CONCEPTION ET PROGRAMMATION WEB

PROJET TETRIS

LEA-PLAZA María

ROLLAND Noé

Structure générale du code

Le jeu de Tetris que nous avons créé est construit en utilisant trois langages principaux, HTML qui définit la structure du jeu, CSS qui définit l'apparence visuelle du jeu, et JavaScript qu'implémente la logique, les interactions et les fonctionnalités du jeu.

HTML

Tout d'abord, le document HTML définit le body où le jeu est dessiné et l'audio gère. Pour l'audio nous avons ajouté deux sons, mainTheme pour l'audio pendant le jeu, et gameOver pour l'audio quand le jeu et finit.

Pour dessiner le jeu, nous avons créé trois element :

- Une div qui a comme id : game.
- Un titre (h1) avec un span d'id : score
- Et un canvas avec l'id : tetris

Où nous définissons que le score commence à 0 et que la taille de notre canva sera 300px par 600px.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tetris</title>
        <link rel="stylesheet" href="style.css">
        <script defer src="script.js"></script>
    </head>
    <body>
        <div id="sound">
            <audio id="mainTheme" controls loop>
                <source src="./mainTheme.mp3" type="audio/mpeg">
            </audio>
            <audio id="gameOver">
                <source src="./gameOver.mp3" type="audio/mpeg">
            </audio>
        </div>

        <div id="game">
            <h1> Score : <span id="score">0</span> </h1>
            <canvas id="tetris" width="300" height="600"></canvas>
        </div>

    </body>
</html>
```

CSS

Dans notre document CSS, nous avons définit la conception visuelle du jeu, garantissant une interface claire et attrayante pour l'utilisateur.

1. Le bord de notre tableau a été créé
2. Le couleur de fond violette (blueviolet) a été choisi est définie. Également, la couleur blanche du titre (white).
3. Le canvas est centré sur l'écran.
4. Le box-shadow est utilisé pour donner un effet 3D au plateau.
5. Le titre et la section du jeu sont stylisés.

ProjetTetris > # style.css > #game

```css
* {
    margin: 0px;
    padding: 0px;
    box-sizing: border-box;
}

body {
    display: flex;
    margin: 0;
    background-color: blueviolet;
}

h1 {
    color: white;
    font-size: 2em;
    text-align: center;
}

canvas {
    border: 1px solid black;
    background-color: blanchedalmond;
    box-shadow: 14px 14px 0px 0px blue;
}

#game {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin-top: 0;
}

#sound{
    display: flex;
    justify-content: normal;
    align-items: top;
    margin: 20px;
}
```

Les différentes fonctionnalités implémentées

JavaScript

Toute la logique et les fonctionnalités du jeu ont été définit en JavaScript. Les pièces, le plateau et les interactions de l'utilisateur sont gérés dans ce document.

1. *Définition, initialisation et variables d'état*

Pour commencer, nous avons récupéré le canvas et son contexte pour dessiner le plateau.

```
ProjetTetris > JS script.js > ...
1   const canvas = document.getElementById('tetris');
2   let isGameRunning = false;
3   let timerId;
4   let score = 0;
5
```

Pour initier le jeu nous avons utilisé :

- isGameRunning : Indique si le jeu est en cours ou non.
- timerId : Contiendra l'identifiant du minuteur du jeu.
- score : Stocke le score du joueur.

1.1. *Création et dessin des pièces*

Ensuite, nous avons définit la taille de la grille et nous avons créé une matrice du plateau bidimensionnelle. Dans cette matrice composée par 0 et 1, nous avons définit la forme de chaque pièce ou 1 est un espace occupé par la pièce et la couleur et 0 reste vide.

Ceci est la représentation sous forme de matrice de la pièce I.

```
16  let shape_I = [
17      [0, 1, 0, 0],
18      [0, 1, 0, 0],
19      [0, 1, 0, 0],
20      [0, 1, 0, 0]
21  ];
22
```

Les pièces et les couleurs sont :

- I : 'cyan', J : 'blue', L : 'orange', O : 'yellow', S : 'green', T : 'purple', Z : 'red'

1.2. *Création du plateau de jeu*

La dimension et la structure ont ensuite été définies dans le plateau.

```
68   const grid = 30;
69   const ROWS = canvas.height / grid;
70   const COLS = canvas.width / grid;
71   const board = Array.from({ length: ROWS }, () => Array(COLS).fill(0));
72
```

- grid = 30 : Définit la taille de chaque case en pixels (chaque carré mesure 30x30 px).
- ROWS : Nombre total de lignes du plateau en divisant la hauteur du canvas par la taille d'une case.
- COLS : Nombre total de colonnes en divisant la largeur du canvas par la taille d'une case.
- board : C'est une matrice bidimensionnelle (ROWS x COLS) où chaque case est initialisée à 0, ce qui représente un espace vide.

1.3. *Avant initialiser le jeu*

Pour garantir que le code à l'intérieur de la fonction ne s'exécutera que lorsque le DOM sera complètement chargé, nous avons utilisé la fonction addEventListener (DOMContentLoaded).

Cette fonction s'exécute lorsque le document HTML est entièrement chargé. Elle configure le contexte du canvas, récupère les éléments audios et définit la fonction displayMessage qui affiche un message à l'écran avant l'initialisation du jeu.

```
73
74   document.addEventListener('DOMContentLoaded', () => {
75       const ctx = canvas.getContext('2d');
76       const mainThemeSound = document.getElementById("mainTheme");
77       const gameOverSound = document.getElementById("gameOver");
78
79       const displayMessage = (param) => {
80           ctx.fillStyle = 'white';
81           ctx.globalAlpha = 0.75;
82           ctx.fillRect(0, canvas.height / 2 - 30, canvas.width, 60);
83           ctx.globalAlpha=1;
84           ctx.fillStyle = 'black';
85           ctx.font = '30px monospace';
86           ctx.textAlign = 'center';
87           ctx.textBaseline = 'middle';
88           ctx.fillText(param, canvas.width / 2, canvas.height / 2);
89       }
90       displayMessage("START GAME!");
91
```

1.4. Entrée utilisateur

Après, nous avons créé les fonctions pour détecter les touches pour initialiser le jeu, déplacer les pièces et les faire tourner.

```
    window.addEventListener('keydown', (e) => {
        console.log(e);
        if ((e.code === 'Space' || e.key === ' ') && !isGameRunning) {
            isGameRunning = true;
            newGame();
            updateScore();
            timerId = setInterval(gameLoop, 500);
        };
    });

    document.addEventListener('keydown', (e) => {
        if (isGameRunning) {
            if (e.code === 'ArrowDown') {
                moveDown();
            } else if (e.code === 'ArrowLeft') {
                moveLeft();
            } else if (e.code === 'ArrowRight') {
                moveRight();
            } else if (e.code === 'ArrowUp') {
                rotate();
            }
        }
    });



    document.addEventListener('keydown', (e) => {
        if (e.code === 'Escape') {
            clearInterval(timerId);
            isGameRunning = false;
            displayMessage("PAUSE");
        }
    });
```

- isGamingRunning : En appuyant sur la barre d'espace (Space), le jeu commence (isGameRunning = true).
- newGame() est appelé pour générer une nouvelle pièce et dessiner le plateau.
- Un minuteur (setInterval) est mis en place pour exécuter gameLoop() toutes les 500 millisecondes.

Pour déplacer la pièce, l'utilisateur doit appuyer sur les flèches de son clavier selon l'endroit où il souhaite la déplacer. Enfin, si l'utilisateur appuyez sur la barre d'espace pendant le jeu, le jeu s'arrête avec le message "Pause".

2. *Jeu initialisé*

Pour initialiser le jeu nous avons utilisé les fonctions essentielles pour le correct fonctionnement du jeu :

- newGame(): Démarre une nouvelle partie en générant une nouvelle pièce (newPiece()) et en dessinant le plateau avec cette pièce (draw()).

- newPiece(): Sélectionne une pièce aléatoire parmi la liste des formes (shapes). Définit les propriétés de la nouvelle pièce (currentShape) :
    - shape : La forme de la pièce.
    - type : Le type de pièce (identifiant).
    - x : La position initiale sur l'axe horizontal, calculée pour être centrée.
    - y : La position initiale sur l'axe vertical (en haut de l'écran).

  En plus, cette fonction affiche dans la console la liste des pièces et la nouvelle pièce créée.

- draw(): Si le jeu est en cours (isGameRunning est true) :
    - Dessine le plateau (drawBoard()).
    - Dessine la pièce actuelle (drawShape()) à sa position et couleur correspondantes.

```
//-----------------GAME-----------------//

    function newGame() {
        newPiece();
        draw();
    }



    function newPiece() {
        const pieces = Object.keys(shapes);
        console.log(pieces);
        const piece = pieces[Math.floor(Math.random() * pieces.length)];
        currentShape = {
            shape: shapes[piece],
            type: piece,
            x: Math.floor(COLS / 2) - Math.floor(shapes[piece][0].length / 2),
            y: 0
        };
        console.log(currentShape);
    }



    function draw() {
        if (isGameRunning) {
            drawBoard();
            drawShape(currentShape.shape,
                currentShape.x,
                currentShape.y,
                colors[currentShape.type]);
        }
    }

```

Pourfaire descendre la pièce et augmenter le score nous avons utilisé la fonction gameLoop. Elle est appellée toute les 500 ms.

```
195
196  function gameLoop() {
197      if (isGameRunning) {
198          moveDown();
199          score++;
200          updateScore();
201      }
202  }
203
204
205
206  function updateScore() {
207      document.getElementById('score').innerText = score;
208  }
209
```

Voici la description des fonctions pendant le jeu initialisé :

2.1. Dessiner le jeu

Ces trois fonctions au-dessous permettent de commencer la conception du tableau à partir de zéro, créer les formes sur les axes verticaux (y) et horizontaux (x) et dessiner chaque carré sur le canvas.

```
162  function drawBoard() {
163      ctx.clearRect(0, 0, canvas.width, canvas.height);
164      board.forEach((row, i) => {
165          row.forEach((value, j) => {
166              if (value) {
167                  drawSquare(j, i, colors[value]);
168              }
169          });
170      });
171  }
172
173
174
175  function drawShape(shape, x, y, color) {
176      shape.forEach((row, i) => {
177          row.forEach((value, j) => {
178              if (value) {
179                  drawSquare(x + j, y + i, color);
180              }
181          });
182      });
183  }
184
185
186
187  function drawSquare(x, y, color) {
188      ctx.fillStyle = color;
189      ctx.fillRect(x * grid, y * grid, grid, grid);
190      ctx.strokeStyle = 'black';
191      ctx.strokeRect(x * grid, y * grid, grid, grid);
192  }
193
```

- drawnBoard :cette fonction permmet de créer un nouveau dessin du plateau de jeu.
    - clearRect(): Efface le canvas avant de dessiner.

Nous parcourons le board et, si une case contient une pièce (value différent de 0), on dessine un carré avec sa couleur correspondante.

- drawnShape : permet à chaque pièce d'être affichée correctement sur le plateau de jeu. Elle s'assure que seuls les blocs qui la composent sont visibles à l'écran.
    - forEach() est utilisé pour parcourir chaque ligne (row) avec son index i.
    - Chaque case (value) est parcourue avec son index j à l'intérieur de chaque ligne.
    - On appelle la fonction drawSquare(x + j, y + i, color), qui dessine un carré à la position (x + j, y + i) avec la couleur correspondante.

2.2. Dessiner un carré

- drawSquare : cette fonction permmet créer un carré selon la forme de la pièce.
    - ctx.fillStyle = color : Définit la couleur de la pièce.
    - ctx.fillRect(x * grid, y * grid, grid, grid) : Dessine un carré à la position (x, y) avec la taille de la case (grid x grid).
    - ctx.strokeStyle = 'black' : Définit la couleur du contour.
    - ctx.strokeRect(x * grid, y * grid, grid, grid) : Dessine un contour noir autour du carré.

2.3. Mouvements et collisions des pièces

La fonction moveDown vérifie si une pièce heurte les bords ou d'autres pièces. C'est à dire que cette fonction vérifie si la pièce peut descendre avec canMove(0,1).

```
216     function moveDown() {
217         if (canMove(0, 1)) {
218             currentShape.y++;
219             draw();
220         } else {
221             lock();
222             newPiece();
223             if (!canMove(0, 0)) {
224                 clearInterval(timerId);
225                 isGameRunning = false;
226                 mainThemeSound.pause();
227                 gameOverSound.play();
228                 displayMessage("GAME OVER");
229             }
230         }
231     }
```

Si elle ne heurte les bords ou d'autres pièces, elle descend et on redessine.

Sinon, elle se verrouille (lock()) et une nouvelle pièce est générée.

Si la nouvelle pièce ne peut pas être placée, le jeu se termine et le message « Game Over » est affiché.

2.4. Rotations des pièces

Nous avons décidé de faire tourner les pièces en transposant la matrice bidimensionnelle et en inversant les lignes.

```
247     function rotate() {
248         const rotatedShape = rotateShape(currentShape.shape);
249         if (canMove(0, 0, rotatedShape)) {
250             currentShape.shape = rotatedShape;
251             draw();
252         }
253     }
254
255     function rotateShape(shape) {
256         const N = shape.length;
257         const newShape = Array.from({ length: N }, () => Array(N).fill(0));
258         shape.forEach((row, i) => {
259             row.forEach((value, j) => {
260                 newShape[j][N - 1 - i] = value;
261             });
262         });
263         return newShape;
264     }
265
```

- rotateShape() : cette fonction génère une nouvelle forme tournée.

Si la pièce peut tourner (canMove()), elle est mise à jour et redessinée.

La fonction canMove() verifie si la piece peut bouger.

```
272     function canMove(dx, dy, newShape = currentShape.shape) {
273         for (let i = 0; i < newShape.length; i++) {
274             for (let j = 0; j < newShape[i].length; j++) {
275                 if (newShape[i][j]) {
276                     const x = currentShape.x + j + dx;
277                     const y = currentShape.y + i + dy;
278                     if (x < 0 || x >= COLS || y >= ROWS) {
279                         return false;
280                     }
281                     if (y >= 0 && board[y][x]) {
282                         return false;
283                     }
284                 }
285             }
286         }
287         return true;
288     }
```

Dans ce cas, dx et dy sont la position à regarder, dx droite ou gauche, dy en dessous

2.5. *Actualisation du plateau*

Chaque fois qu'une pièce tombe et se verrouille sur sa position, nous mettons à jour la matrice du plateau (board).

```
284     function lock() {
285         currentShape.shape.forEach((row, i) => {
286             row.forEach((value, j) => {
287                 if (value) {
288                     const x = currentShape.x + j;
289                     const y = currentShape.y + i;
290                     board[y][x] = currentShape.type;
291                 }
292             });
293         });
294         checkRows();
295     }
296
```

- lock() : la fonction verrouille la position de la pièce. Nous parcourons currentShape.shape et obtenons les coordonnées exactes de la pièce sur le plateau.
    - Nous mettons à jour board[y][x] = currentShape.type; pour indiquer que la pièce est verrouillée à cette position.

- checkRows(); vérifie si des lignes complètes doivent être supprimées.

2.6. *Suppression des lignes*

- chechRows : Cette fonction nous aide à supprimer les lignes complètes et à ajouter de nouvelles en haut.

```
297     function checkRows() {
298         for (let i = ROWS - 1; i >= 0; i--) {
299             if (board[i].every((value) => value)) {
300                 console.log(board);
301                 board.splice(i, 1);
302                 board.unshift(Array(COLS).fill(0));
303                 score += 100;
304                 updateScore();
305                 i++;
306             }
307         }
308         draw();
309     }
310 });
```

Nous parcourons le board de bas en haut.

- Si une ligne est pleine (board[i].every(value => value)), nous devons la supprimer.
- board.splice(i, 1) : Supprime la ligne complète.
- board.unshift(Array(COLS).fill(0)) : Ajoute une nouvelle ligne vide en haut.
- On augmente le score et met à jour l'affichage.

- On redessine le plateau (draw())

Avec toutes ces fonctions, nous avons réussi à réaliser le jeu de Tetris résultant en un code fonctionnelle qui permet contrôler des pièces, réaliser les collisions et la suppression des lignes, donner un score à l'utilisateur en fonction de son interaction avec le jeu et finalement lui permettre de se déteindre.

Points importants du projet

Conclusions

Après la réalisation de cet project, nous avons trouvé que les point clés pour son succès étaient les suivants :

1. Générer la relation entre les trois documents HTML, CSS et JS : Cela permet de donner un sens à la structure, au design et à la fonctionnalité. Maintenir un ordre, une synchronisation et un code propre et lisible.

2. Utilisation du canvas : Toute la partie visuelle est gérée avec canvas, permettant de dessiner les pièces et de mettre à jour l'écran rapidement. Elle permet également un rendu efficace du jeu.

3. Gestion des matrices : Celle-ci représente l'état du plateau et des pièces. Le plateau est une matrice ROWS x COLS, ce qui permet une représentation facile des pièces.

4. Détection des collisions : Cette détection empêche les pièces de traverser les bords ou de se superposer. Elle garantit que les pièces ne traversent pas d'autres ou les bords. Avec cette fonction, nous garantissons une bonne expérience utilisateur pendant le jeu.

5. Interaction avec l'utilisateur : Elle permet le déplacement et la rotation des pièces avec le clavier, en fonction des besoins de l'utilisateur.