Tetris Chrome Extension

Synopsis over årsprøveprojekt, Programmering

Kasper Rosenberg Zimmermann, 3.C

Vejleder: JCJE

Uddannelse: Odense Tekniske Gymnasium

Dato: fredag, d. 28. februar

Indholdsfortegnelse

[Tetris Chrome Extension 1](#_Toc191662829)

[1. Indledning 3](#_Toc191662830)

[2. Krav til program 3](#_Toc191662831)

[3. Funktionalitet af spil og player input 4](#_Toc191662832)

[4. Udvikling af programmet 4](#_Toc191662833)

[4.1: Tetris Gitter (Grid) 4](#_Toc191662834)

[4.2: Faldende stykker og kollision 5](#_Toc191662835)

[4.3: Spiller Input 6](#_Toc191662836)

[Logbog 8](#_Toc191662837)

# 1. Indledning

Google Chrome er den mest brugte webbrowser i 2024 og holder omkring 2/3 af markedet[[1]](#footnote-1). En af Google Chromes features er Chrome Extensions. Normalt kan extensions installeres i chrome webshop[[2]](#footnote-2), men google chrome har en udviklertilstand til extensions og guides på hvordan man kan lave og køre sine egne extensions[[3]](#footnote-3). Denne opgave viser, hvordan jeg har programmet et web tetris spil ved brug af JavaScript og html, som kan køre som en chrome extension.

# 2. Krav til program

* Spillet skal kunne køres på Google Chrome Extensions.
  + Skrevet udelukkende i JavaScript & HTML·
* Spillet skal have funktionalitet, som et normalt tetris spil.
  + Tetris grid med et faldende stykke, som kan kontrolleres af spillerinput
  + Kollision, som tjekker om det faldende stykker kolliderer med noget og derfor skal placeres.
  + Udfyldte linjer cleares.
  + Score-system baseret på antal lines, der bliver clearet.
  + Lokalt gemt high score.
* Spillet skal gemmes lokalt på computeren når det bliver lukket, så man ikke skal starte forfra når det åbnes igen.

|  |
| --- |
| Et billede, der indeholder tekst, skærmbillede, Rektangel, linje/række  Indhold genereret af kunstig intelligens kan være forkert. |
| Figur et billede fra spillet. Commit: ”Saving The Grid” |

# 3. Funktionalitet af spil og player input

Spillet kan tilgås, ved at klikke ind på extensions i Google Chrome, og derefter at trykke på ”Tetrist”.

Når spillet er åbent, så kan brættet ses. Når spillet åbner første gang, så er brættet helt tomt og et grønt stykke falder ned fra loftet.

På figur 1 ses et eksempel af spillet, som er i gang. Nede i bunden af figuren, kan et sort stykke ses. Dette stykke er placeret på brættet og rykker sig ikke. Oppe i toppen kan et grønt stykke ses, som spilleren kan kontrollere. For at komme af med de sorte kasser, skal spilleren fylde en række ud og dermed lave en ’line clear’.

For at fylde rækkerne ud har tetris spil en række af forskellige figurer, som falder ned fra toppen. For at gøre det mere tydeligt har jeg farvet de faldende stykker grønne. Der kommer et faldende stykke ad gangen, og dette stykke kan bevæges af spilleren med tastaturet.

Her er en samling af de mulige keypresses i spillet:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Tastkode** | **Funktion** | | ’w’ og ’ArrowUp’ | Roter faldende stykke | | ’a’ og ’ArrowLeft’ | Ryk faldende stykke en tak til venstre | | ’d’ og ’ArrowRight’ | Ryk faldende stykke en tak til højre | | ’s’ og ’ArrowDown’ | Ryk faldende stykke hurtigere ned | | ’d’ og space (’ ’) | ”Hard Drop”, ryk stykket ned indtil det støder på noget | |
| Tabel |

Så snart stykket har en kasse under sig, så bliver stykket sort og låst fast i gitteret der hvor det er og et nyt faldende stykke spawner oppe i toppen af gitteret.

Yderligere, så når der er fulde rækker, så tilføjes der point til spilleren og når spilleren slår sin highscore og restarter spillet, så gemmes den lokalt, så den ikke bliver væk. Yderligere, så gemmes gitteret og scoren, for at man kan lukke og åbne spillet uden at miste fremskridt.

Når spillet bliver åbnet, så kigger programmet på om der er gemt noget i ”localStorage”, med tagget ”grid”. Hvis der er, så loades det gemte grid. Ellers laves der et nyt 10x20 grid fyldt med 0’er (tomme celler).

# 4. Udvikling af programmet

## 4.1: Tetris Gitter (Grid)

Et standard tetris gitter er 10 bred og 20 høj, så det er også det jeg har valgt at lave det her.

Først og fremmest har jeg lavet en Board class. Her er der en constructor, som jeg giver argumenterne rows og cols for at definere størrelsen af gitteret. Oprindeligt bliver gitteret oprettet ved:

|  |  |
| --- | --- |
| 1 | Array.from({length:rows}, () => Array(cols).fill(0)); |
|  | Tabel |

Her laves et grid med størrelsen (x: rows, y: cols), hvor alle celler er fyldt med 0.

* *Array.from()* opretter et nyt array med *rows* antal elementer. *{length: rows}* er et objekt, som angiver længden på arrayet, så antal rækker.
* *Array(cols)* laver et array med cols antal elementer. Dette bliver gjort for hver række.
* *.fill(0)* fylder alle pladser i de arrays med *0*.

I min board class har jeg en draw funktion, som er i stand til at farve bestemte celler på gitteret.

## 4.2: Faldende stykker og kollision

Der skal laves et objekt med relevante attributter, såsom: position, form, rotation.

For at vise en kasse på skærmen, så kan der under gitteret tegnes en firkant med den farve cellen skal være.

|  |
| --- |
| Et billede, der indeholder Rektangel, jackstik  Indhold genereret af kunstig intelligens kan være forkert. |
| Figur en faldende kasse fra Commit: ”Falling Box”. |

For at tegne den, så er der inde i main.js. Her er der en ”*window.addEventListener(’load’, function() {}”*, som betyder at koden inden for ”*{}”* kører når vinduet bliver loadet. Der bliver her defineret at farven, som bliver tegnet med er sort. Sidst bliver funktionen ”*animate()”* defineret. Den får et interval på 500ms. Med ”*this.setInterval(animate, 500);”.* Funktionen bliver dermed kørt 2 gange i sekundet. I animate bliver alle tidligere tegnede kasser fjernet fra skærmen, og der bliver derefter kørt 2 funktioner fra piece.js: *”Piece.draw()”* og *”Piece.update()”*, som tegner stykket og derefter rykker stykket ned en plads.

Senere i programmet, så skiftede jeg denne enkelte kasse ud med hele stykker.

Stykket ender med at have en global x- og y-koordinat. Og baseret på hvilket stykke man har og rotationen, så beregnes hvor stykket er på en række segmenter, som er illustreret her i tabellen under med i I-stykke:

| **Rotation** | **Koordinat Offsets** | **Visuel Illustration** |
| --- | --- | --- |
| 0 | {x: -1, y: 0}, {x: 0, y: 0}, {x: 1, y: 0}, {x: 2, y: 0} | 🟦 🟦 🟦 🟦 |
| 1 | {x: 0, y: -1}, {x: 0, y: 0}, {x: 0, y: 1}, {x: 0, y: 2} | 🟦 🟦 🟦 🟦 |
| 2 | {x: -2, y: 0}, {x: -1, y: 0}, {x: 0, y: 0}, {x: 1, y: 0} | 🟦 🟦 🟦 🟦 (spejlvendt) |
| 3 | {x: 0, y: -2}, {x: 0, y: -1}, {x: 0, y: 0}, {x: 0, y: 1} | 🟦 🟦 🟦 🟦 (spejlvendt) |

Tabel 3

Kollisions skal her tjekkes for hvert segment individuelt ved brug af et loop. Samme logik, som beskrevet ved den enkelte kasse benyttes for hvert segment.

## 4.3: Spiller Input

Som tidligere nævnt, så har spilleren en række input at benytte sig af:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Tastkode** | **Funktion** | | ’w’ og ’ArrowUp’ | Roter faldende stykke | | ’a’ og ’ArrowLeft’ | Ryk faldende stykke en tak til venstre | | ’d’ og ’ArrowRight’ | Ryk faldende stykke en tak til højre | | ’s’ og ’ArrowDown’ | Ryk faldende stykke hurtigere ned | | ’d’ og space (’ ’) | ”Hard Drop”, ryk stykket ned indtil det støder på noget | |
| Tabel 4 |

Det er under main.js-scriptet af inputs bliver håndteret. Der bliver tjekket for keydown-events med en eventlistener-funktion, og en funktion handleInput(), der bliver kørt hvert gameloop, for at tjekke om der har været inputs. Hvis der har været et input, så bliver en funktion kaldet, som kan udføre den funktion, som knappen skulle. For eksempel: currentPiece.rotateLeft(board);, som roterer stykket. Et problem med denne strategi er at gameloopet kører mange mange gange i sekundet. Derfor er det næsten umuligt at undgå at et meget hurtigt klik med højre piltast ikke bare rykker stykket et felt, men derimod hele vejen over til højre. For at modvirke dette, så har jeg en række timer-variabler, som sikrer at spillet kun accepterer keypresses med et større mellemrum mellem, som gør at spillet virker mere, som det originale tetrisspil og at der er mulighed for at kontrollere stykket, som man vil:

1. // Rotate (Throttled)

2. if (now - lastRotateTime > rotateCooldown) {

3. if (keys['ArrowUp'] || keys['w']) {

4. currentPiece.rotateLeft(board);

5. lastRotateTime = now;

6. }

7. }

8.

Kodeudsnit 1

## 4.4 Scoresystem

Først implementerede jeg et scoresystem. Jeg har valgt at tilføje til spillerens score baseret på line clears. Når spilleren dropper et stykke, så kan spilleren få et antal line clears på: 0, 1, 2, 3 og 4. Inde i min board class har jeg blandt andet følgende kode:

1. clearRows() {

2. // Filter out full rows

3. let newGrid = this.grid.filter(row => row.some(cell => cell === 0));

4. let rowsCleared = this.rows - newGrid.length;

5.

6. // Add empty rows to the top

7. while (newGrid.length < this.rows) {

8. newGrid.unshift(new Array(this.cols).fill(0));

9. }

10.

11. this.grid = newGrid;

12.

13. if (rowsCleared > 0) {

14. this.updateScore(rowsCleared);

15. console.log(`Cleared ${rowsCleared} row(s)!`);

16. }

17. }

18.

19. updateScore(rowsCleared) {

20. const points = [0, 100, 300, 500, 800]

21. this.score += points[rowsCleared] || (rowsCleared \* 200);

22.

23. // update the score html

24. document.getElementById('score').innerText = `Score: ${this.score}`;

25. }

Kodeudsnit 2

Clear rows bliver altid kaldt når et stykke bliver placeret på brættet. Her tjekkes antallet af rows, som bliver clearet og det kommer så ind som argument i updateScore funktionen, som så adderer et antal point baseret på antallet af rækker, som blev clearet.

# Konklusion

Målet med projektet var at udvikle et tetris spil, som kunne fungere som en chrome extension. Synopsen har gennemgået funktionaliteten af spillet og der er udvalgt et par væsentlige funktionaliteter i spillet, som er blevet dokumenteret. Spillet kunne godt forbedres på en række punkter, præcis hvad er nævnt i bunden af README-filen i projektmappen.

# Logbog

26. januar:

* Undersøge js, CSS og html. (Grids)

27. januar

* Se video: <https://www.youtube.com/watch?v=TBGEt_-rFSs&ab_channel=Frankslaboratory>
* Jeg har set hele videoen og besluttet mig for at bruge de samme koncepter til at programmere ”Tetrist”.

28. januar og fremad:

* Programmering, programmering og mere programmering, rapportskrivning og det var sådan set det.

Kilder:

<https://developer.chrome.com/docs/extensions/get-started/tutorial/hello-world>

<https://en.wikipedia.org/wiki/Usage_share_of_web_browsers>

<https://chromewebstore.google.com/category/extensions?utm_source=ext_sidebar&hl=da>

Noter:

Hilite.me JavaScript manni.

1. https://en.wikipedia.org/wiki/Usage\_share\_of\_web\_browsers [↑](#footnote-ref-1)
2. https://chromewebstore.google.com/category/extensions?utm\_source=ext\_sidebar&hl=da [↑](#footnote-ref-2)
3. https://developer.chrome.com/docs/extensions/get-started/tutorial/hello-world [↑](#footnote-ref-3)