans une bonne combinaison avec une bonne balance, l'IA pouvait avoir un trop grand penchant pour certaines configuration et, au contraire, éviter complètement d'autres.
- Optimiser l'utilisation de la mémoire en plaçant des nanosleep aux bons endroits dans les boucles.
- Quelques fautes de frappe m'ont coûté beaucoup de temps: rotate_right et rotate_left étaient inversées dans la fonction de recherche de l'IA, ce qui faisait jouer à l'IA des mouvements absurdes. Je pensais que le problème était dans la fonction d'évaluation.
- L'utilisation de SDL_Mixer.
- Fuite de mémoire dans la file de pièces lorsque l'IA exécute sa fonction de recherche. Il s'agissait d'un problème de shallow copy lié au fait que la file était un tableau de pointeur de Piece. Pour y remédier, j'ai transformé la file en un tableau de Piece.

Ce que j'ai appris

Ce projet m'a permis d'avoir une meilleure compréhension du fonctionnement d'une IA, de leur fonction de recherche et fonction d'évaluation. J'ai aussi appris à coder plus proprement, utilisant des macro et des types énumérés pour améliorer la lisibilité du code, par exemple en utilisant ces types énumérés pour faire référence à un index de tableau. Écrire des modules indépendants m'a permis d'éviter que des problèmes dans un module cascade aussi dans d'autres modules.

En général, écrire les modules de cette façon a rendu le projet bien plus facile à gérer. Ce projet m'a aussi permis d'avoir une première expérience avec la librairie SDL que j'ai utilisé pour une grande majorité du projet. De plus, les rendez-vous avec monsieur Fang (toutes les deux semaines) m'ont forcé à être plus régulier dans mon travail. Enfin, parmi tous les projets sur lesquels j'ai pu travailler jusqu'à présent, celui-ci a été le plus long et ça a été une très bonne expérience à avoir.