# Tävling, Multiplayer Snake!

*Whoop whoop!*

## 1. Omfattning

1.1. Systemet ska utgöra ett komplett spel av typen Snake med stöd för både lokal spelomgång och nätverksbaserat multiplayer-läge.

1.2 Denna tävling och utmaning är frivillig.

1.3 All form av hjälp som ni kan ta till är tillåtet!



*Kalla in era favoritnördar, ert hemliga kodorakel eller vilken AI som helst!*

## 2. Grafiskt gränssnitt och interaktion

2.1. Spelet ska visa en spelplan där minst följande visuella element framgår tydligt:

- spelaren(s) mask(ar),
- spelobjekt som masken kan konsumera (t.ex. “mat”) skapas på slumpmässig plats,
- spelstatus (t.ex. pågående, pausad, game over),
- aktuell poäng per spelare.

2.2. Det grafiska gränssnittet ska upplevas som visuellt tilltalande, vilket i kravtermer innebär:

- tydlig visuell separation mellan bakgrund, spelplan, mask och spelobjekt,
- konsekvent färgsättning och formgivning inom spelets gränssnitt,
- läsbar typografi för text (poäng, namn, tidsinformation etc.).

2.3. Spelet ska kunna styras med tangentbord. Minimikrav:

- fyra riktningskommandon (upp, ner, vänster, höger) ska finnas,
- styrningen ska vara responsiv; ändring av riktning ska slå igenom nästkommande speluppdatering (tick).

2.4. Vid game over ska användaren få tydlig visuell feedback om att omgången är avslutad.

2.5. Utan omladdning av sidan ska användaren kunna starta en ny omgång efter game over.

## 3. Spelmekanik, grundläggande beteende

3.1. En mask ska representeras som en sekvens av sammanhängande segment (t.ex. rutor) där ett segment är markerat som huvudet.

3.2. Masken ska röra sig stegvis över spelplanen:

- varje speluppdatering (“tick”) flyttas huvudet ett steg i aktuell riktning,
- resten av kroppen följer huvudet enligt Snake-konvention (segmenten flyttas framåt).

3.3. När masken konsumerar ett spelobjekt som är avsett att öka längden ska:

- maskens längd öka enligt spelreglerna,
- den konsumerade “maten” ska försvinna och en ny “mat” ska placeras på slumpmässig plats.

3.4. Spelplanens dimensioner ska vara ändliga; spelaren får inte kunna lämna spelplanens begränsningsyta.

3.5. Speltakt (tick-frekvens) ska vara tillräckligt hög för att uppfattas som ett flytande spel, men inte styras dynamiskt av spelet på ett sätt som gör det oförutsägbart för spelaren.

## 4. Kollisionshantering

4.1. Systemet ska hantera minst följande kollisionstyper:

- kollision mellan mask och vägg (spelplanens gräns),
- kollision mellan mask och maskens egen kropp,
- kollision mellan två maskar.

4.2. Vid kollision med vägg eller egen kropp ska följande ske:

- masken ska “dö” och omedelbart återställas,
- maskens längd ska återställas till 2 segment,
- masken ska placeras på en slumpmässigt vald startposition som är giltig inom spelplanen,
- maskens nya position får inte placeras i en uppenbart ogiltig position (t.ex. utanför spelplanen).

4.3. Vid kollision mellan två maskar ska den mask som “kör in” i den andra anses ha förlorat kollisionen:

- masken som kör in ska behandlas som “död” enligt 4.2,
- masken som blir påkörd ska fortsätta existera oförändrad,
- det ska vara deterministiskt definierat vad “kör in” innebär (t.ex. att ett huvud går in i en ruta där annan mask redan befinner sig).

4.4. Kollisionslogiken ska vara konsekvent och reproducerbar; samma situation ska alltid ge samma utfall.

## 5. Matchlogik och game over

5.1. En match ska ha en ändlig speltid.

5.2. När speltiden är slut ska matchen avslutas och en vinnare utses.

5.3. Vinnare definieras som den mask (spelare) som vid speltidens slut har längst längd.

5.4. Vid lika längd ska systemet definiera och implementera en deterministisk tie-breaker (t.ex. oavgjort eller baserat på annan mätbar parameter). Val av modell är valfritt men måste vara konsekvent inom implementationen.

5.5. När matchen avslutas ska systemet:

- visa vilken spelare som vunnit (eller att matchen är oavgjord),
- stoppa speluppdateringar (ingen fortsatt rörelse),
- ge användaren möjlighet att starta en ny match utan sidomladdning.

## 6. Scoreboard och persistens

6.1. Systemet ska innehålla ett scoreboard för lagring av spelresultat.

6.2. Scoreboard ska minst lagra:

- spelarnamn (sträng),
- poäng (tal, där högre värde representerar bättre resultat).

6.3. Scoreboard ska använda `localStorage` i webbläsaren som lagringsmekanism.

6.4. Scoreboard ska vara persistent över sidomladdningar; data i `localStorage` får inte raderas vid normal användning.

6.5. Systemet ska kunna läsa in befintligt scoreboard från `localStorage` vid start.

6.6. Systemet ska kunna skriva nya poster till `localStorage` vid relevant tidpunkt (t.ex. vid matchens slut).

6.7. Scoreboard ska visas i det grafiska gränssnittet:

- minst en vy där användaren kan se sparade resultat,
- listan ska tydligt visa både namn och poäng.

6.8. Det ska finnas en definierad sorteringsordning för scoreboard (t.ex. fallande poäng). Val av ordning är valfritt men måste vara konsekvent.

## 7. Multiplayer

7.1. Systemet ska ha ett multiplayer-läge där flera klienter kan delta i samma match via en extern server. En API-modul på klientsidan och en server tillhandahålls av utbildaren.

7.2. Varje nätverksbaserad match ska kopplas till en unik session på servern.

7.3. En klient som vill skapa en ny multiplayer-session ska initiera detta via API-modulens funktion .host(). API-modulen kommunicerar med servern.

7.4. När en multiplayer-session skapas genererar servern ett sessions-ID.

7.5. Klienter som ska ansluta till befintlig multiplayer-session API-modulens funktion .join(sessionID)

7.6. Klientapplikationen får inte använda några egna nätverksanrop mot servern; all nätverkskommunikation ska ske via API-modulens funktionsanrop.

7.7. Klientapplikationen får inte innehålla någon kunskap om underliggande kommunikationsprotokoll; sessioner, kommandon och data ska hanteras abstrakt via API-modulens publika metoder.

## 8. Kommunikationsprotokoll (Färdig API-modul)

8.1. Importera och skapa en ny instans av API-modulen:

```
import { MultiplayerApi } from './MultiplayerApi.js';
const api = new MultiplayerApi('ws://localhost:8080');
```

8.2 Skapa en session som värd (returnerar en Promise och resolvar när sessionen är skapad):

```
api.host();
```

8.3 Ansluta till en befintlig session (returnerar en Promise och resolvar när ansluten till session):

```
api.join(sessionID, data);
```

(data är ett Object och kan tex. innehålla spelarens namn, vald färg på ormen osv.)

8.4 Skicka spel-data (data är ett Object och skickas till samtliga klienter inkl. host):

```
api.game(data);
```

(Varje meddelande markeras med ett sekventiellt id nummer som ska användas för att sortera inkommande händelser i kronologisk ordning, detta görs inte av servern eller API-modulen)

8.5 Lyssna på inkommande händelser:

```
const unsubscribe = api.listen((event, messageId, clientId, data) => {});
```

Följande “event” (som är en String) kan komma in:

- “joined” (en spelare har anslutit till sessionen)
- “leaved” (en spelare har lämnat sessionen) OBS! Inte implementerat än!
- “game” (inkommande spel-data från andra klienter)

8.6 Sluta lyssna på inkommande meddelande:

```
unsubscribe();
```

## 9. Stabilitet och robusthet i multiplayer

9.1. Systemet ska kunna hantera att klienter ansluter och kopplar från en session under pågående match.

9.2. Systemet ska kunna hantera att minst två klienter deltar i samma session samtidigt.

9.3. Multiplayer-funktionaliteten ska vara stabil i den meningen att:

- spelare i samma session ska få relevanta uppdateringar om spelets tillstånd utan oavsiktliga låsningar eller krascher,
- en enskild klients frånkoppling inte får stoppa spelet.

9.4. Vid nätverksfel eller oförväntad frånkoppling ska klientlogiken kunna hantera situationen kontrollerat (t.ex. genom att visa fel, pausa spel etc.). Exakt hantering är valfri men ska vara deterministisk inom implementationen.

## 10. Tävlingen

10.1 Spelet måste finnas publikt åtkomligt för alla senast 2025-12-22 00:00:00

10.2 Länken till spelet ska lämnas till utbildaren som lägger upp samtliga spel som har lämnats in I tid på en gemensam sida.

10.3 Utbildaren tilldelar ett nummer till varje student som ska synas tydligt i spelet som sen används i röstningen.

10.4 Spelet ska länka till en omröstningslänk som tillhandahålls av utbildaren.

10.5 Omröstning sker under julveckan (2025-12-22 00:00:00 till 2025-12-28 00:00:00)

10.6 Utvecklaren med flest röster vinner ett litet pris.

10.7 God sed förutsätts. (Man får inte rösta på sitt eget spel eller låta sina vänner rösta på sitt spel)

10.8 Det är tillåtet att låta nära och kära att spela och rösta. Men lämna då den gemensamma länken där alla spel hittas.

## 10. Kodstruktur och användning av klasser

10.1. Implementationen ska använda klasser för att strukturera spelkoden.

10.2. Minst de centrala domänkoncepten i spelet ska representeras av klasser, exempelvis:

- spel/match
- mask
- spelplan/bana
- hantering av multiplayer-tillstånd

10.3. Klasserna ska utgöra separerade ansvarsområden; en klass ska inte kombinera orelaterade funktioner (t.ex. spelregler och rendering i samma klass utan tydligt motiv).

10.4. Det finns inga krav på specifik modulstruktur, namngivning av filer eller katalogdisciplin, men koden ska vara organiserad på ett sätt som möjliggör tydlig separering mellan:

- spelregler och logik,
- användargränssnitt,
- kommunikation mot API-modulen.

## 11. Icke-krav och begränsningar

11.1. Det ställs inga krav på:

- WCAG-efterlevnad,
- automatiserade tester,
- extern dokumentation,
- byggsystem,
- deploymiljö.

11.2. Ingen särskild renderingsmetod föreskrivs; val av teknik för rendering (t.ex. canvas, DOM, bibliotek) är fritt.

11.3. Spelet ska inte använda någon färdig spelmotor (game engine)