Kennis Check Blok 2

Kennis Check Blok 2.1

datasource

Vallende Stenen

Waarom is het handig om de speler en de vallende steen elk met een eigen variabele voor x en y te beheren?

Zo kun je de positie van elk object onafhankelijk aanpassen en er later berekeningen mee doen, zoals bewegen of botsing detecteren. Als je maar één variabele zou gebruiken, zou je die flexibiliteit verliezen.

Wat is het verschil tussen tekenen op het scherm en het updaten van een positie?

Tekenen op het scherm gebeurt in de `draw()`-functie en bepaalt wat de speler ziet. Updaten van posities gebeurt in de `update()`-functie en bepaalt waar objecten zich bevinden. Zonder update gebeurt er niets; zonder draw zie je niets.

Waarom gebruik je een functie zoals `colliderect()` en schrijf je geen eigen code om te controleren of objecten elkaar raken?

`colliderect()` is getest en betrouwbaar. Zelf botsingsdetectie schrijven is foutgevoelig en complex. Door bestaande functies te gebruiken maak je je code korter, leesbaarder en minder foutgevoelig.

Waarom wordt er gewerkt met `random.randint()` om de x-positie van de steen te kiezen?

Daardoor weet de speler niet waar de volgende steen zal vallen, wat het spel spannender maakt. Zonder toeval zou het spel voorspelbaar en saai worden.

Wat gebeurt er als je geen limiet stelt aan de onderkant van het scherm voor vallende objecten?

Dan vallen stenen eindeloos door en verdwijnen ze uit beeld, wat kan zorgen voor geheugenproblemen of verwarring bij de speler. Je moet ze resetten of verwijderen als ze te ver vallen.

Waarom wordt in het spel vaak gebruik gemaakt van globale variabelen?

Omdat `draw()` en `update()` automatisch worden aangeroepen door Pygame Zero en geen parameters gebruiken, moeten variabelen buiten deze functies beschikbaar zijn. Globale variabelen maken dit mogelijk.

Wat is het verschil tussen een spel dat ‘reageert’ op invoer en een animatie die altijd hetzelfde doet?

Een spel reageert op wat de speler doet, zoals het indrukken van pijltjestoetsen. Bij een animatie loopt alles vanzelf, zonder dat de gebruiker invloed heeft. Interactiviteit is wat van een animatie een spel maakt.

Snake

Waarom bestaat een slang uit een lijst met segmenten in plaats van één positie?

Doordat de slang uit meerdere segmenten bestaat kun je de groei en beweging ervan beter simuleren: bij eten wordt een segment toegevoegd en anders wordt het achterste segment verwijderd. Zo ontstaat een realistische slangbeweging.

Wat is het nut van een timer variabele in `update(dt)`?

De timer bepaalt hoe snel de slang beweegt (bijv. elke 0.15 seconden). Door `dt` op te tellen weet je precies wanneer de slang een stapje moet maken, los van de framerate.

Waarom gebruiken we een `direction_queue` in plaats van alleen de huidige richting?

Een queue maakt het mogelijk om meerdere toetsaanslagen op te slaan vóór de volgende beweging. Zo kan de slang snel achter elkaar van richting veranderen, zonder meteen om te draaien en zichzelf te 'eten'.

Waarom gebruiken we `collide rect()` ?

`collide rect()` is eenvoudiger en betrouwbaarder: het controleert rechthoekige botsingen automatisch. Handmatig rekenen op coördinaten kan fouten veroorzaken.

Wat zou er gebeuren als we de slang laten 'wrappen' naar de andere kant van het scherm bij randbotsing?

Door wrappen speelt het spel vloeiender en blijft de slang ononderbroken bewegen. Zonder wrap zou botsen op de rand een abrupt einde betekenen.

Hoe draagt het willekeurig plaatsen van voedsel bij aan de speelervaring?

Het willekeurig zetten van voedsel voorkomt patronen en houdt het spel uitdagend. De speler moet telkens de slang naar een onverwachte plek sturen.

Wat gebeurt er als de slang op zichzelf botst, en waarom willen we dat checken vóór het toevoegen van een nieuw segment?

Botst de slang op zichzelf, dan eindigt het spel. Door te checken vóór je de kop toevoegt, voorkom je dat de slang in al zijn segmenten terechtkomt — dat voorkomt onverwacht gedrag.

Waarom hebben we een `reset()`-functie die opnieuw alles initialiseert?

Een `reset()`-functie zorgt voor een strak begin na Game Over, met dezelfde beginvoorwaarden (slangpositie, timer, richting). Zo kun je herstarten zonder fouten.

Hoe maak je het spel moeilijker zonder de basiscode te veranderen?

Je kunt de moeilijkheid verhogen door de bewegingstimer in te korten, extra obstakels toe te voegen, of minder ruimte voor voedsel te geven. Zo blijft het uitdagend.

Introductie AI

Wat maakt AI anders dan gewone code?

AI-code is gebaseerd op het leren uit voorbeelden en herkent patronen, terwijl gewone code exact reageert op geprogrammeerde instructies. AI kan dus variabelen voorspellen, terwijl gewone code voorspelbaar blijft :contentReference[oaicite:1]{index=1}.

Waarom onderscheidt men AI-toepassingen in categorieën als classificatie, associatie, enz.?

Zo kun je concrete voorbeelden koppelen aan de manier waarop AI werkt. Elke categorie heeft eigen eigenschappen: classificatie herkent labels, optimalisatie zoekt naar de beste oplossing, creatie gaat over nieuwe inhoud maken, etc. :contentReference[oaicite:2]{index=2}.

Hoe helpt het indelen van AI in typen zoals voorspelling en optimalisatie om je begrip te verdiepen?

Je leert zo dat AI niet één technologie is, maar verschillende werkwijzen heeft afhankelijk van het doel—zoals voorspellen van verkoop of optimaliseren van routes. Dit maakt het makkelijker te kiezen welke techniek geschikt is voor een probleem.
:contentReference[oaicite:3]{index=3}.

Wat betekent het dat AI 'minder voorspelbaar' is dan klassieke code?

Normale code geeft altijd hetzelfde resultaat bij dezelfde input. AI geeft verschillende output, omdat het leert van data en tot een inschatting komt. Hierdoor kan het fouten maken in onbekende situaties :contentReference[oaicite:4]{index=4}.

Waarom is het belangrijk om de voor- en nadelen van AI te begrijpen?

AI heeft veel kracht, zoals patronen herkennen en personaliseren, maar kan ook fouten maken, bevooroordeeld zijn of ondoorzichtig zijn ("black box"). Door deze aspecten te begrijpen kun je AI verantwoordelijk en effectief inzetten. :contentReference[oaicite:5]{index=5}.

Hoe zorgt goede prompt engineering ervoor dat je AI beter helpt begrijpen wat je wilt?

Door context, doel, doelgroep en format duidelijk te geven, krijgt de AI de juiste informatie om relevante en bruikbare output te leveren. Vage prompts leiden vaak tot onsamenhangende antwoorden.

Waarom moet je AI-output altijd controleren, vooral in belangrijke situaties?

AI-systemen kunnen fouten maken, ongepaste of bevooroordeelde antwoorden geven—vooral zonder toezicht. Daarom is menselijke verificatie essentieel, bijvoorbeeld bij medische of juridische adviezen. :contentReference[oaicite:6]{index=6}.

Wat betekent het dat AI een ‘black box’ is, en waarom is dat problematisch?

Je weet vaak niet precies hoe een AI tot een bepaalde beslissing komt. Dit maakt het lastig om die beslissing uit te leggen of te controleren, wat risico's kan veroorzaken in gevoelige toepassingen. :contentReference[oaicite:7]{index=7}.

?? Opdracht

Maak nu de kennis-check.

? Inleveren

Aan het einde van de kennis-check ontvang je een certificaat. Maak een schermafdruk en lever deze in.

Kennis-check Blok 2.2

?? Opdracht

Maak nu de kennis-check.

? Inleveren

Aan het einde van de kennis-check ontvang je een certificaat. Maak een schermafdruk en lever deze in.

Revision #14
Created 2025-06-14 19:02:25 UTC by Max

Updated 2026-07-08 15:03:11 UTC by Max