Nathan GACHET

TD javaScript : Réalisation d'un jeu de TETRIS

Table des Matières

Description du projet

Jeu du TETRIS

Le but de ce TP va être de réaliser avec un mélange d'HTML et de javaScript un jeu de TETRIS sur navigateur simple. Un jeu de TETRIS se présente de la manière suivante :

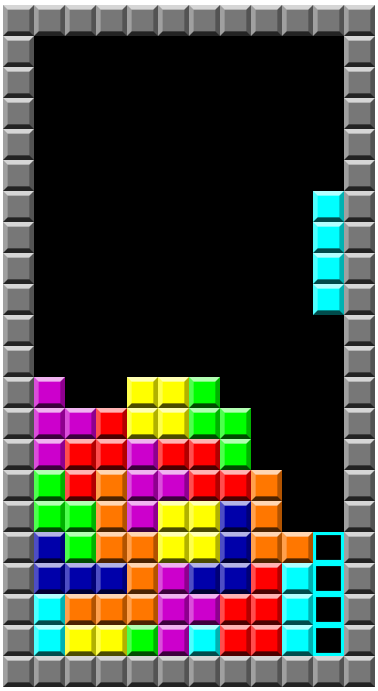
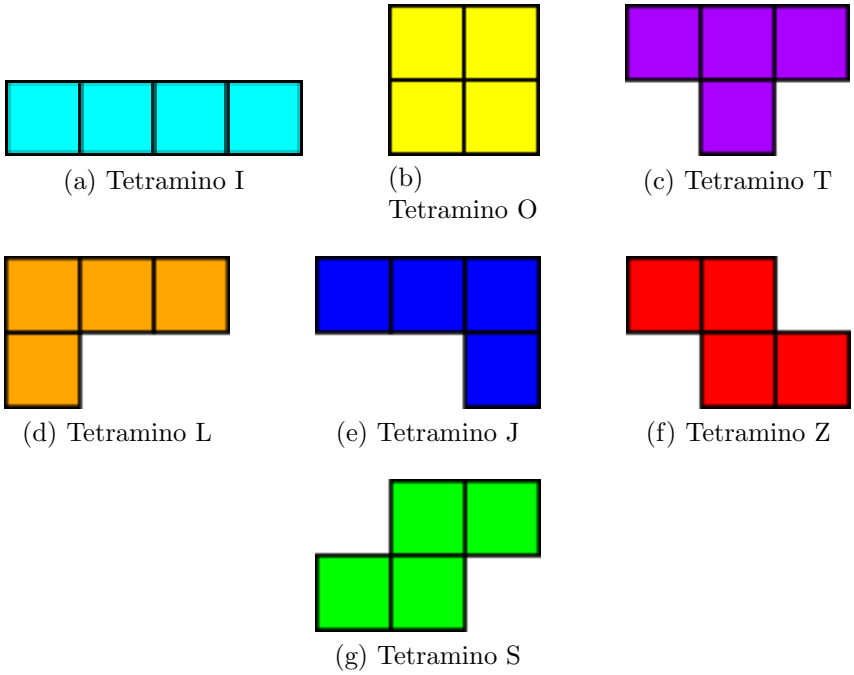


Figure 1: Image d'un jeu de TETRIS

- Un jeu de Tetris est un tableau composé en général de 10 colonnes et de 20 lignes. Ce tableau est initialement vide. (en noir sur l'image)
- Le tableau se met à jour à intervalles réguliers, qu'on appelera des *tics*.
- Au début du jeu, une pièce de puzzle appelée *Tétramino* tombe depuis le haut du tableau. Un tétramino descend globalement d'une case à chaque tic.
- Un tétramino peut prendre 7 formes différentes, qui sont les suivantes :



(a) Tetramino I

(b) Tetramino O

(c) Tetramino T

(d) Tetramino L

(e) Tetramino J

(f) Tetramino Z

(g) Tetramino S

Figure 2: Formes de tétraminos

- Le joueur peut contrôler le déplacement du tétramino pendant sa chute. Il peut le faire bouger vers la gauche du tableau, la droite du tableau, le faire tourner de 90^o, et le faire tomber plus vite.
- Lorsque le tétramino entre en contact avec le bas du tableau, ou un autre tétramino, il est alors figé sur place, dans sa position courante, et un nouveau tétramino aléatoire tombe depuis le haut du tableau.
- Si une ligne entière du tableau est remplie de tétraminos, cette dernière disparaît, le joueur marque des points, et tous les tétraminos déjà figés tombent d'une ligne vers le bas.
- Si jamais un tétramino se retrouve figé tout en haut du tableau, le joueur perd la partie.
- Dans notre exemple, les tics auront lieu toutes les 1000 ms, un bloc de tétramino fera 30 px de côté, les joueurs marqueront 10 pts par ligne de tétramino remplie, et les couleurs des tétraminos seront les mêmes que celles de la figure 2 (couleurs officielles du jeu TETRIS).

Organisation du projet

Le projet HTML inclus avec ce sujet de TD contient :

- Un fichier index.html qui contient le projet.
- Un dossier css qui contient un fichier style.css, qui au début ne sert à rien.
- Un dossier js contenant 4 fichiers de code javaScript qui programment le jeu de TETRIS du projet : constants.js / piece.js / gameModel.js / tetris.js

Démarrez Visual Studio Code, et dans le menu *File*, sélectionnez l'option *Open Folder*. Sélectionnez ensuite le dossier *TP-Tetris* dans lequel ce document se trouve.

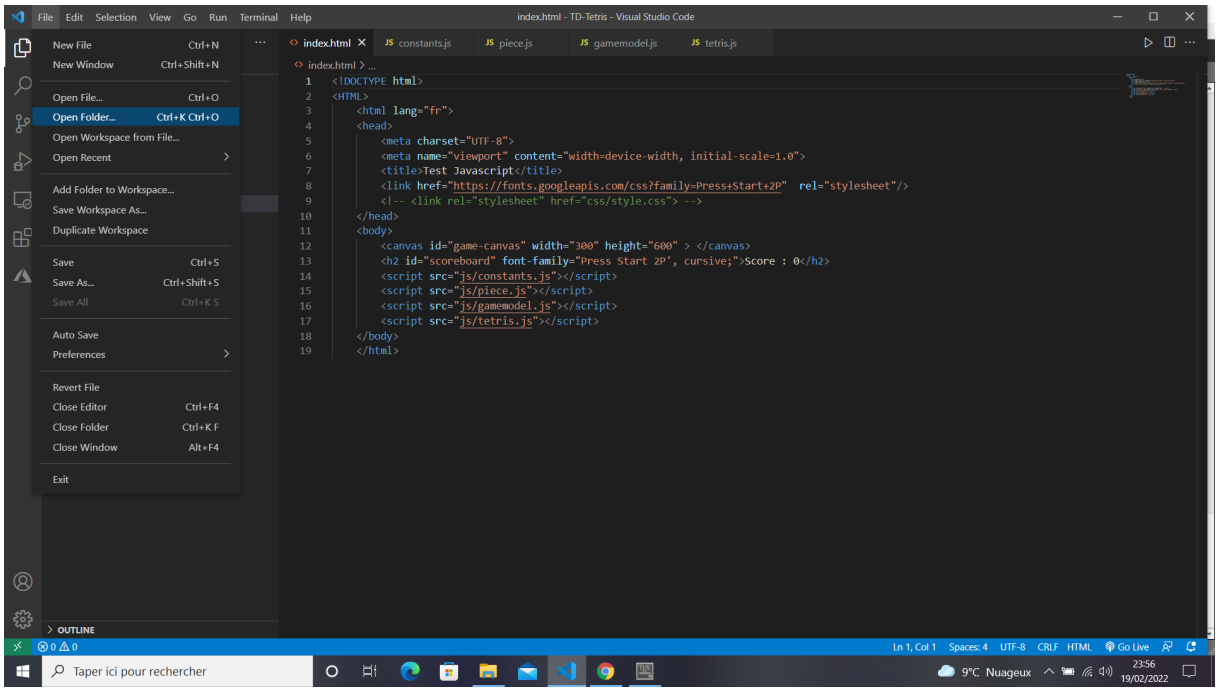


Figure 3: Menu Visual Studio Code

Vous verrez alors une fenêtre comme celle-ci s'ouvrir :
Nous allons compléter dans TD les fichiers du projet pour créer notre jeu de TETRIS !

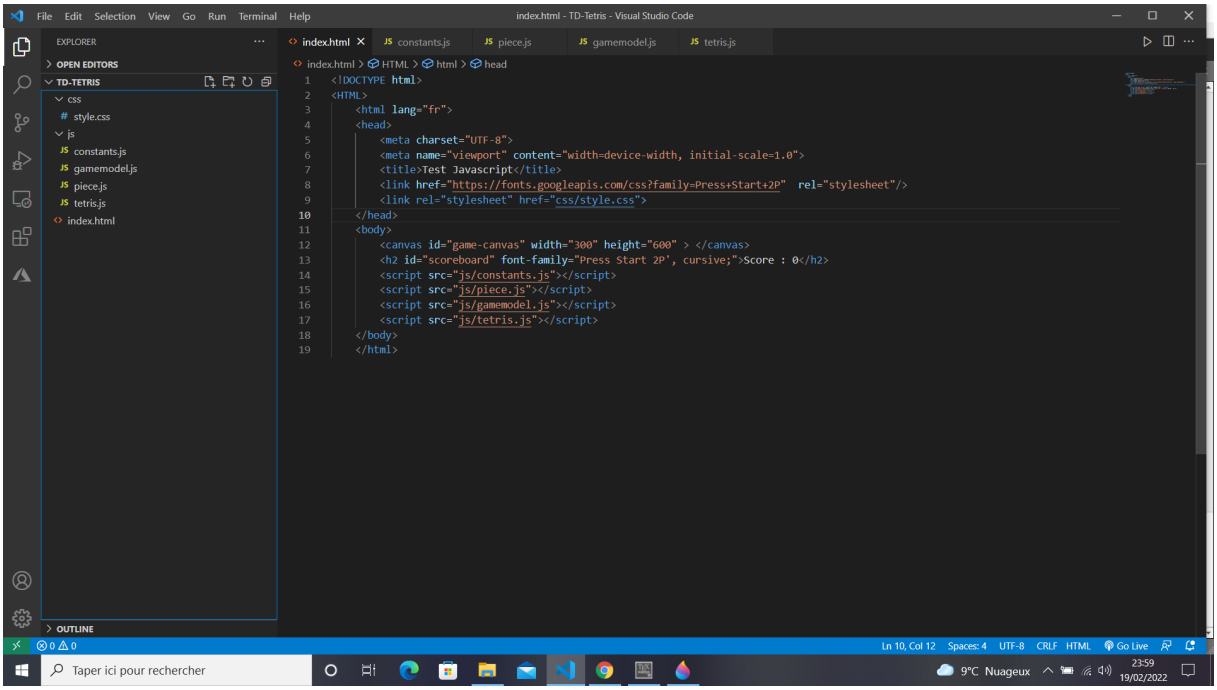


Figure 4: Arboréscence du projet

Jeu de TETRIS simple

constants.js

Dans ce fichier on va déclarer les constantes les plus importantes au projet. Ces constantes comprennent par exemple la taille des blocs des tétraminos, ou le nombre de colonnes du tableau.

1) Définissez les constantes suivantes :

- GAME_CLOCK l'intervalle de temps entre 2 tics.
- ROWS le nombre de lignes du tableau.
- COLS le nombre de colonnes du tableau.
- BLOCK_SIDE_LENGTH la taille du côté d'un bloc de tétramino.
- SCORE_WORTH le nombre de points gagnés par le joueur en remplissant une ligne complète du tableau.

On va maintenant définir les différentes formes que peuvent prendre les tétraminos (qui sont fixes). Pour cela, il faut comprendre la structure du tableau :

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0
0	4	4	0	0	2	0	0	1	0
0	4	4	0	0	2	0	0	1	0
0	6	7	7	2	2	5	5	1	3
6	6	6	7	7	5	5	3	3	3

Le tableau est composé de 20 lignes, et de 10 colonnes. Chacune de ses cellules contient soit un zéro (cellule vide) soit un entier entre 1 et 7 (cellule remplie). A chacun de ces entiers est associé une couleur, correspondant au tétramino représenté (donc cyan pour un tétramino-I par exemple, et noir pour le vide).

2) Construisez la constante SHAPES, qui contient les différents patrons de tétraminos existants, y compris le tétramino *vide* qui ne contient aucun bloc. Cette constante sera un array de matrices carrées. Chaque matrice carrée contient un patron de tétramino existant, représenté comme sur la figure ci-dessus, avec les mêmes valeurs d'entiers pour les cellules des tétraminos.

3) Construisez la constante COLORS qui associe à son index la couleur du tétramino correspondant (ex : index = 1 => Tétramino I => Couleur cyan). Cette constante sera un array de taille 8 contenant des couleurs. Attention, les couleurs en HTML/javaScript sont définies en héxadécimal.

Maintenant que nos constantes sont prêtes, on peut définir le comportement global des tétraminos dans notre jeu de TETRIS.

piece.js

Ce fichier contient une classe Piece qui va définir le comportement des tétraminos dans le projet. Les attributs de la classe seront :

- shape, la forme du patron adopté par le tétramino.
- ctx, le contexte de la page sur laquelle se trouve le tétramino.
- Les positions x et y du tétramino sur le canvas (origine placée au sommet supérieur gauche du tableau).

Cette classe contient un constructeur, et une méthode renderPiece() qui permet de dessiner les tétraminos sur le canvas tout au cours de la partie de TETRIS.

4) Définissez la méthode constructeur de cette classe Piece en prenant comme argument le contexte de la page, et la forme du patron adoptée par le tétramino. On initialisera l'ordonnée à 0 et l'abscisse à la moitié de la largeur du tableau (arrondie à l'entier le plus proche).

La fonction Array.map(function) retourne une copie d'un array, dans laquelle on applique la fonction donnée en argument à tous les éléments non-vides, sans modifier l'array d'origine. Par exemple :

$$const\ numbers = [4, 9, 16, 25]; \tag{1}$$
$$const\ newArr = numbers.map(Math.sqrt) \tag{2}$$
$$alert(newArr) -> \ [2, 3, 4, 5] \tag{3}$$

5) Complétez le code de la méthode renderPiece de manière à colorier de la bonne couleur les tétraminos sur le canvas. Vous utiliserez pour ça les méthodes Array.fillRect et Array.fillStyle. On précise que dans la classe tetris.js, comme nous le verrons plus tard, le contexte de la page a été échelonné à la taille des blocs de tétraminos, donc (x,y) = (1,2) correspond à (BLOC_SIDE_LENGTH, 2*BLOC_SIDE_LENGTH) sur nos écrans.

gameModel.js

Ce fichier contient la classe GameModel qui sert à définir le fonctionnement global du jeu de TETRIS. Les attributs de la classe sont :

- fallingPiece, le tétramino tombant dans le canvas, qui sera null initialement.
- ctx, le contexte de la page sur laquelle se trouve le tétramino.
- grid, le tableau du canvas représentant les cases occupées et vides.
- score, le score obtenu par le joueur, initialement nul (score=0).

La classe contient également plusieurs méthodes :

- Un constructeur.
- makeStartingGrid(), une méthode qui va initialiser le tableau du canvas au début de la partie, en le remplissant de zéros.
- collision(x, y, candidate=null), une méthode qui retourne un booléen : vrai si la pièce candidate entre en collision avec les parois du tableau ou un tétramino déjà tombé à la position (x,y) du tableau, faux sinon.
- renderGameState() qui dessiner sur le canvas tous les éléments qui s'y trouvent (espaces vides, tétraminos tombés, et téramino tombant).
- moveDown(), une fonction qui sera appelée à chaque tic. Si il n'y a pas de tétramino tombant, elle appelera juste la méthode renderGameState() avant de finir. Sinon, si elle testera s'il y a collision au prochain tic pour le téramino tombant. Si oui, elle bloquera en dessinant le tétramino sur le tableau, et testera si le joueur est en *game over*, c'est à dire si la pièce entre en collision en haut du tableau, en ordonnée y=0. Si oui, elle fera apparaître un message d'alerte, et réinitialisera le tableau du canvas pour recommencer une partie. Si il n'y a pas de collision, elle fera simplement descendre d'un cran vers le bas le tétramino (Attention les y sont croissants quand le tétramino tombe).

6) Définissez le constructeur de la classe.

7) En partant d'une array vide, construisez le tableau d'origine grid utilisé par le jeu dans la méthode makeStartingGrid(), soit un tableau de zéros de dimensions (COLS, ROWS).

8) Par analogie avec la question 5, complétez la méthode renderGameState() pour colorer à la fois le tétramino tombant de la bonne couleur (en utilisant la méthode renderPiece() préalablement définie), les tétraminos tombés et les espaces vides.

La méthode collision(x, y, candidate = null) est déjà définie et commentée dans la classe, car elle est quelque peu complexe. Analysez-la bien, pour pouvoir réaliser la question suivante.

9) Complétez la méthode moveDown(). La boucle if que vous devez compléter gère ce qu'il se passe si jamais au tic suivant, en tombant, il y aurait collision entre le tétramino tombant et quelque chose. Vous devez donc figer la position du tétramino tombant sur la grille. Indice : utilisez à nouveau les méthode Array.map.

tetris.js

Ce fichier contient des méthodes qui décrivent le comportement du jeu. Les méthodes en question sont :

- allField(row) qui teste si la ligne du tableau row est entièrement remplie ou non, et retourne un booléen en fonction.
- fullSend() qui va regarder toutes les lignes du tableau, et s'il trouve une ligne remplie, la supprime, insère une ligne vide au tableau (en position y = 0), et ajoute au score de la partie le nombre de points correspondant.
- newGameState(), la fonction qui va s'appliquer à chaque tic et faire descendre les tétraminos sur le canvas, et en créer un avant de la faire descendre si le tétramino tombant est nul.

10) Complétez la méthode allField(row).

11) Complétez la méthode fullSend(). Indice : Pensez à utiliser les méthodes Array.splice et Array.unshift.

12) Complétez la méthode newGameState(). Pour la création d'une nouvelle fallingPiece, on pensera à générer un patron de forme de tétramino aléatoire parmi ceux de SHAPES.

Une fois cette question effectuée, vous pouvez lancer votre projet. Normalement vous verrez des tétraminos tomber depuis le haut du canvas, à la même position en abscisse, jusqu'à atteindre le haut du tableau, avant qu'un message de *GAME OVER* n'apparaisse. Le jeu n'est pas complet, car le joueur ne peut ni faire tourner le tétramino, ni le déplacer. On va rectifier ça maintenant.

Rotations et contrôles clavier

On va ajouter des contrôles au jeu maintenant :

13) Dans la classe GameModel, ajoutez la méthode move(right) où right est un booleen. Si right vaut vrai, le tétramino est déplacé vers la droite, et inversement.

La méthode rotate() est déjà implémentée car assez complexe (basée sur des transposées de matrices). Analysez-la bien au cas où.

La méthode document.addEventListener permet de définir des contrôles clavier/souris sur votre page web. Décommentez le bout de code en fin de fichier tetris.js comprenant cette méthode.

14) Complétez le switch case défini dans cette méthode afin que :

- Si le joueur appuie sur la touche *d*, le tétramino tombant se déplace vers la droite.
- Idem avec la touche *q* pour la gauche.
- Si le joueur appuie sur la touche *s*, le tétramino tombe plus vite.
- Si le joueur appuie sur la touche *z*, le tétramino tourne sur lui-même.

Et voilà ! Vous avez un projet de jeu TETRIS simple complet ! Félicitations !

Améliorations (si vous avez du temps ou que vous voulez flex)

Bonus 1) Ajoutez la musique de tétris en fond sur la page pendant la partie, en boucle, avec un boutton pour activer/désactiver le son sur la page. Si vous êtes motivés, ajoutez des bruitages en cas de collision/rotation du tétramino tombant.

Bonus 2) Mettez en place des niveaux : Plus le score du joueur augmente, plus rapide seront les tics d'actualisation du canvas.

Bonus 3) Bossez le CSS du projet pour améliorer le look de votre jeu de TETRIS.