**Dokumentation: Snake-Spiel mit HTML, CSS, JavaScript**

**1. Planung**

**1.1 Ziel des Projekts**

Ziel ist es, ein klassisches Snake-Spiel mit modernen Webtechnologien (HTML5, CSS3 und JavaScript) umzusetzen. Das Spiel soll vollständig im Browser laufen, responsive gestaltet sein und ein einfaches, aber funktionierendes Spielerlebnis bieten. Die Umsetzung erfolgt objektorientiert, modular und leicht erweiterbar.

**2. Aussehen**

**2.1 Layout**

Die Seite besteht aus:

* einem **Titel** ("Snake Spiel")
* einem **Canvas-Element**, das das Spielfeld darstellt
* einem **Punkteanzeiger** unter dem Spielfeld

Das Design ist schlicht gehalten: Schwarzer Hintergrund, grüne Schlange, rotes Futter. Der Canvas hat einen weissen Rand und ist auf Mobil- und Desktopgeräten sichtbar und anpassbar.

**2.2 Responsive Design**

Das Spielfeld passt sich dynamisch der Fenstergrösse an:

* Auf grossen Bildschirmen bleibt es bei max. 600x600px
* Auf kleinen Geräten (z. B. Smartphones) nutzt es 90 % der Bildschirmbreite
* Das Seitenverhältnis ist immer quadratisch

**3. Architektur**

**3.1 Technologiestack**

* **HTML5**: Struktur der Webseite (Canvas, Titel, Punktestand)
* **CSS3**: Gestaltung (Farben, Layout, Responsive Design)
* **Vanilla JavaScript**: Spiellogik, Objektorientierung, Steuerung, Kollision

**3.2 Hauptkomponenten**

**Komponente Beschreibung**

*SnakeGame* Klasse Hauptklasse mit Spiellogik, Update-Zyklus, Zeichnen

*snake* Array Speichert die Segmente der Schlange (jeweils mit x/y)

*food* Objekt Position des aktuellen Punkts

*velocity* Objekt Aktuelle Bewegungsrichtung der Schlange

*update()* Methode Führt Spielschritte aus (Bewegung, Kollision, Wachstum)

*draw()* Methode Zeichnet Schlange und Food auf das Canvas

*resizeCanvas()* Passt Spielfeld bei Fensteränderung automatisch an

*isCollision()* Prüft auf Game-Over-Situationen (Wand/Selbstkollision)

**3.3 Datenstruktur**

* Spielfeld ist in ein **Grid (Raster)** mit 20x20 Feldern unterteilt
* Positionen der Schlange und des Futters werden in **x/y-Koordinaten** gespeichert
* Durch Set-Interval entsteht ein regelmässiger Spielrhythmus (Tickrate = 100ms)

**4. Tests**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Testfall** | **Beschreibung** | **Eingabe/ Aktion** | **Erwartetes Ergebnis** | **Eingetroffen?** |
| 1 | Spielstart | Seite aufrufen | Canvas ist sichtbar, Schlange erscheint in der Mitte | Ja |
| 2 | Bewegung | Pfeiltaste rechts drücken | Schlange bewegt sich nach rechts | Ja |
| 3 | Punkte sammeln | Schlange frisst rotes Quadrat | Schlange wird länger, Punktestand erhöht sich | Ja |
| 4 | Kollision mit Wand | Schlange gegen Rand steuern | Spiel stoppt, „Game Over“-Meldung erscheint | Ja |
| 5 | Selbstkollision | Schlange in sich selbst lenken | Spiel stoppt, „Game Over“-Meldung erscheint | Ja |
| 6 | Responsive Design | Fenstergrösse ändern / auf Smartphone testen | Canvas passt sich an, bleibt quadratisch | Ja |

**5. Basisversion**

* **Steuerung** über Pfeiltasten
* **Kollisionserkennung** mit Wand und sich selbst
* **Responsive Canvas**
* **Zufällig platzierte Punkte (Food)**
* **Schlange wächst mit jedem Punkt**
* **OOP-Ansatz mit der SnakeGame Klasse**
* **Clean Code mit Trennung von Logik und Darstellung**

**6. Probleme**

Bevor ich das Spiel abgeben wollte, ist mir noch aufgefallen, dass das Essen auch auf der Schlange spawnen kann. Dieses Problem musste ich also noch beheben. Dazu habe ich die Methode randomPosition() etwas überarbeitet. Ich habe eine do-while-Schleife eingebaut, die so lange eine neue Position generiert, bis sichergestellt ist, dass das Essen nicht auf einem Teil der Schlange erscheint.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

