Tetris - Ontwikkeling basis spel

Workshop @ AP Hogeschool Antwerpen
Opleiding Toegepaste Informatica

20 februari 2019

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

1 Wat is het doel van dit document?

Dit document overloopt de verschillende stappen om een tetris spel te maken dat binnen een webbrowser zal werken. Om dit te realiseren gaan we gebruik maken van HTML en JavaScript. HTML is een soort opmaaktaal, het wordt gebruikt om de structuur van webpagina's te definiëren. JavaScript is een programmeertaal die (meestal) in combinatie van HTML wordt gebruikt om dynamische webpagina's te creëren.

2 Lets Go!

Laten we beginnen met het creëren van een mapje ergens op je computer waarin we het `tetris` project in zullen maken.

Stel in dit mapje een document op met de naam `tetris.html`. Deze kan je gewoon aanmaken met VS Code (Tip: VS Code is een tekstverwerkingsprogramma speciaal gericht op het coderen en kan ons ook beter ondersteunen voor de taken die we moeten uitvoeren, klik op de naam hierboven om VS Code te installeren).

Kopieer het volgende in het bestand:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Tetris</title>
	</head>
	<body>


	</body>
</html>
```

Dit is de structuur van een basis HTML pagina. Door middel van 'tags' zoals `<head>` en `<body>` kunnen we onze pagina structuur opdelen.

`<!DOCTYPE html>`	Specificeer het document type
`<html> … </html>`	Bevat alle document data
`<head> … </head>`	De 'header', bevat data die niet direct zichtbaar gaat zijn in de webpagina
`<title> … </title>`	Titel van de webpagina
`<body> … </body>`	Zichtbare inhoud van de pagina

Voor het spel tetris hebben we een plaats nodig waar we op kunnen tekenen (in ons geval blokjes). Deze definiëren we met de `canvas` tag. Voeg het volgende toe binnen in de body tags:

```
<canvas id="myCanvas" width="640" height="640">
</canvas>
```

We geven nog een paar attributen mee zoals de grootte van het tekenveld en het ID. Het ID is belangrijk omdat we deze nodig hebben om het vanuit JavaScript aan te spreken.

Op dit moment kan je het HTML bestand al met een webbrowser openen maar zal je nog niets zien aangezien dat we nog niets op het veld hebben getekend.

3 Time for JavaScript!

Om iets te kunnen tekenen hebben we JavaScript nodig. JavaScript code kunnen we direct in hetzelfde HTML bestand steken maar het zal overzichtelijker zijn om dit nog in een apart document te steken.

Maak een bestand `tetris.js` aan met de volgende inhoud:

```
function init()
{
      canvas  = document.getElementById('myCanvas');
      context = canvas.getContext('2d');

      context.fillStyle = "black";
      context.fillRect(10, 10, 100, 100);
}
```

Hierin definiëren we een JavaScript functie `init`. Deze haalt eerst een verwijzing op naar het `canvas` element. Omdat we eigenlijk op een `canvas` element ook in 3D kunnen tekenen moeten we expliciet specificeren dat we `2d` willen. Vervolgens zeggen we dat de kleur die we gaan gebruiken zwart is en dat we een vierkant van 100 pixels op 100 pixels willen tekenen op de positie 10, 10. Dit betekent 10 pixels van de linkerkant van het tekenveld en 10 pixels van de bovenkant.

Natuurlijk, dit JavaScript bestand is niet direct gelinkt met ons HTML bestand. We moeten nog enkele dingen binnen in `tetris.html` definiëren om te zorgen dat dit allemaal wordt uitgevoerd.

Eerst en vooral moeten we een referentie specificeren naar het JavaScript document. Dit kan je doen met het volgende stukje HTML:

```
<script type="text/javascript" src="tetris.js"></script>
```

Deze moet je plaatsen binnen in de `head` tags, na de `title` definitie.

Daarna moeten we zeggen dat wanneer de pagina gedaan is met laden dat we de `init` functie gaan oproepen. Verander de `body` tag als volgt:

```
<body onload="init()">
```

Als je `index.html` nu in een webbrowser open zou doen zou je een mooi zwart vierkant moeten zien.

4 Time for tetris!

Nu dat we al wat kunnen tekenen is het tijd dat we eens met het tetris gedeelte beginnen.

Tetris heeft een aantal bepaalde vaste waarden die we best op voorhand al kunnen definiëren (denk aan de hoogte, breedte van het veld, de verschillende blokjes…). Aangezien dat deze waarden redelijk vast liggen en in feite niet veel met code te maken hebben zit kan je deze gegevens op de volgende pagina vinden. Maak hiervoor een nieuw document aan met de naam `tetris_data.js`. Link daarna naar dit document vanuit `tetris.html` door een `script` tag toe te voegen in de header (zoals we met `tetris.js` hadden gedaan). Belangrijk is dat deze referentie gespecificeerd is voor de referentie naar `tetris.js`.

```
var BLOCK_SIZE   = 32;
var FIELD_HEIGHT = 20;
var FIELD_WIDTH  = 10;
var T_FALL_DELAY = 1000;
var T_LINE_DELAY = 50;
var T_START_Y    = -1;
var T_START_X    =  3;

var T_BACK_COLOR  = "BLACK";
var T_BLOCK_COLOR = "red";

var tetrominos = [
                            [
                                  [0, 0, 0, 0],
                                  [1, 1, 1, 1],
                                  [0, 0, 0, 0],
                                  [0, 0, 0, 0]
                            ],
                            [
                                  [1, 1],
                                  [1, 1]
                            ],
                            [
                                  [0, 1, 0],
                                  [1, 1, 1],
                                  [0, 0, 0]
                            ],
                            [
                                  [0, 1, 1],
                                  [1, 1, 0],
                                  [0, 0, 0]
                            ],
                            [
                                  [1, 1, 0],
                                  [0, 1, 1],
                                  [0, 0, 0]
                            ],
                            [
                                  [1, 0, 0],
                                  [1, 1, 1],
                                  [0, 0, 0]
                            ],
                            [
                                  [0, 0, 1],
                                  [1, 1, 1],
                                  [0, 0, 0]
                            ]
                      ];

var KEY_LEFT  = 37;
var KEY_UP    = 38;
var KEY_RIGHT = 39;
var KEY_DOWN  = 40;
```

5 Arrays en multidimensionale arrays

Zoals je in `tetris_data.js` kan zien, hebben we een groot gedeelte met vierkante haakjes en de getallen 0 en 1. Hierin specificeren we in feite onze tetris blokjes. Een 0 stelt geen blokje voor en een 1 wel. Door deze in lijsten te organiseren kunnen we ze gemakkelijk in onze code te gebruiken.

In JavaScript noemen we lijsten zoals deze arrays. In feite is een array gewoon een verzameling van gegevens die numeriek zijn geïndexeerd. Je zou bijvoorbeeld een array met enkele getallen op de volgende manier kunnen definiëren:

```
var mijnArray = [5, 6, 1, 9, 10, 1];
```

Om er iets uit te halen, zou je het volgende gebruiken:

```
mijnArray[0]
```

Dit haalt het cijfer op dat op index 0 staat (het eerste cijfer, in ons geval 5). Het is zeer belangrijk om te weten dat in JavaScript (en de meeste andere programeertalen) dat we met 0 beginnen om lijsten en andere gegevensstructuren te indexeren.

We kunnen ook gegevens van andere typen in arrays steken. Het is zelfs mogelijk om een array binnen in een array te steken waardoor we in feite multidimensionale structuren kunnen krijgen. Deze gebruiken we om al onze verschillende tetris blokjes op te slaan en gaan we ook gebruiken voor ons speelveld.

6 Het speelveld

Bovenaan in tetris.js gaan we nu ons speelveld definiëren.

```
var t_field = [];
```

Op het moment zit er nog niets in, maar we gaan een functie opstellen om deze op te vullen met allemaal array-tjes en getallen. In feite, uiteindelijk zal het veld in het geheugen ongeveer als volgt uitzien.

[
	[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, ...],
	....
]

De functie om dit op te stellen ziet er als volgt uit:

```
function clearField(){
     for (var y = 0; y < FIELD_HEIGHT; y++)
     {
          t_field[y] = [];
          for (var x = 0; x < FIELD_WIDTH; x++)
          {
               t_field[y][x] = 0;
          }
     }
}
```

Op eerste zicht ziet dit er misschien een beetje complex uit, maar het valt wel mee. Zoals je kan zien maken we gebruik van `for`-structuren. Deze worden gebruikt om stukken code meerdere keren uit te voeren.

```
for (var y = 0; y < FIELD_HEIGHT; y++)
```

In het Nederlands staat hier:

Stel `y` gelijk aan 0 en zolang `y` kleiner is dan de constante `FIELD_HEIGHT` voer je de code uit tussen de haakjes en verhoog je `y` met 1.

De code hier gaat dus `FIELD_HEIGHT` aantal keer worden uitgevoerd.

Het eerste wat we doen is opnieuw een lege array creëren die we in het veld gaan plaatsen op index `y`. Daarna hebben we een nieuwe `for`-loop die voor `FIELD_WIDTH` keer het getal 0 is deze nieuwe array plaatst.

Vervolgens moeten wij nu code schrijven om het veld te tekenen.

Laten we beginnen met een functie om één enkele vierkantje te tekenen:

```
function drawBlock(x, y, color)
{
     context.fillStyle = color;
     context.fillRect(x * BLOCK_SIZE, y * BLOCK_SIZE,
BLOCK_SIZE, BLOCK_SIZE);
}
```

Als we deze functie een positie en een kleur meegeven zal het voor ons een mooi blokje tekenen.

Code om heel het veld te tekenen ziet er als volgt uit:

```
function drawField()
{
     for (var y = 0; y < FIELD_HEIGHT; y++)
     {
          for (var x = 0; x < FIELD_WIDTH; x++)
          {
               var color;
               if (t_field[y][x] == 1)
                    color = T_BLOCK_COLOR;
               else
                    color = T_BACK_COLOR;

               drawBlock(x, y, color);
          }
     }
}
```

Deze code is vergelijkbaar met de code om het veld leeg te maken met nulletjes. We hebben ook twee `for`-loops maar in plaats van een waarde in te stellen gaan we het ophalen en op basis van de inhoud gaan we een rood blokje of een zwart blokje tekenen.

Als we nu de code in `init()` vervangen zodat we, in plaats van een vierkantje te tekenen van 100 op 100, een aanroep doen naar de functies `clearField` en `drawField`, dan zul je zien dat we nu een groot zwart veld hebben met de dimensies van een tetris spel.

7 Tetrominos Go!

Nu dat het veld klaar is voor gebruik moeten we beginnen met onze blokjes (tetrominos).

Binnen in onze code hebben we tetrominos gedefinieerd als tweedimensionale arrays. De functie de we nodig hebben om eentje te tekenen verschilt niet veel van de functie die we gebruiken om het veld te tekenen:

```
function drawTetromino(tetromino, xx, yy)
{
     for (var y=0; y<tetromino.length; y++)
     {
          for (var x=0; x<tetromino.length; x++)
          {
               if (tetromino[y][x] == 1)
                    drawBlock(xx+x, yy+y, T_BLOCK_COLOR);
          }
     }
}
```

Hier moeten we de tetromino positie en kleur meegeven.

Wat moeten we ook nog met een tetromino kunnen doen? Draaien natuurlijk! Hiervoor gaan we ook een functie schrijven.

```
function rotateTetromino(tetromino)
{
     var tetromino_out = [];

     for (var x=0; x<tetromino.length; x++)
     {
          tetromino_out[x] = [];
          for (var y=0; y<tetromino.length; y++)
          {
               tetromino_out[x][tetromino.length-y-1] =
tetromino[y][x];
          }
     }

     return tetromino_out;
}
```

Deze functie maakt eigenlijk een nieuwe tetromino aan die de gedraaide versie is.

Een tetromino moet vast komen zitten als het iets aanraakt. Hiervoor gaan we een eerst een functie schrijven om te kijken of er wel iets is op een welbepaalde plaats:

```
function getFieldBlock(x, y) {
     if (x < 0 || x >= FIELD_WIDTH || y >= FIELD_HEIGHT)
          return 1;
     else if (y < 0)
          return 0;
     else
          return t_field[y][x];
}
```

Hier geven we 1 terug als we in het veld zitten en er iets op die locatie te vinden is, 1 als we links, rechts of naast het veld zitten (hiermee 'simuleren' we randen) en 0 als we ín het veld zitten en er niks op die locatie is of wanneer we boven het speelveld zitten.

De functie om te checken of het tetrimino blokje iets aanraakt ziet er dan als volgt uit:

```
function checkCollision(tetromino, xx, yy)
{
     for (var x=0; x<tetromino.length; x++)
     {
          for (var y=0; y<tetromino.length; y++)
          {
               if (tetromino[y][x] == 1 && getFieldBlock(xx+x,
yy+y) == 1)
                    return true;
          }
     }

     return false;
}
```

Als er een aanraking zou zijn, dan gaan we data van het tetromino blokje kopiëren naar het speelveld. Hiervoor gaan we ook een functie aanmaken.

```
function placeTetromino(tetromino, xx, yy)
{
     for (var x=0; x<tetromino.length; x++)
     {
          for (var y=0; y<tetromino.length; y++)
          {
               if (tetromino[y][x] && yy + y >= 0)
                    t_field[yy+y][xx+x] = 1;
          }
     }
}
```

Het volgende wat we moeten doen is code schrijven om een nieuw blokje op het veld te plaatsen.

```
function randomType()
{
	return Math.floor(Math.random() * tetrominos.length);
}

function placeNewTetromino()
{
	t_type		= randomType();
	t_block		= tetrominos[t_type];

	t_x = T_START_X;
	t_y = T_START_Y;
}
```

`randomType` geeft een getal terug dat we gaan gebruiken om een nieuw tetromino blokje te kiezen. `placeNewTetrimino` gaat dan de juiste tetromino ophalen.

Na dat we dit allemaal hebben gedaan kunnen we beginnen met de functies te linken.

```
function moveDown()
{
	if (checkCollision(t_block, t_x, t_y + 1))
	{
		placeTetromino(t_block, t_x, t_y);
		placeNewTetromino();

		if (checkCollision(t_block, t_x, t_y))
			gameOver();
	}
	else
	{
		t_y++;
	}

	drawAll();
}
function drawAll()
{
	drawField();
	drawTetromino(t_block, t_x, t_y);
}
function newGame()
{
	clearField();
	placeNewTetromino();

	t_timer_down = setInterval(moveDown, T_FALL_DELAY); //Roep periodiek
moveDown op

	drawAll();
}
```

Goed, nu vallen de blokjes, maar we kunnen zo nog niet bewegen. Om ervoor te zorgen dat dit wel zo is moeten we een 'event' instellen die een functie aanroept en deze verwerkt.

Binnen in onze init functie voegen we het volgende lijntje toe

```
window.addEventListener('keydown', checkKey, false);
```

Vervolgens creëren we de functie checkKey:

```
function checkKey(e)
{
    var code = e.keyCode;

     if ( code == KEY_LEFT && !checkCollision(t_block, t_x-
1, t_y) )
     {
        t_x--;
    }
    else if( code == KEY_RIGHT && !checkCollision(t_block,
t_x+1, t_y) )
     {
        t_x++;
    }
     else if( code == KEY_UP )
     {
          var t = rotateTetromino(t_block);
          if (!checkCollision(t, t_x, t_y))
               t_block = t;
     }
    else if ( code == KEY_DOWN )
     {
          moveDown();
    }

     drawAll();
}
```

Deze checkt welke toets is ingedrukt en zal dan de tetromino verplaatsen naar de gewenste locatie.

De volgende stap is een check toevoegen om volledige lijnen te verwijderen.

```
function isFullLine(y)
{
     return t_field[y].indexOf(0) == -1;
}

function checkLine()
{
     var removed = false;
     var y = 0;

     while( y < FIELD_HEIGHT  && !removed)
     {

          if (isFullLine(y))
          {
               var newTopLine = [];
               for (x = 0; x < FIELD_WIDTH; x++)
               {
                    newTopLine[x] = 0;
               }

               t_field.splice(y, 1);
               t_field.unshift(newTopLine);

               drawAll();
               removed = true;
          }

          y++;
     }
}
```

Deze functie moeten we ook dan periodiek oproepen. Dit doen we door het volgende lijntje in newGame() toe te voegen.

```
     t_timer_line = setInterval(checkLine, T_LINE_DELAY);
```

En als laatst een ‘game over’ functie die alles stopt:

```
function gameOver()
{
     clearInterval(t_timer_down);
     clearInterval(t_timer_line);

     alert("Game Over");
}
```