Snake mit Node.js

Ziel

Wir wollen das Spiel Snake über Node.js nachbauen. Das ist nicht schwer und leicht zu verstehen.

Vorbereitung

Wir legen einen neuen Ordner im Windows-Explorer an. Wo dieser Ordner liegt, ist nicht von Belang, da wir hier nur den NodeJS-Server und keine Datenbank und den XAMPP benötigen. Nennt ihn z.B. `snake`.

Öffnet diesen Ordner im VSCode. Erzeugt dann ein neues Terminal und initialisiert NodeJS mit dem Befehl `npm init`. Dadurch wird eine Datei `package.json` in diesem Ordner erzeugt.

```
npm init
```

Lasst die Standard-Einstellungen bei der Abfrage. Einzig bei `entry point` könnt ihr game.js eintragen, spielt aber für den weiteren Verlauf keine Rolle.

Dann installiert ihr zwei Packeges ansi und keypress mit dem Befehl

```
npm install ansi keypress
```

Startdatei anlegen

Nun legt im Root-Verzeichnis eine Datei start.js an und speichert folgenden Code dort ab:

```
'use strict';
let ansi = require('ansi');

let cursor = ansi(process.stdout);

try {
    // cursor.bg.... setzt die Hintergrundfarbe, so können wir
mit dem Leerzeichen farbige Flächen malen
    cursor.bg.red();
    cursor.goto(5, 5).write(' ');
    cursor.goto(6, 5).write(' ');
    cursor.goto(7, 5).write(' ');
    // mit reset setzt du die Hintergrundfarbe wieder zurück
    cursor.bg.reset();

    // cursor..... setzt die Textfarbe
    cursor.yellow();
    cursor.goto(9, 5).write('MY GAME');
    cursor.reset();

    cursor.bg.red();
    cursor.goto(17, 5).write(' ');
    cursor.goto(18, 5).write(' ');
    cursor.goto(19, 5).write(' ');
    cursor.bg.reset();
} catch (ex) {
    // hier werden Fehler erkannt und ausgegeben
    console.log(ex.toString());
} finally {
    // zum Schluss müssen wir das Spiel beenden
    quitGame();
}

function quitGame() {
    cursor.reset();
    cursor.bg.reset();
    cursor.goto(1, 10);
    process.exit();
}
```

Um zu Testen, ob alles funktioniert, könnt Ihr im Terminal den Befehl `node start.js` aufrufen. Nodemon braucht Ihr dafür nicht, da das Spiel direkt in der Konsole läuft, deswegen ist der `entry point` bei der Konfiguration auch nicht so wichtig. Ihr könnt die Konsolen-Befehle aber auch in der Eingabeaufforderung oder der Windows-Shell ausführen, falls ihr ein eigenes Fenster haben wollt.

Schaut Euch den Code an und versuch zu verstehen, was da abläuft. Durch die Kommentare ist der gar nicht so schwer zu verstehen.

Spielfeld zeichnen

Nun kommt das eigentliche Spiel. Dazu legt ihr wiederum im Root-Ordner eine Datei game.js an und schreibt dort folgenden Code:

```
'use strict';
let ansi = require('ansi');

let cursor = ansi(process.stdout);
let width = 40;
let height = 20;


function drawHorizontalLine(col, row, length) {
    for (let i = 0; i < length; i++) {
        cursor.goto(col + i, row).write(' ');
    }
}

function drawVerticalLine(col, row, length) {
    for (let i = 0; i < length; i++) {
        cursor.goto(col, row + i).write(' ');
    }
}

function quitGame() {
    cursor.reset();
    cursor.bg.reset();
    cursor.goto(1, 10);
    process.exit();
}


try {
    // draw game area
    cursor.bg.grey();
    drawHorizontalLine(1, 1, width);
    drawHorizontalLine(1, height, width);
    drawVerticalLine(1, 1, height);
    drawVerticalLine(width, 1, height);
    cursor.bg.reset();
} catch (ex) {
    console.log(ex.toString());
} finally {
    quitGame();
}
```

Das wird wieder über Leerzeichen gemacht, welche eine graue Hintergrundfarbe erhalten. Um nicht jedes Leerzeichen einzeln zu coden nutzen wir die beiden Funktionen `drawHorizontalLine` und `drawVerticalLine` mit jeweils einer `for`-Schleife.

Die Schlange wird als grüner Punkt dargestellt. Für die Bewegung der Schlange benötigen wir noch ein paar Variablen, die wir am Anfang der Datei hinter die Initialisierung der Variable `height` platzieren. Diese Variablen haben folgende Funktion:

Variable	Wert	Beschreibung
`posX`	-1	nach links bewegen
`posX`	0	Bewegung nach oben oder unten durch `posY`
`posX`	1	nach rechts bewegen
`posY`	-1	nach oben bewegen
`posY`	0	Bewegung nach links oder rechts durch `posX`
`posY`	1	nach unten bewegen

Initialisiert also zunächst die Variablen hinter `let height = 20;`

```
let posX = 0;
let posY = 0;
let dirX = 1;
let dirY = 0;
```

Im darauffolgenden try-Block wird zu Anfang ein neuer Code hinzugefügt, welcher die Konsole bereinigt und der Cursor ausblendet:

```
try {
    // clear output
    process.stdout.write('\x1Bc');
    // hide cursor
    process.stderr.write('\x1B[?25l');
```

Nach der Zeile cursor.bg.reset(); im `try`-Block fügen wir noch den Code für die initiale Position der Schlange und dem Start des Game-Loops ein:

```
    // set initial position of snake
    posX = Math.floor(width / 2);
    posY = Math.floor(height / 2);

    // start game loop
    gameLoop();
```

Den Game-Loop platzieren wir in der gleichnamigen Funktion, welche nun noch hinter der letzten Funktionsdefinition eingefügt wird:

```
function gameLoop() {
    // remove snake at old position
    removeSnake(posX, posY);

    // set new position
    posX = posX + dirX;
    posY = posY + dirY;

    // check new position
    if (posX == 1 || posX == width || posY == 1 || posY == height) {
        cursor.red();
        cursor.bg.white();
        setText(width / 2 - 6, height / 2, "  GAME OVER  ");
        quitGame();
    }

    // draw snake at new position
    drawSnake();

    // call gameLoop
    setTimeout(gameLoop, 500);
}
```

In der `if`-Struktur wird geprüft, ob die Schlange gegen die Spielbegrenzung trifft. Ist das der Fall ist das Spiel beendet. Im Game-Loop werden noch weitere Funktionen aufgerufen, die die Schlange an die alte Position verschieben (`removeSnake()`) und die Schlange zeichnen (`drawSnake()`). Dann wird noch die Funktion `setText()` benötigt, die den `GAME OVER`-Text an die entsprechende Position schreibt. Und zu guter Letzt noch die Funktion `drawPoint()`, die einen farbigen Punkt an eine beliebige Stelle setzt. Diese wird in den beiden Funktionen `drawSnake()` und `removeSnake()` benötigt.

```
function removeSnake() {
    cursor.bg.black();
    drawPoint(posX, posY);
    cursor.bg.reset();
}

function drawSnake() {
    cursor.bg.green();
    drawPoint(posX, posY);
    cursor.bg.reset();
}

function drawPoint(col, row, char) {
    cursor.goto(col, row).write(' ');
}

function setText(col, row, text) {
    cursor.goto(col, row).write(text);
}
```

Auch diese Funktionen bitte hinter die zuletzt hinzugefügten Funktionsdefinitionen definieren.

Wichtig! Wir haben zu Beginn des try-Blocks den Cursor ausgeblendet. Dieser muss in der Funktion quitGame() wieder eingeblendet werden. Fügt die gekennzeichnete Zeile in der Funktion ein.

```
function quitGame() {
    cursor.reset();
    cursor.bg.reset();
    process.stderr.write('\x1B[?25h');
    cursor.goto(1, height + 4);
    process.exit();
}
```

Auf Eingabe des Users reagieren

Um die Schlange mit den Cursortasten steuern zu können, muss auf das keypress-Ereignis reagiert werden. Die erforderliche Codezeile setzen wir vor dem initialen Setzen der Schlangenposition im `try`-Block ein:

```
    // handle key press events
    process.stdin.on('keypress', handleInput);

    // set initial position of snake
    posX = Math.floor(width / 2);
    posY = Math.floor(height / 2);
```

Der Eventhandler referenziert auf eine Funktion handleInput, welche nun definiert werden muss. Diese fügen wir hinter die zuletzt definierte Funktion ein:

```
function handleInput(chunk, key) {
    if (key.name == 'q') {
        quitGame();
    } else if (key.name == 'right') {
        dirX = 1;
        dirY = 0;
    } else if (key.name == 'left') {
        dirX = -1;
        dirY = 0;
    } else if (key.name == 'up') {
        dirX = 0;
        dirY = -1;
    } else if (key.name == 'down') {
        dirX = 0;
        dirY = 1;
    }
}
```

Die Werte, welche für die Eigenschaft `key.name` geprüft werden repräsentieren die Taste q und die vier Cursor- bzw. Pfeiltasten und setzen die neue Richtung der Schlange bzw. beenden das Spiel.

Apfel fressen

Jetzt benötigen wir noch den Apfel, der von der Schlange gefressen werden soll. Dazu setzen wir nach der Initialisierung der Variablen am Anfang des Skriptes noch weitere Variablen mit:

```
let applePosX = 0;
let applePosY = 0;
```

Die nächste Funktion ist für das Zeichnen des Apfels an einer beliebigen, zufällig errechneten Stelle im Spielfeld verantwortlich. Sie wird am Ende der Funktionsdefinitionen eingefügt.

```
function drawApple() {
    applePosX = Math.ceil(Math.random() * (width - 2)) + 1;
    applePosY = Math.ceil(Math.random() * (height - 2)) + 1;

    cursor.bg.red();
    drawPoint(applePosX, applePosY);
    cursor.bg.reset();
}
```

Der Aufruf dieser Funktion geschieht vor dem Aufruf der Funktion gameLoop() im try-Block:

```
    // draw first apple
    drawApple();

    // start game loop
    gameLoop();
```

Bevor in der Game-Loop die Schlange neu gezeichnet wird, muss überprüft werden, ob der Apfel berührt wurde. Ist das der Fall muss ein neuer Apfel gezeichnet werden:

```
    // check if apple is touched
    if (posX == applePosX && posY == applePosY) {
        // draw new apple
        drawApple();
    }

    // draw snake at new position
    drawSnake();
```

Punkte zählen und Geschwindigkeit erhöhen

Um die gefressenen Äpfel zählen zu können und die Geschwindigkeit der Schlange nach jedem gefressenen Apfel erhöhen zu können benötigen wir in unserer Reihe der zu initialisierenden Variablen zwei neue Variablen für die Punkte und die Geschwindigkeit.

```
let points = 0;
let speed = 3; // moves per second
```

Diese Variablen werden in der Game-Loop verändert. Fügt sie in das `if`-Konstrukt ein, in welchem wir prüfen, ob der Apfel getroffen wurde:

```
    // check if apple is touched
    if (posX == applePosX && posY == applePosY) {
        // increase points
        points++;

        // increase speed
        if (speed < 20) {
            speed++;
        }

        // draw new apple
        drawApple();
    }
```

Am Ende der Game-Loop ändern wir die setTimeout-Funktion dahingehend, dass wir die starre Zeit von 500ms durch eine dynamische Zeitangeb ersetzen, indem wir 1000 durch die Geschwindigkeit dividieren.

```
    // call gameLoop
    setTimeout(gameLoop, 1000 / speed);
```

Zuletzt fügen wir noch in der Funktion drawApple() am Ende die Zeilen hinzu, die für die Ausgabe der Punkte und der Geschwindigkeit verantwortlich sind:

```
    setText(1, height + 2, "Points: " + points.toString());
    setText(1, height + 3, "Speed: " + speed.toString());
```

Dann viel Spaß beim Spielen 😉