

Mircică Melisa-Maria, Igescu Rareș-Andrei, Ceașescu Claudiu-Marius

Anul 2, Semestrul 2, Grupa 232

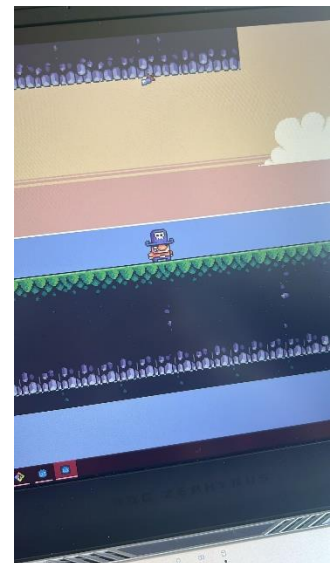
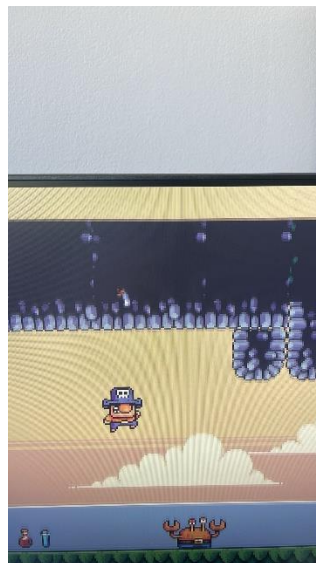
Metode de Dezvoltare Software

Fenomenul de Prompt Engineering

“Prompt Engineering” este procesul de structurare și construire a unei instrucțiuni pentru a produce cel mai bun posibil output de la un model de Inteligență Artificială generativ. Mai informal, este acțiunea directă prin care cerem unui model AI să rezolve o cerință.

În cadrul proiectului dezvoltat de echipa mea, am folosit diferite tool-uri de generare automată – în special ChatGPT și DeepSeek – fie că vorbim de generarea unui exemplu de cod, de redactarea unor explicații sau de bunele practici ale dezvoltării jocurilor video. Am apelat la modele de inteligență artificială deoarece eu și colegii mei am intrat fără cunoștințe în dezvoltarea unui joc video și aveam nevoie să procurăm informație concisă dar corectă cât mai rapid. Cum am menționat mai sus, pe parcursul dezvoltării am comparat în mod constant răspunsurile dintre ChatGPT și DeepSeek. Deși cele 2 modele nu primeau mereu același prompt, ele au fost folosite pentru aceleași scopuri, printre care se numără: clarificarea sintaxei GDScript, generarea bucăților de cod complexe sau “de nișă” – care vor fi prezentate mai jos – și crearea de idei și tematici care să se potrivească cu povestea de bază a jocului nostru.

În cele ce urmează vom compara răspunsurile la diferite prompt-uri primite de cele 2 modele. Una din problemele pe care am avut-o destul de devreme în dezvoltarea proiectului a fost asignarea corectă a animațiilor, în special asupra sabiei caracterului principal.



Pentru a face o comparație corectă, vom da ca celor 2 modele același prim prompt, pentru a vedea care oferă un răspuns mai corect. Ambele modele au primit acest prompt:

Dezvolt un joc in 2D in Godot alaturi de 2 colegi de ai mei la facultate. Jocul este un platformer. Avem o problema la niste animatii ale caracterului principal: el poate sa isi arunce sabia care mai apoi se infinge in pamant si are 4 feluri in care se poate infinge, in tavan, in peretele din stanga, in peretele din dreapta si in podea, problema noastra este ca, de exemplu, atunci cand se infinge in tavan, se activeaza animatia de infingere in podea si totul arata foarte gresit, m-ai putea ajuta in acest aspect?

Răspunsul primit de la ChatGPT a fost, la general, unul mult mai plin de detalii, am primit, pe lângă cod, o structură de bază a scenei “Sword” din Godot în care este explicat fiecare nod și la ce ajută în cadrul sabiei. În următoarea secvență ne-a fost descris codul (chiar întreg scriptul) care ar putea fi asociat scenei menționate anterior. Bineînțeles fiecare bucată de cod a fost însoțită de scurte comentarii pentru a înțelege mai ușor la ce folosește fiecare funcție și variabilă definită. Mai apoi urmează multe alte aliniat generate de ChatGPT, dintre care merită menționate următoarele: felul în care pot seta parametrii pentru a obține efecte variate în aruncare sabiei, greșeli frecvente și cum as putea să le repar (de apreciat, având în vedere cunoștințele noastre reduse în dezvoltarea jocurilor video) și alte variante avansate prin care aș putea implementa aruncarea sabiei.

1. Structura de bază a săbiei

Presupunem că nodul „Sword” (săbia) arată cam așa:

scss

Copy

Edit

Sword (KinematicBody2D)

└ CollisionShape2D # forma de coliziune (dreptunghi/tragaci)

└ AnimatedSprite (sau AnimationPlayer)

- **KinematicBody2D**: ca să poți folosi `move_and_collide()` și să primești obiectul `KinematicCollision2D`.
- **CollisionShape2D**: definești coliziunea săbiei.
- **AnimatedSprite** (sau **AnimationPlayer**): pentru a reda animațiile de „infingere” în cele patru direcții. Să presupunem că ai patru animații diferite:
 - `"stick_floor"` (infingere în podea),
 - `"stick_ceiling"` (infingere în tavan),
 - `"stick_wall_left"` (infingere în perete stânga),
 - `"stick_wall_right"` (infingere în perete dreapta).

2. Fizica și detectarea coliziunii

În `_physics_process(delta)`, vei muta sabia cu `move_and_collide()` (sau `move_and_slide()`), dar pentru detectarea normalelor e mai simplu cu `move_and_collide()`. Când `move_and_collide` găsește o coliziune, îți returnează un obiect `KinematicCollision2D`, din care extragi `collision.normal`. Apoi poți compara direcția normalelor cu patru vectori canonici:

- `Vector2.UP` (0, -1) indică suprafață orizontală (podea) (normala sa e în sus),
- `Vector2.DOWN` (0, +1) indică tavan (normala în jos),
- `Vector2.LEFT` (-1, 0) indică perete-dreapta (normala spre stânga),
- `Vector2.RIGHT` (+1, 0) indică perete-stânga (normala spre dreapta).

Cel mai sigur e să folosești dot-product și un prag (de exemplu > 0.9) ca să nu ai erori de floating-point.

Exemplu de script pentru „Sword.gd”:

```
gdsript Copy Edit

extends KinematicBody2D

# * viteza săbiei *
var velocity: Vector2 = Vector2.ZERO
# * gravitație, dacă ar mai cădea *
const GRAVITY = Vector2(0, 2000)
# * viteză inițială când e aruncată (setează exterior) *
var initial_speed = 800

# Nume animații în AnimatedSprite (sau referință la AnimationPlayer)
const ANIM_STICK_FLOOR = "stick_floor"
const ANIM_STICK_CEILING = "stick_ceiling"
const ANIM_STICK_WALL_L = "stick_wall_left"
const ANIM_STICK_WALL_R = "stick_wall_right"

onready var sprite: AnimatedSprite = $AnimatedSprite

func _ready():
    # La început, sabia e inactivă sau fără mișcare; poate fi lansată din exterior.
    velocity = Vector2.ZERO
    sprite.play("idle") # eventual o animație „inertă”

func throw_sword(direction: Vector2):
    # Apelată din afară: direction e Vector2(-1,0) sau (1,0) etc.
    # Îi dai viteza inițială și îi permiți să cadă
    velocity = direction.normalized() * initial_speed
```

4. Situații frecvente de greșeli și cum le reparați

1. Nu folosești `move_and_collide` (sau `move_and_slide`) corect:

- Dacă pui sabia sub formă de `Area2D` fără să aibă un `Body2D`, nu primești normală direct. Într-un `Area2D` poți doar detecta overlap-ul și nu ai normală.
- Soluția: fă sabia `KinematicBody2D` sau `RigidBody2D` (cu modul `STATIC/CHARACTER`) ca să primești `collision.get_normal()`.

2. Ai setat forțele/impactul cam foarte mic, așa că nu declanșează coliziunea „corect”:

- Dacă viteza e prea mică, `move_and_collide(velocity * delta)` nu vede coliziunea; asigură-te că distanța parcursă în calcul e potrivită.

3. Animația implicită (ex: “stick_floor”) rămâne activă pentru că nu oprești fizica:

- Dacă nu apelezi `set_physics_process(false)` după ce se fixează sabia, în următorul *frame* `_physics_process` se execută iar și poate detecta altă coliziune, cauzând suprascrierea animației.
- Soluția: imediat după ce dai animația, dezactivezi procesarea fizicii, sau setezi `monitoring = false`, `collision_layer = 0` etc.

4. Ai mai multe colizioane aproape de suprafață (ex: două tile-uri) și detectezi mai multe normaluri:

- În colțurile unde două tile-uri adiacente au coliziuni (colț între podea și perete), e posibil să primești normala medie `(0.707, -0.707)`. Dacă nu folosești praguri (ex. `>0.9`), s-ar putea să ajungă la altă animație.
- Soluția: alege ce componentă a normalului e mai mare (`abs(normal.x) > abs(normal.y)` → e perete; altfel → podea/tavan), ori folosește un prag mai mic.

Răspunsul inițial primit de la DeepSeek a fost unul mult mai scurt. În care era prezentată doar o funcție în GDScript care ar trebui să implementeze logica cerută de mine. A menționat anumite metode de debugging pentru a înțelege de ce nu funcționează așa cum mă aștept, apoi mi-a descris pe scurt câteva cauze pentru care codul meu nu rulează corect.

Soluție propusă

1. Verificați logica de detectare a coliziunii:

gdscript

Copy Download

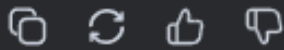
```
func _on_Sword_body_entered(body):
    var collision_normal = $CollisionShape2D.get_global_transform().xform_inv(body.global_position)
    var impact_angle = rad2deg(collision_normal.angle())

    if impact_angle > 45 and impact_angle < 135: # Impact cu tavan
        $AnimationPlayer.play("stick_ceiling")
    elif impact_angle < -45 and impact_angle > -135: # Impact cu podea
        $AnimationPlayer.play("stick_floor")
    elif impact_angle >= 135 or impact_angle <= -135: # Impact cu perete stânga
        $AnimationPlayer.play("stick_left_wall")
    else: # Impact cu perete dreapta
        $AnimationPlayer.play("stick_right_wall")
```

Dacă problema persistă, aş avea nevoie de mai multe detalii despre:

- Cum este implementat sistemul de aruncare a sabiei
- Cum sunt configurate animațiile în AnimationPlayer
- Ce tip de coliziuni folosiți (Area2D, KinematicBody2D etc.)

Sper că aceste sugestii vă ajută să rezolvați problema!



Cu toate acestea, m-a rugat să îi ofer mai multe detalii pentru a înțelege mai bine problema de care am parte, cuvinte care nu au fost prezente în răspunsul primit de la competitorul său. Astfel, am decis să îi ofer informațiile despre care m-a întrebat, fără ca totuși să îi dau cod.

Avem un sistem de coliziuni care se bazeaza pe Area2D, avem o zona pentru fiecare directie in care putem arunca sabia (Up, Down, Left, Right). In momentul in care jucatorul apasa tasta E, se declanseaza aruncarea sabiei din mana caracterului principal si parcurge distanta setata in Script, ca mai apoi sa se infinga in peretele in care se opreste. Daca jucatorul e cu fata spre stanga, va arunca sabia spre stanga, daca e in saritura va arunca sabia in sus, la fel pentru celelalte directii. Dupa ce sabia a parcurs distanta setata, ar trebui sa se declanseze animatia pentru directia din care a fost aruncata.

Cu toate aceste explicații în plus față de ChatGPT, răspunsul a fost unul la fel de dezamăgitor ca primul. Am primit 3 bucăți de cod, care duc lipsă de comentarii pentru a înțelege mai ușor ceea ce a fost descris și nu dispun nici de o explicație separată pentru a înțelege felul în care ar trebui legate între ele. Singurul plus de care am avut parte este că, drept “bonus” am primit o funcție de recall, prin care, după un anumit timp din momentul în care a fost aruncată, sabia se va întoarce la jucător.

În concluzie, am decis să folosim ChatGPT drept ajutor pentru dezvoltarea proiectului. Ne-a ajutat să adunăm informații utile într-un timp relativ scurt și ori de câte ori aveam întrebări reușea să ofere un răspuns mult mai aproape de adevăr față de alte modele. Am recurs la ajutor din partea inteligenței artificiale deoarece, în comparație cu alte motoare de dezvoltare a jocurilor video (Ex.

Unity) nu am avut parte de foarte multe exemple sau documentații, lucru care a fost compensat de multe ori de inteligența artificială.

Referințe:

- [Wikipedia](#)
- [Coursera](#)