Institut Puig Castellar
Santa Coloma de Gramenet

MULTI GAMES GUAU

(Projecte de desenvolupament)

CFGM Administració de Sistemes Informàtics i Xarxes

Autors: Aitor Carrillo Mata
Grup: B
Curs acadèmic: SMX2

INDEX

Llicències

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Resum del projecte:
Multi Games GUAU és un projecte de desenvolupament d'un conjunt de videojocs arcade inspirats en clàssics del gènere, creat íntegrament amb el motor Godot Engine 4 i el llenguatge de programació GDScript.
L'objectiu del projecte és desenvolupar un producte funcional, estable i accessible que reuneixi diversos jocs jugables des d'un menú principal comú, cadascun amb les seves pròpies mecàniques, interfície i sistema de puntuació màxima persistent entre sessions.
Per assolir aquest objectiu es va seguir una metodologia àgil, dividint el desenvolupament en fases curtes i iteratives: investigació de tecnologies, disseny dels jocs, implementació de la lògica i les interfícies, proves i correcció d'errors, i integració final de tots els jocs en un mateix entorn. Es van utilitzar GitHub Desktop per al control de versions i Padlet per a la gestió de tasques.
Cal destacar que el projecte va haver de canviar d'orientació a mitja durada, passant d'un joc de rol Pokemon Classic a la proposta actual, cosa que va suposar un exercici real de gestió de projecte i presa de decisions tècniques.
Com a conclusions, el projecte ha complert tots els objectius definits: els vuit jocs funcionen correctament, el sistema de guardat de puntuacions opera de manera transparent, i el resultat final és un producte complet, jugable i accessible per a qualsevol tipus d'usuari.

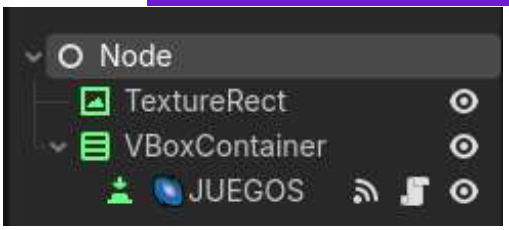
Paraules clau: Videojocs arcade, Godot Engine 4, GDScript, Desenvolupament de videojocs, Dificultat progressiva, Menú principal, Puntuació màxima, Metodologia àgil.

Abstract:
Multi Games GUAU is a development project consisting of a collection of arcade video games inspired by genre classics, built entirely using the Godot Engine 4 game engine and its programming language, GDScript.
The objective of the project is to deliver a functional, stable, and accessible product that brings together several playable games within a shared main menu, each featuring its own mechanics, interface, and persistent high score system.
To achieve this objective, an agile methodology was followed, dividing the development into short iterative phases: technology research, game design, implementation of game logic and interfaces, testing and bug fixing, and final integration of all games into a unified environment. GitHub Desktop was used for version control and Padlet for task management. It is worth noting that the project had to change direction midway through development, shifting from a role-playing game concept Pokemon Classic to the current proposal, which represented a real exercise in project management and technical decision-making.
In conclusion, the project has met all defined objectives: all eight games work correctly, the score-saving system operates transparently in the background, and the final result is a complete, playable product accessible to any type of user regardless of their gaming experience.

Keywords: Arcade video games, Godot Engine 4, GDScript, Video game development, Progressive difficulty, Main menu, High score system, Agile methodology.

Llista de figures

MENÚ PRINCIPAL



Estructura de nodes:

Node: Node arrel de l'escena. Conté tots els elements del menú principal.

TextureRect: Mostra la imatge de fons del menú principal amb els diferents jocs representats visualment.

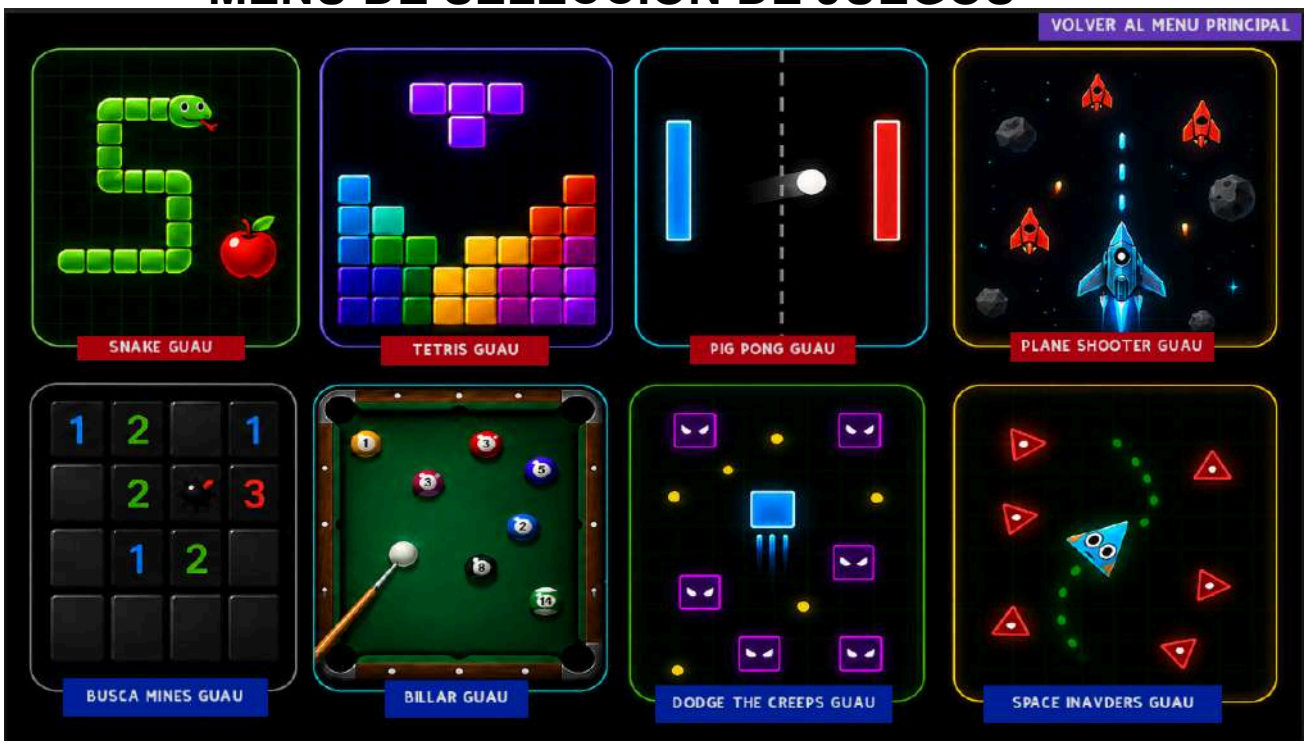
VBoxContainer: Contenidor vertical que organitza els elements interactius del menú de manera ordenada.

JUEGOS (Button): Botó principal que permet a l'usuari navegar cap al menú de selecció de jocs. Té associat el script ***JUEGOS.gd***.

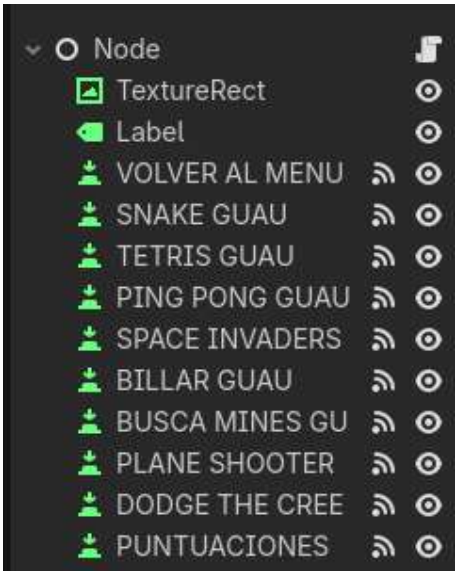
Script: JUEGOS.gd

Aquest script s'encarrega de gestionar l'acció del botó principal del menú. Quan l'usuari prem el botó, la funció ***_on_pressed()*** s'executa i canvia l'escena activa a la pantalla de selecció de jocs mitjançant la funció ***change_scene_to_file()***.

MENÚ DE SELECCIÓN DE JUEGOS



Árbol de nodos y scripts



selecion_de_juegos.gd

```gdscript
extends Node

func _on_snake_guau_pressed():
    get_tree().root.content_scale_size = Vector2i(1000, 1050)
    get_tree().change_scene_to_file("res://SNAKE GUAU/scenes/main.tscn")

func _on_tetris_guau_pressed():
    get_tree().root.content_scale_size = Vector2i(850, 750)
    get_tree().change_scene_to_file("res://tetris-guau-main/scenes/tile_map.tscn")

func _on_billar_guau_pressed():
    get_tree().root.content_scale_size = Vector2i(1200, 775)
    get_tree().change_scene_to_file("res://Pool_tut-main/scenes/main.tscn")

func _on_busca_mines_guau_pressed():
    get_tree().root.content_scale_size = Vector2i(750, 750)
    get_tree().change_scene_to_file("res://Busca-minas-guau-main/scenes/main.tscn")

func _on_pig_pong_guau_pressed() -> void:
    get_tree().change_scene_to_file("res://PingPongGodot-master/Scenes/main.tscn")

func _on_space_inavders_guau_pressed() -> void:
    get_tree().change_scene_to_file("res://TopDownShooter/scenes/game.tscn")

func _on_volver_al_menu_principal_pressed() -> void:
    get_tree().change_scene_to_file("res://SCENES/menu.tscn")


func _on_plane_shooter_guau_pressed() -> void:
    get_tree().root.content_scale_size = Vector2i(860, 1500)
    get_tree().change_scene_to_file("res://PlaneShooter-master/Scenes/MainScene.tscn")


func _on_dodge_the_creeps_guau_pressed() -> void:
    get_tree().change_scene_to_file("res://dodge_the_creeps/Main.tscn")
```

Estructura de nodes:

Node: node arrel de l'escena

TextureRect: imatge de fons de la pantalla de selecció

Label: text informatiu de la pantalla

VOLVER AL MENU: botó per tornar al menú principal

SNAKE GUAU: botó que carrega Snake

TETRIS GUAU: botó que carrega Tetris

PING PONG GUAU: botó que carrega Pong

SPACE INVADERS: botó que carrega Space Invaders

BILLAR GUAU: botó que carrega Billar

BUSCA MINES GUAU: botó que carrega Busca Mines

PLANE SHOOTER GUAU: botó que carrega Plane Shooter

DODGE THE CREE GUAU: botó que carrega Dodge the Creeps

PUNTUACIONES: botó que carrega la pantalla de puntuacions

Script: selecion_de_juegos.gd

Cada funció canvia la mida de la finestra i carrega l'escena corresponent al joc seleccionat. Per exemple, Snake carrega a 1000×1050 píxels, Tetris a 850×750, Billar a 1200×775, etc. Això permet que cada joc s'adapti a la resolució òptima per al seu disseny.

SNAKE GUAU

Estructura de nodes:

Main (Node): node arrel que conté tota la lògica i els elements de l'escena del joc.

TextureRect: mostra la imatge de fons del tauler de joc, amb el patró quadriculat característic del Snake.

Hud (CanvasLayer): capa d'interfície que es mostra per sobre del joc en tot moment i conté:

- ScorePanel: panell visual que emmarca la informació de puntuació.
- **ScoreLabel:** etiqueta que mostra la puntuació actual del jugador en temps real.
- **Salir (Button):** botó per tornar a la selecció de jocs. En prémer-lo, restaura la resolució a 1920×1080 i carrega l'escena de selecció de jocs.

MoveTimer (Timer): temporitzador que controla la velocitat de moviment de la serp. Cada vegada que s'esgota, la serp avança una cel·la en la direcció activa.

Food: node que representa el menjar que ha de recollir la serp. La seva posició es genera aleatòriament en cel·les lliures del tauler.

GameOverMenu (CanvasLayer): capa d'interfície que apareix quan la partida acaba, i conté:

- **GameOverPanel:** panell visual del menú de fi de partida
- **ResultLabel:** etiqueta que mostra el resultat final
- **RestartButton:** botó que emet el senyal restart per reiniciar la partida sense recarregar l'escena

Scripts:

main.gd: és el script principal del joc. S'encarrega de:

- Definir el tauler de 20×20 cel·les amb una mida de 50px per cel·la
- Registrar els controls de moviment (fletxes i WASD) dinàmicament si no existeixen
- Generar la serp inicial de 3 segments centrada al tauler
- Moure la serp cada tick del MoveTimer, actualitzant les posicions de tots els segments
- Detectar col·lisions amb els límits del tauler i amb el propi cos de la serp
- Gestionar la recollida del menjar: suma 1 punt, afegeix un segment nou i genera una nova posició de menjar aleatòria en una cel·la lliure
- Guardar la puntuació màxima amb Puntuacion.save_score("snake", score) en acabar la partida
- Mostrar el GameOverMenu i pausar l'arbre de nodes quan la partida acaba

GameOverMenu.gd: script del node GameOverMenu. Quan el jugador prem RestartButton, emet el senyal restart que el node Main recull per reiniciar la partida.

salir.gd: script del botó de sortida del Hud. Restaura la resolució de la finestra a 1920×1080 i torna a l'escena de selecció de jocs.

TETRIS GUAU

Estructura de nodes:

TileMap: node arrel del joc. Gestiona el tauler de joc mitjançant un sistema de tessel·les (tiles) organitzat en dues capes: la capa 0 és el tauler fix on es col·loquen les peces que han aterrat, i la capa 1 és la capa activa on es dibuixa la peça en moviment. Conté el script principal del joc.

HUD (CanvasLayer): capa d'interfície que es mostra per sobre del joc i conté:

- **Panel:** panell visual de fons de la interfície
- **Label:** etiqueta informativa del joc
- **ScoreLabel:** etiqueta que mostra la puntuació actual en temps real
- **GameOverLabel:** etiqueta que s'amaga durant la partida i apareix quan el joc acaba
- **StartButton:** botó per iniciar una nova partida

Button: botó per tornar a la selecció de jocs. Restaura la resolució a 1920×1080 i carrega l'escena de selecció de jocs.

TECLAS: node informatiu que mostra els controls del joc a la pantalla.

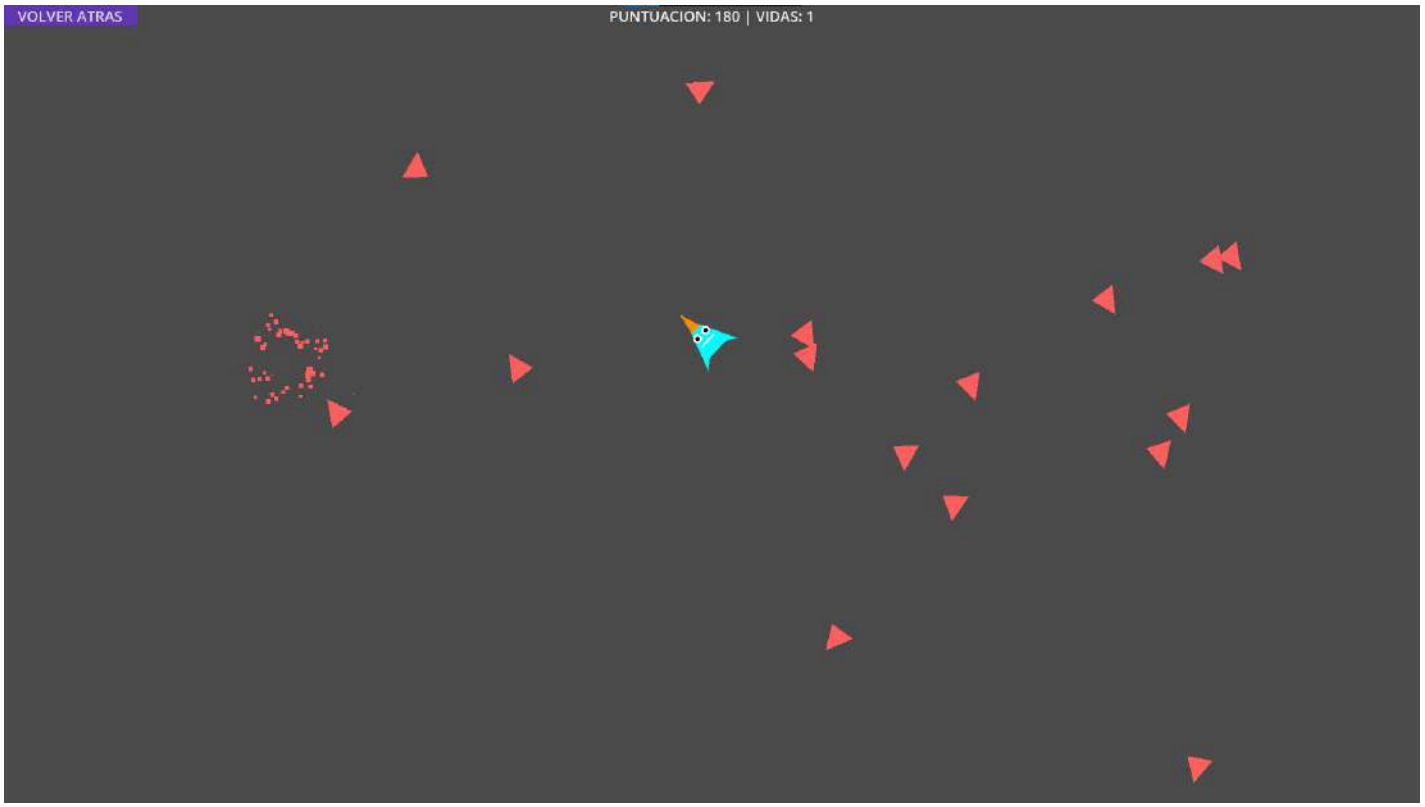
Scripts:

TileMap.gd: script principal del Tetris. S'encarrega de:

- Definir les 7 peces clàssiques del Tetris cadascuna amb les seves 4 rotacions possibles, representades com arrays de posicions Vector2i
- Gestionar un tauler de 10 columnes × 20 files
- Registrar els controls del teclat (WASD) dinàmicament si no existeixen al InputMap
- Seleccionar peces aleatòriament amb un sistema de bossa (bag) que garanteix que totes les peces apareguin abans de repetir-se
- Moure la peça activa cap a l'esquerra, la dreta i avall, i rotació cap amunt
- Detectar si un moviment o rotació és vàlid comprovant si les cel·les destí estan lliures i dins dels límits del tauler
- Quan la peça no pot baixar més, la fixa al tauler (capa 0), comprova si hi ha files completes, les elimina i desplaça les files superiors cap avall, sumant 100 punts per fila eliminada
- Augmentar la velocitat de caiguda progressivament amb cada fila eliminada ***(speed += 0.25)***
- Mostrar la peça següent en un panell lateral
- Detectar el final de la partida quan una peça nova no pot col·locar-se, guardar la puntuació amb ***Puntuacion.save_score("tetris", score)*** i mostrar el ***GameOverLabel***

Button.gd: script del botó de sortida. Restaura la resolució de la finestra a 1920×1080 i torna a l'escena de selecció de jocs.

SPACE INVADERS GUAU



Estructura de nodes:

Game (Node2D): node arrel que conté tota la lògica de l'escena i gestiona l'aparició dels enemics. Conté el script principal del joc.

WorldEnvironment: node que defineix l'entorn visual de la pantalla, incloent el color de fons fosc característic del joc.

Player (CharacterBody2D): node que representa la nau del jugador, dibuixada proceduralment mitjançant codi. Conté dos nodes fills:

- **CollisionShape2D:** defineix la forma de col·lisió de la nau per detectar el contacte amb els enemics.
- **ShootyPart:** punt de referència que indica des d'on surten les bales disparades.

Walls (Node): node contenidor dels límits invisibles de la pantalla, format per quatre nodes fills:

- Top, Left, Right, Bottom (StaticBody2D): parets que delimiten l'àrea de joc i impedeixen que el jugador surti de la pantalla.

Timer: temporitzador que controla la freqüència d'aparició dels enemics. Comença a 0.5 segons i es redueix progressivament fins a un mínim de 0.2 segons.

CanvasLayer: capa d'interfície que es mostra per sobre del joc i conté:

- **ScoreLabel:** etiqueta que mostra la puntuació actual i les vides restants en temps real

Button: botó per tornar a la selecció de jocs.

Scripts:

Game.gd: script principal del joc. S'encarrega de:

- Carregar i instanciar els enemics dinàmicament cada vegada que el Timer s'esgota, generant-los en posicions aleatòries de la pantalla sempre a més de 150 píxels del jugador
- Generar 2 enemics per tick del temporitzador
- Reduir el temps entre aparicions en 0.01 segons cada tick, augmentant progressivament la dificultat fins al mínim de 0.2 segons
- Actualitzar el ScoreLabel cada frame amb la puntuació i les vides actuals obtingudes del ScoreManager global

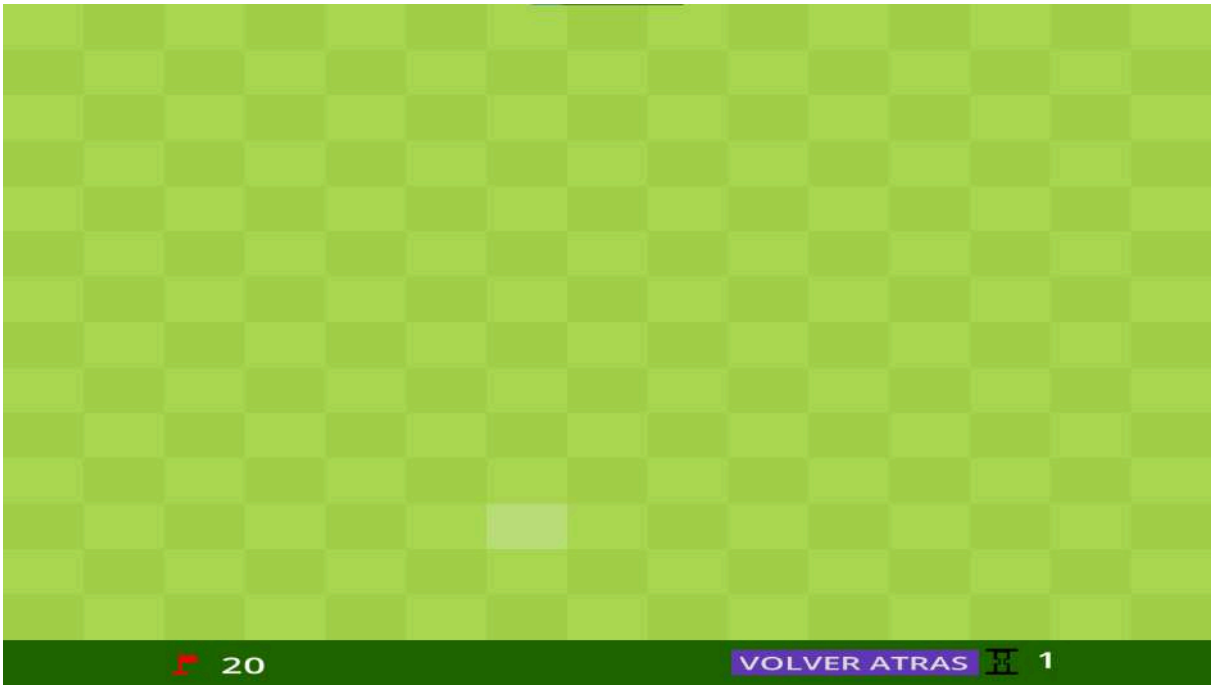
Player.gd: script de la nau del jugador. S'encarrega de:

- Moure la nau amb les tecles WASD a una velocitat de 300 píxels per segon
- Rotar la nau cap al cursor del ratolí en tot moment
- Disparar bales cap al cursor en fer clic esquerre, amb un temps de recàrrega de 0.2 segons entre dispars
- Detectar col·lisions amb enemics del grup enemies i cridar ***take_damage()***

- En rebre dany, cridar ***ScoreManager.lose_life()*** per reduir les vides. Si no en queden, guarda la puntuació amb ***Puntuacion.save_score("space_invaders", score)*** i reinicia el joc
- Dibuixar la nau proceduralment amb _draw() usant polígons de color cian i taronja, sense necessitat de cap imatge externa

Button.gd: script del botó de sortida. Restaura la resolució a 1920×1080 i torna a l'escena de selecció de jocs.

BUSCA MINAS GUAU



Estructura de nodes:

Main (Node): node arrel que coordina tota la lògica global del joc: el temps, les mines restants, l'estat de la primera jugada i la gestió del final de partida.

Background (TextureRect): node que mostra la imatge de fons de l'escena.

TileMap: node principal del tauler de joc. Gestiona cinc capes de tessel·les superposades:

- **Capa 0 (mine_layer):** conté la posició de les mines ocultes
- **Capa 1 (number_layer):** conté els números que indiquen quantes mines hi ha al voltant de cada cel·la
- **Capa 2 (grass_layer):** capa visible que cobreix totes les cel·les fins que el jugador les descobreix
- **Capa 3 (flag_layer):** conté les banderes col·locades pel jugador amb el clic dret
- **Capa 4 (hover_layer):** ressalta visualment la cel·la sobre la qual passa el cursor

HUD (CanvasLayer): capa d'interfície que mostra la informació de la partida i conté:

- **Panel:** panell visual de fons
- **Flag:** icona de bandera
- **RemainingMines:** etiqueta que mostra el nombre de mines restants
- **Time:** etiqueta informativa del temps
- **Stopwatch:** etiqueta que mostra el temps transcorregut en segons
- **VOLVER ATRAS (Button):** botó per tornar a la selecció de jocs

GameOver (CanvasLayer): capa que apareix en acabar la partida, ja sigui per guanyar o per explotar, i conté:

- **RestartButton:** botó per reiniciar la partida, emet el senyal restart
- **Panel:** panell visual de fons
- **Label:** etiqueta que mostra "YOU WIN!" o "BOOM!" segons el resultat

Scripts:

Main.gd: script principal que gestiona:

- Les variables globals del joc: 20 mines totals, temps transcorregut i mines restants
- El reinici complet de la partida amb ***new_game()***
- L'actualització cada frame del cronòmetre i el comptador de mines al HUD
- La recepció dels senyals del TileMap: ***flag_placed*** i ***flag_removed*** per actualitzar el comptador de mines, ***end_game*** per mostrar "BOOM!" i ***game_won*** per mostrar "YOU WIN!"
- La pausa de l'arbre de nodes quan la partida acaba

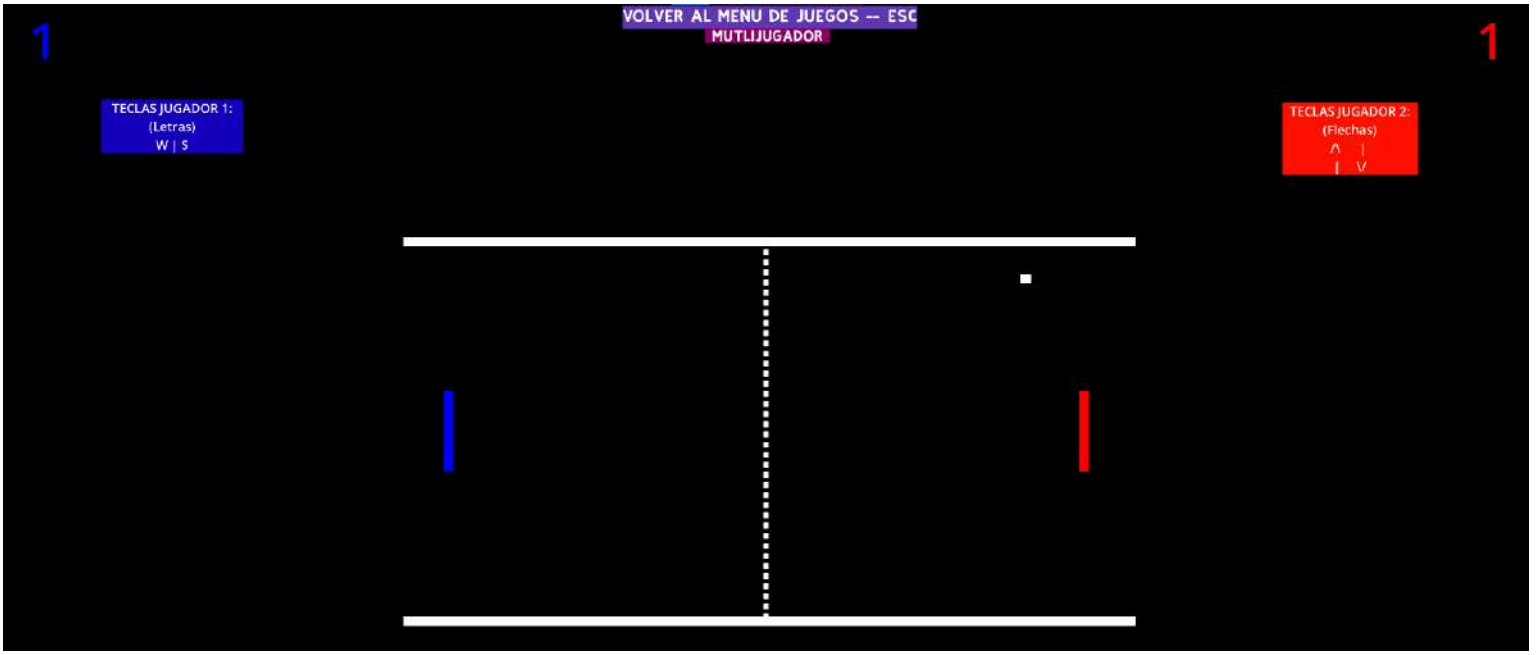
TileMap.gd: script del tauler de joc. S'encarrega de:

- Generar aleatòriament 20 mines al tauler de 15×14 cel·les sense repetir posicions
- Calcular i dibuixar els números de cada cel·la buida comptant les mines als 8 veïns
- Generar la capa de gespa amb un patró en escaquer visual
- Gestionar el clic esquerre: descobreix la cel·la i, si és buida, desencadena una revelació en cascada de totes les cel·les adjacents buides
- Gestionar el clic dret: col·loca o elimina una bandera a la cel·la
- Si el primer clic del jugador cau sobre una mina, la mou a una cel·la lliure per garantir una primera jugada sempre segura
- Ressaltar la cel·la sota el cursor en cada frame
- Detectar la victòria quan totes les cel·les amb número han estat descobertes
- Permetre escanejar mines amb clic esquerre + dret simultanis sobre una cel·la descoberta

VOLVER ATRAS.gd: script del botó de sortida. Restaura la resolució a 1920×1080 i torna a la selecció de jocs.

GameOver.gd: emet el senyal restart quan el jugador prem RestartButton, que el Main recull per reiniciar la partida.

PING PONG GUAU



Estructura de nodes:

main (Node2D): node arrel que coordina tota la lògica del joc: puntuació, detecció de gols i reinici de partida.

ColorRect: node que proporciona el fons negre de la pantalla de joc.

VOLVER AL MENU (Button): botó per tornar al menú de selecció de jocs prement ESC.

MULTIJUGADOR (Label): etiqueta informativa que indica que el joc és per a dos jugadors.

TECLAS JUGADOR 1 (Label): etiqueta que mostra els controls del jugador 1 (W/S).

TECLAS JUGADOR 2 (Label): etiqueta que mostra els controls del jugador 2.
Paddle (RigidBody2D): node de la pala del jugador 1. Conté:

- Sprite2D: representació visual de la pala
- CollisionShape2D: forma de col·lisió de la pala

EnemyPaddle (RigidBody2D): node de la pala del jugador 2. Té exactament la mateixa estructura i el mateix script que Paddle.

Camera2D: càmera que centra la vista del joc.

Walls (Node): contenidor de les parets superior i inferior:

- **TopWall (StaticBody2D):** paret superior amb Sprite2D i CollisionShape2D
- **BottomWall (StaticBody2D):** paret inferior amb Sprite2D i CollisionShape2D

Ball (CharacterBody2D): node de la pilota. Conté:
- **Sprite2D:** representació visual de la pilota
- **CollisionShape2D:** forma de col·lisió de la pilota
- **AudioStreamPlayer:** reprodueix un so cada vegada que la pilota rebota contra una pala

Line2D: línia central vertical que divideix el camp visualment.

LeftEdge (Area2D): àrea invisible al costat esquerre de la pantalla. Quan la pilota hi entra, emet el senyal point_scored per indicar que el jugador 2 ha marcat. Conté CollisionShape2D.

RightEdge (Area2D): àrea invisible al costat dret de la pantalla. Quan la pilota hi entra, emet el senyal point_scored per indicar que el jugador 1 ha marcat. Conté CollisionShape2D.

UI (CanvasLayer): capa d'interfície que mostra la puntuació. Conté:
- **MarginContainer:** contenidor que organitza les etiquetes de puntuació
- **PlayerPoints:** etiqueta amb la puntuació del jugador 1
- **EnemyPoints:** etiqueta amb la puntuació del jugador 2

Scripts:

main.gd: script principal del joc. S'encarrega de:
- Mantenir la puntuació dels dos jugadors
- Quan LeftEdge o RightEdge detecten la pilota, cridar ***enemy_scored()*** o ***player_scored()*** respectivament
- Sumar 1 punt al jugador corresponent i actualitzar la ***UI***
- Quan un jugador arriba a 7 punts, mostrar el missatge de victòria ("¡JUGADOR 1 GANA!" o "¡JUGADOR 2 GANA!")
- Reiniciar les posicions de les pales i la pilota entre punts amb ***reset_game_state()***
- Permetre tornar a la selecció de jocs prement ESC en qualsevol moment

Paddle.gd: script compartit per Paddle i EnemyPaddle. Mou la pala verticalment amb les tecles W (amunt) i S (avall) a una velocitat de 800 unitats per segon usant RigidBody2D amb ***linear_velocity***.

Ball.gd: script de la pilota. S'encarrega de:
- Iniciar la pilota en una direcció aleatòria horitzontal i vertical
- Moure's cada frame amb ***move_and_collide()***
- En col·lisionar, rebotar usant el vector normal de la col·lisió i multiplicar la velocitat per 1.02, fent que la pilota s'acceleri progressivament amb cada rebot
- Reproduir el so de rebot quan col·lisiona amb una pala

LeftEdge.gd / RightEdge.gd: script de les àrees de gol. Quan detecten que la pilota (Ball) entra al seu interior, emeten el senyal ***point_scored*** que el main recull per sumar el punt corresponent.

UI.gd: script de la interfície. Gestiona les etiquetes de puntuació dels dos jugadors i, quan un jugador guanya, crea dinàmicament una etiqueta gran de color groc centrada a la pantalla amb el missatge de victòria.

PLANE SHOOTER GUAU

Estructura de nodes:

MainScene (Node2D): node arrel que conté tots els elements del joc.

Camera2D: càmera que segueix l'acció i centra la vista del joc.

Background (Sprite2D): node que mostra el fons del joc amb un desplaçament vertical continu per simular el moviment de l'avió cap amunt.

Player (Area2D): node que representa l'avió del jugador. Conté:

- **Sprite2D:** representació visual de l'avió
- **CollisionPolygon2D:** forma de col·lisió poligonal de l'avió
- **AnimationPlayer:** gestiona les animacions de l'avió
- **GunFlash (Node2D):** efecte visual de flaix en disparar, amb dos nodes fills 1 i 2 que representen els punts de flaix de cada canó
- **Shooter (Node):** node que gestiona el sistema de disparament, amb dos nodes fills:
    - **GunPoint_0 (Marker2D):** punt de sortida de les bales del canó esquerre
    - **GunPoint_1 (Marker2D):** punt de sortida de les bales del canó dret
    - **AudioStreamPlayer2D:** reprodueix el so de disparament

Shredder1 i Shredder2 (Area2D): àrees invisibles als límits de la pantalla que destrueixen automàticament qualsevol projectil o element que en surti, evitant que s'acumulin fora del camp de joc. Cadascun conté un CollisionShape2D.

Spawner (Node2D): node que gestiona l'aparició dels enemics durant la partida.

GameUI (CanvasLayer): capa d'interfície completa que conté tots els panells del joc:

- **HUD (Node):** interfície principal durant la partida:
  - **PlayerHealth (ProgressBar):** barra de vida del jugador
  - **CoinCounter (Panel):** comptador de monedes amb icona Coin i etiqueta Label
  - **PauseButton (Button):** botó per pausar la partida
  - **WaveText:** text que mostra l'onada actual amb AnimationPlayer

- **StartScreen (Panel):** pantalla inicial abans de començar la partida:
  - **Title:** títol del joc amb AnimationPlayer
  - **Start (Button):** botó per iniciar la partida
  - **Exit (Button):** botó per sortir al menú de selecció
  - **TECLAS:** node informatiu amb els controls

- **GameComplete (Panel):** pantalla de victòria:
  - **Title:** títol amb AnimationPlayer
  - **CoinCounter:** resum de monedes recollides
  - **NextLevel (Button):** botó per avançar al nivell següent
  - **Exit (Button):** botó per sortir

- **Pause (Panel):** pantalla de pausa:
  - **Title:** títol amb AnimationPlayer
  - **Resume (Button):** botó per reprendre la partida
  - **Exit (Button):** botó per sortir

Scripts:

Background.gd: desplaça la regió de la textura de fons verticalment cada frame a una velocitat de 100 unitats per segon, creant la il·lusió de moviment continu cap amunt.

Player.gd: script principal del jugador. S'encarrega de:

- Moure l'avió amb les tecles de fletxa o arrossegant amb el ratolí, limitat als marges de la pantalla amb un padding de 80px
- Detectar col·lisions amb bales enemigues (EnemyBullet), monedes (Coin) i potenciadors (PowerUp)
- En rebre una bala enemiga, reduir la barra de vida i reproduir el so de dany. Si la vida arriba a 0, genera l'efecte d'explosió i crida ***game_ui.game_over()***
- En recollir una moneda, sumar 1 al comptador i reproduir el so corresponent
- En recollir un potenciador, restaurar la vida completa

Shooter.gd: script del sistema de disparament, compartit entre el jugador i els enemics. S'encarrega de:

- Localitzar els GunPoint (Marker2D) fills per saber des d'on disparar

- Disparar bales des de cada GunPoint en prémer espai o clic esquerre (jugador) o automàticament cada 0.2 segons (enemics)
- Mostrar el flaix del canó durant 0.1 segons en cada disparo
- Reproduir el so de disparament

Shredder.gd: script de les àrees de neteja. Quan qualsevol element entra a la seva àrea, el destrueix immediatament amb ***queue_free()***.

Spawner.gd: script del generador d'enemics. S'encarrega de:
- Carregar les dades del nivell actual des d'un fitxer de recursos (level_X.tres) que defineix el tipus d'enemics, la quantitat i el temps entre aparicions
- Generar enemics aleatoris en posicions horitzontals aleatòries dins d'un rang de ±260px
- Quan s'han generat tots els enemics del nivell, comprovar si queden enemics vius. Si no en queda cap i el jugador segueix viu, cridar ***game_ui.game_complete()***. Si el jugador ha mort, cridar ***game_ui.game_over()***

GameUI.gd: script de tota la interfície. S'encarrega de:
- Mostrar i amagar les pantalles d'inici, pausa, victòria i derrota
- Actualitzar el comptador de monedes i la barra de vida en temps real
- En finalitzar la partida (victòria o derrota), guardar la puntuació amb ***Puntuacion.save_score("plane_shooter", count)*** i pausar el joc
- Gestionar la sortida al menú de selecció de jocs i el pas al nivell següent

BILLAR GUAU

Estructura de nodes:

Main (Node): node arrel que coordina tota la lògica del joc: generació de boles, detecció d'embocades, puntuació i gestió del final de partida.

Table (StaticBody2D): node que representa la taula de billar. Conté dos grups de nodes fills:

- **Cushions (StaticBody2D):** les bandes de la taula, formades per sis CollisionPolygon2D que defineixen la forma irregular de cada banda per fer rebotar les boles de manera realista
- **Pockets (Area2D):** les troneres de la taula, formades per sis CollisionShape2D (una per cada tronera). Quan una bola hi entra, es crida la funció ***potted_ball()***

PottedPanel: panell visual que mostra les boles embocades al llarg de la partida.

Cue (Sprite2D): node que representa el pal de billar. Rota cap al cursor del ratolí i gestiona el disparament. Conté el script de control del tac.

PowerBar (ProgressBar): barra visual que mostra la potència del tir actual mentre el jugador manté el clic.

LivesLabel (Label): etiqueta que mostra les vides restants del jugador.

ShotsLabel (Label): etiqueta que mostra el nombre total de tirs realitzats.

Hud (CanvasLayer): capa d'interfície que apareix en acabar la partida i conté:

- **ResultPanel (Panel):** panell de resultat final amb:
  - **ResultLabel:** etiqueta que mostra "YOU WIN!" o "GAME OVER!"
  - **RestartButton:** botó per reiniciar la partida

VOLVER AL MENU (Button): dos botons per tornar a la selecció de jocs, un accessible durant la partida i l'altre des del menú de fi de partida.

Scripts:

Main.gd: script principal del joc. S'encarrega de:

- Carregar les 16 textures de les boles des dels recursos del projecte
- Generar les 15 boles de joc en formació triangular de 5 columnes al centre-esquerra de la taula, assignant a cada bola el seu número i textura corresponent
- Crear la bola blanca (***cue_ball***) en la posició inicial (890, 340)
- Cada frame, comprovar si alguna bola segueix en moviment. Quan totes s'aturen, mostrar el tac i la barra de potència perquè el jugador pugui fer el següent tir
- Quan el jugador dispara (***_on_cue_shoot***), aplicar un impuls a la bola blanca i sumar 1 al comptador de tirs
- Quan una bola entra a una tronera (***potted_ball***):
  - Si és la bola blanca: restar una vida, eliminar-la i tornar-la a col·locar al reiniciar. Si les vides arriben a 0, derrota

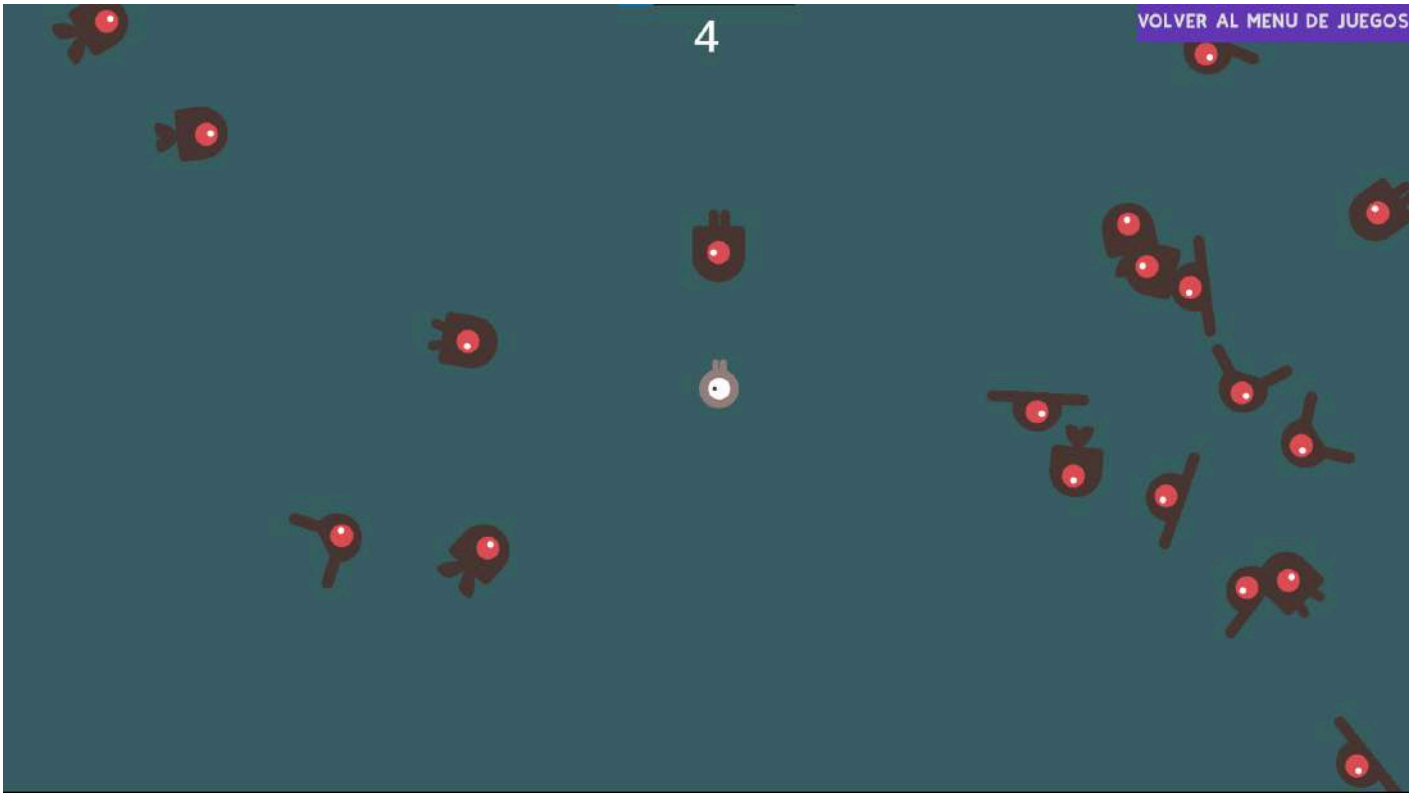
    - Si és la bola negra (número 8): victòria si s'han embocinat totes les lises o les ratllades, derrota en cas contrari
    - Si és una bola lisa (1-7) o ratllada (9-15): sumar al comptador corresponent i mostrar-la al ***PottedPanel***
- Guardar la puntuació i mostrar el Hud en acabar la partida
- Permetre tornar al menú de selecció prement ESC en qualsevol moment

Cue.gd: script del tac de billar. S'encarrega de:

- Rotar el tac cap a la posició del cursor del ratolí en cada frame amb ***look_at()***
- Mentre el jugador manté el clic esquerre, augmentar i disminuir la potència de manera oscil·lant entre 0 i MAX_POWER (8.0)
- En alliberar el clic, calcular la direcció del tir i emetre el senyal shoot amb la força resultant ***(dir * power * 200)***, que el Main recull per aplicar l'impuls a la bola blanca

PowerBar.gd: script de la barra de potència. Actualitza el seu valor cada frame llegint la potència actual del node Cue i convertint-la a un percentatge de 0 a 100.

DODGE THE CREEPS GUAU



Estructura de nodes:

Main (Node): node arrel que coordina tota la lògica del joc: generació de mobs, puntuació i gestió del final de partida.

ColorRect: node que proporciona el fons de color de la pantalla de joc.

Player (Area2D): node que representa el personatge del jugador. Conté:
- **AnimatedSprite2D:** gestiona les animacions del personatge segons la direcció de moviment (dreta, esquerra, amunt, avall)
- **CollisionShape2D:** forma de col·lisió del personatge per detectar el contacte amb els mobs
- **Trail:** efecte visual de rastre darrere del personatge en moviment

MobTimer (Timer): temporitzador que controla cada quant temps apareix un mob nou. Quan s'esgota, crida ***_on_MobTimer_timeout()*** per generar un enemic nou.

ScoreTimer (Timer): temporitzador que suma 1 punt al marcador cada segon que el jugador sobreviu.

StartTimer (Timer): temporitzador d'espera inicial que s'activa en iniciar una nova partida. Quan s'esgota, inicia el MobTimer i el ScoreTimer.

StartPosition (Marker2D): node que marca la posició inicial del jugador al centre de la pantalla.

MobPath (Path2D): camí que defineix el perímetre de la pantalla per on apareixen els mobs. Conté:
- **MobSpawnLocation (PathFollow2D):** node que recorre el camí en posicions aleatòries per determinar el punt d'aparició de cada mob

HUD (CanvasLayer): capa d'interfície que mostra la informació del joc. Conté:
- **ScoreLabel:** etiqueta que mostra la puntuació actual en temps real
- **MessageLabel:** etiqueta que mostra missatges com "¡Prepárate!" o "Game Over"
- **StartButton (Button):** botó per iniciar o reiniciar la partida, emet el senyal start_game
- **vovler (Button):** botó per tornar a la selecció de jocs
- **MessageTimer (Timer):** temporitzador que controla quant temps es mostra el MessageLabel

Music (AudioStreamPlayer): node que reprodueix la música de fons durant la partida.

DeathSound (AudioStreamPlayer): node que reprodueix el so de mort quan el jugador col·lisiona amb un mob.

Scripts:

Main.gd: script principal del joc. S'encarrega de:
- Gestionar el flux complet de la partida amb ***new_game()*** i ***game_over()***
- En iniciar una nova partida, eliminar tots els mobs existents del grup mobs, reiniciar la puntuació a 0, col·locar el jugador a StartPosition, iniciar el StartTimer i mostrar el missatge "¡Prepárate!"
- Cada vegada que el MobTimer s'esgota, instanciar un mob nou, col·locar-lo en una posició aleatòria del MobPath, assignar-li una direcció

perpendicular al camí amb variació aleatòria de ±45°, i donar-li una velocitat aleatòria entre 150 i 250 unitats per segon
- Cada segon que el ScoreTimer s'esgota, sumar 1 punt i actualitzar el HUD
- En finalitzar la partida, aturar els temporitzadors, guardar la puntuació amb ***Puntuacion.save_score("dodge_the_creeps", score)***, mostrar la pantalla de game over, aturar la música i reproduir el so de mort

Player.gd: script del jugador. S'encarrega de:
- Moure el personatge amb les tecles de fletxa o WASD a 400 píxels per segon, normalitzant la velocitat en moviments en diagonal
- Limitar el moviment als límits de la pantalla amb clamp()
- Canviar l'animació segons la direcció: animació right per al moviment horitzontal (amb ***flip_h*** per a l'esquerra) i animació up per al moviment vertical (amb rotació de 180° per al moviment cap avall)
- Quan un mob toca el jugador, amagar-lo, emetre el senyal hit i desactivar la col·lisió de manera diferida per evitar deteccions múltiples simultànies

HUD.gd: script de la interfície. S'encarrega de:
- Mostrar missatges temporals amb ***show_message()***, que els amaga automàticament quan el MessageTimer s'esgota
- En game over, mostrar "Game Over", esperar que el MessageTimer s'esgoti, mostrar el títol "Dodge the Creeps" un segon més i finalment mostrar el StartButton
- Quan el jugador prem StartButton, amagar-lo i emetre el senyal ***start_game*** que el Main recull per iniciar una nova partida

vovler.gd: script del botó de sortida. Restaura la resolució a 1920×1080 i torna a la selecció de jocs.

PUNTACIONES

Estructura de nodes:
Puntuaciones (Control): node arrel de la pantalla que mostra les puntuacions màximes de tots els jocs.
TextureRect: node que mostra la imatge de fons de la pantalla.
VOLVER (Button): botó per tornar a la selecció de jocs.
SNAKE GUAU (Label): etiqueta que mostra la puntuació màxima del Snake.
TETRIS GUAU (Label): etiqueta que mostra la puntuació màxima del Tetris.
PLANE SHOOTER GUAU (Label): etiqueta que mostra la puntuació màxima del Plane Shooter.
BILLAR GUAU (Label): etiqueta que mostra la puntuació màxima del Billar.
DOGDE THE CREEPS GUAU (Label): etiqueta que mostra la puntuació màxima del Dodge the Creeps.
SPACE INVADERS GUAU (Label): etiqueta que mostra la puntuació màxima del Space Invaders.

Scripts:
puntuaciones.gd: script de la pantalla de puntuacions. En carregar-se l'escena (***_ready()***), llegeix la puntuació màxima de cada joc cridant ***Puntuacion.get_score()*** amb el nom del joc corresponent i actualitza el text de cada etiqueta.

VOLVER.gd: script del botó de sortida. Restaura la resolució a 1920×1080 i torna a la selecció de jocs.

AUTOLOAD: Puntuacion.gd
Aquest script és un Autoload (singleton global) de Godot Engine 4, és a dir, s'executa automàticament en iniciar el joc i és accessible des de qualsevol escena sense necessitat d'instanciar-lo. S'encarrega de tot el sistema de guardat i recuperació de puntuacions del projecte.

Utilitza un fitxer ConfigFile guardat a ***user://scores.cfg***, que és una ubicació local al dispositiu de l'usuari gestionada automàticament per Godot.

Té tres funcions principals:
load_scores(): en iniciar-se, intenta carregar el fitxer de puntuacions. Si existeix, llegeix totes les seccions (una per joc) i emmagatzema els valors al diccionari scores en memòria.

save_score(game, new_score, lower_is_better): quan un joc acaba, comprova si la nova puntuació supera la millor registrada. Si és millor (o si no n'hi havia cap), actualitza el diccionari i sobreescriu el fitxer ***scores.cfg***. El paràmetre ***lower_is_better*** permet adaptar el sistema a jocs on una puntuació més baixa és millor (com podria ser un temps).

get_score(game): retorna la puntuació màxima d'un joc concret. Si el joc no té cap puntuació guardada, retorna 0 per defecte.

1 Introducció

Multi Games GUAU és un projecte de desenvolupament d'un conjunt de videojocs arcade inspirats en clàssics del gènere, creat íntegrament amb el motor de videojocs Godot Engine 4 i el seu llenguatge de programació GDScript. El projecte consisteix en vuit jocs independents (Snake, Tetris, Pong, Space Invaders, Busca Mines, Billar, Plane Shooter i Dodge the Creeps) accessibles des d'un menú principal comú, cadascun amb les seves pròpies mecàniques, interfície i sistema de puntuació màxima persistent entre sessions.

L'objectiu principal és oferir un producte funcional, estable i accessible per a qualsevol tipus d'usuari, independentment del seu nivell d'experiència amb els videojocs. Per aconseguir-ho, s'ha desenvolupat cada joc de manera independent però seguint una estructura tècnica comuna, compartint el sistema de guardat de puntuacions mitjançant un Autoload global i garantint una navegació coherent entre tots els jocs a través del menú principal.

Cal destacar que el projecte va néixer d'un canvi d'orientació respecte a la idea original. Inicialment es volia desenvolupar Pokemon Classic, un joc de rol en 2D amb mapa del món, batalles per torns i progressió de personatge. Després d'analitzar la viabilitat tècnica real del projecte, es va constatar que la seva complexitat superava els recursos i el temps disponibles, i es va prendre la decisió de reorientar el treball cap a Multi Games GUAU, una proposta més assequible però igualment completa i tècnicament rica.

1.1 Context

Els videojocs arcade clàssics van néixer als anys 70 i 80 amb títols com Space Invaders (1978), Snake (1976), Tetris (1984) o Pong (1972), i van definir les bases del que avui coneixem com a disseny de videojocs: mecàniques simples, aprenentatge immediat i dificultat progressiva. Malgrat les dècades transcorregudes, aquests jocs continuen sent una referència en el sector i es fan servir habitualment en contextos educatius de programació per la seva estructura clara i la seva complexitat tècnica accessible.

Avui dia, gràcies a motors de videojocs moderns gratuïts i de codi obert com Godot Engine 4, és possible recrear i adaptar aquests clàssics sense necessitat de grans recursos econòmics ni d'eines de pagament. Godot Engine 4 ofereix un entorn de desenvolupament integrat amb editor visual, sistema de nodes i escenes, físiques 2D i un llenguatge de programació propi (GDScript) de sintaxi accessible, cosa que el converteix en una eina ideal per a projectes educatius.

El projecte neix en aquest context: aprofitar les possibilitats que ofereixen les eines actuals per desenvolupar un conjunt de videojocs arcade originals que serveixin alhora com a producte final funcional i com a exercici pràctic d'aplicació de coneixements de programació, disseny d'interfícies i gestió de projectes.

1.2 Justificació

El projecte es justifica per diversos motius de naturalesa tècnica, acadèmica i pràctica.

En primer lloc, el canvi de Pokemon Classic a Multi Games GUAU va suposar un exercici real de gestió de projecte. Identificar que un plantejament inicial és tècnicament inviable dins dels recursos disponibles, i saber reorientar el treball de manera ràpida i organitzada, és una competència fonamental en qualsevol àmbit professional del sector informàtic. Aquesta situació va obligar a reorganitzar la planificació, redistribuir les tasques i ajustar els objectius en un termini curt mantenint la qualitat del resultat final.

En segon lloc, Multi Games GUAU és un projecte tècnicament ric i variat. Cada joc planteja reptes de programació diferents: gestió de col·lisions, control del flux del programa, generació aleatòria d'elements, simulació de física, sistemes de puntuació, dificultat progressiva i guardat persistent de dades. Aquesta varietat ha permès aplicar i consolidar un conjunt ampli de coneixements de programació de manera pràctica.

En tercer lloc, el projecte demostra que és possible desenvolupar un producte de qualitat amb cost econòmic pràcticament nul, fent ús exclusivament d'eines gratuïtes i de codi obert. Això el fa replicable i accessible per a qualsevol estudiant o desenvolupador que vulgui aprendre desenvolupament de videojocs.

Finalment, Multi Games GUAU és un producte accessible i usable per a qualsevol persona, cosa que demostra que un bon disseny d'interacció és tan important com la programació en el desenvolupament d'un producte de qualitat.

1.3 Objectius

Objectiu principal: L'objectiu inicial del nostre projecte era desenvolupar un videojoc en 2D anomenat Pokemon Classic, inspirat en els jocs originals de Pokémon de Nintendo. Volíem recrear l'experiència clàssica amb mapa del món transitable, sistema de batalles per torns i progressió de personatge. No obstant això, durant el desenvolupament vam comprovar que la complexitat tècnica del projecte era massa elevada per completar-lo correctament dins del temps del que disposàvem. Per aquest motiu, dos mesos i mig abans de la data de lliurament, vam decidir canviar l'enfocament i desenvolupar Multi Games GUAU, un projecte més assequible però igualment complet i funcional.

Producte final: Desenvolupar un conjunt de videojocs arcade sota el nom Multi Games GUAU, tots ells funcionals, jugables i accessibles des d'un menú principal unificat, amb una interfície clara i un sistema de guardat de puntuació màxima.

Coneixements aplicats: Programació amb GDScript, disseny d'interfícies d'usuari, creació de mecàniques de joc, gestió d'escenes amb Godot Engine 4, control de versions amb GitHub Desktop i organització de tasques amb Padlet.

Resultat esperat: Aconseguir un conjunt de jocs estables, fàcils d'utilitzar i entretinguts, que puguin ser jugats de manera independent i sense errors importants, tots accessibles des d'un menú principal comú.

1.3.1 Objectiu general

Desenvolupar un conjunt de videojocs arcade originals sota el nom Multi Games GUAU, inspirats en jocs clàssics del gènere, que siguin funcionals, estables i entretinguts, i que ofereixin una experiència de joc variada i accessible des d'un mateix entorn unificat, amb un sistema de guardat de puntuació màxima persistent entre sessions.

1.3.2 Objectius específics

- Dissenyar i implementar vuit jocs arcade (Snake, Tetris, Pong, Space Invaders, Busca Mines, Billar, Plane Shooter i Dodge the Creeps) amb un funcionament correcte i sense errors greus.
- Programar les mecàniques de joc de cada títol: moviment dels elements, detecció de col·lisions, sistemes de puntuació i comportament dels elements de joc.
- Implementar sistemes de dificultat progressiva en els jocs que ho permetin, per millorar l'experiència del jugador a mesura que avança la partida.
- Desenvolupar un menú principal i un menú de selecció de jocs amb una interfície clara, intuïtiva i fàcil d'utilitzar.
- Implementar un sistema global de guardat de puntuació màxima mitjançant un Autoload, de manera que les dades es conservin entre sessions de joc.
- Garantir l'estabilitat del projecte, evitant errors crítics que puguin interrompre les partides.
- Organitzar el projecte de manera eficient, repartint les tasques entre els dos membres de l'equip i fent ús d'eines de planificació per optimitzar el temps de desenvolupament.

1.4 Estratègia i planificació del projecte

Per dur a terme el projecte es van valorar dues estratègies principals.

La primera opció era partir de codi font existent disponible en repositoris públics i adaptar-lo a les necessitats del projecte. Aquesta estratègia hauria accelerat el procés de desenvolupament, però hauria limitat l'aprenentatge tècnic i hauria dificultat la integració de tots els jocs dins d'un entorn comú amb un sistema de puntuacions compartit.

La segona opció era desenvolupar tots els jocs des de zero amb codi propi. Aquesta estratègia requereix més temps, però ofereix control total sobre l'estructura del codi, les mecàniques de cada joc i la integració entre escenes.

Es va escollir la segona opció perquè s'ajustava millor als objectius acadèmics del projecte i permetia aplicar i demostrar els coneixements adquirits durant el curs de manera autèntica.

Pel que fa a la planificació, el desenvolupament es va dividir en cinc fases: investigació i configuració de l'entorn, disseny i prototipatge, desenvolupament de la lògica i les interfícies, proves i correcció d'errors, i integració final de tots els jocs en el menú principal. El canvi de Pokemon Classic a Multi Games GUAU va obligar a replanejar completament les dues primeres fases a mitja durada del projecte, cosa que va reduir el marge de temps disponible per al desenvolupament però no va impedir assolir els objectius finals.

1.5 Metodologia de treball

Per al desenvolupament de Multi Games GUAU es va adoptar una metodologia àgil, concretament un enfocament basat en iteracions curtes similar a Scrum, en lloc d'una metodologia tradicional tipus waterfall.

La metodologia àgil es va considerar la més adequada per diversos motius. En primer lloc, el projecte tenia un abast variable i subjecte a canvis, com va quedar demostrat amb el canvi de Pokemon Classic a Multi Games GUAU. Una metodologia tradicional, on tota la planificació es tanca al principi, hauria fet molt difícil absorbir aquest canvi sense desorganitzar completament el treball. En segon lloc, el desenvolupament de videojocs és un procés inherentment iteratiu: cada mecànica s'implementa, es prova i es corregeix abans de continuar, cosa que encaixa perfectament amb els cicles curts de la metodologia àgil.

Per al seguiment del projecte es van utilitzar les eines següents. Padlet va servir per gestionar les tasques de cada membre de l'equip, assignar responsabilitats i fer un seguiment visual del progrés. El diagrama de Gantt va permetre planificar els temps de cada fase i controlar que el projecte avançava dins del calendari previst. GitHub Desktop va ser l'eina de control de versions, registrant tots els canvis realitzats i permetent recuperar versions anteriors en cas d'error. Documents compartits van centralitzar la informació, les decisions i els canvis del projecte, accessibles per als dos membres en qualsevol moment.

1.6 Estudi econòmic i pressupostari

Per dur a terme el desenvolupament de Multi Games GUAU, vam analitzar els recursos necessaris i els costos associats al projecte. En tractar-se d'un projecte educatiu desenvolupat per dos estudiants, la majoria dels recursos utilitzats han estat gratuïts o ja estaven disponibles sense cost addicional.

Pel que fa al programari, vam utilitzar Godot Engine 4 com a motor de desenvolupament principal, que és completament gratuït i de codi obert. GitHub Desktop, que vam fer servir per al control de versions, també és gratuït per a ús personal i educatiu. Padlet, l'eina que vam utilitzar per organitzar les tasques, disposa d'una versió gratuïta que va ser suficient per a les nostres necessitats. Per tant, el cost total en programari va ser de zero euros.

Pel que fa al maquinari, vam fer servir els ordinadors personals de cada membre de l'equip i els ordinadors disponibles a l'aula durant les hores de classe. No va ser necessari adquirir cap component de maquinari addicional, ja que Godot Engine 4 funciona correctament en ordinadors estàndard sense necessitat de targetes gràfiques especialitzades ni processadors d'alta gamma.

Si haguéssim de calcular el cost real del projecte tenint en compte el temps dedicat, podríem estimar-lo en funció de les hores de treball invertides pels dos membres de l'equip. Considerant un total aproximat de tres mesos de desenvolupament efectiu, amb una dedicació mitjana de diverses hores setmanals cadascun, el cost en hores de treball seria el més significatiu del projecte. No obstant això, en tractar-se d'un projecte educatiu, aquest cost no representa una despesa econòmica real.

En conclusió, el projecte Multi Games GUAU s'ha desenvolupat amb un cost econòmic directe pràcticament nul, aprofitant eines gratuïtes i recursos ja disponibles, cosa que demostra que és possible desenvolupar un producte de qualitat sense necessitat d'una inversió econòmica elevada.

2 Descripció del projecte

2.1 Anàlisi de requisits

Per garantir que Multi Games GUAU funcionés correctament i oferís una bona experiència als usuaris, es va definir un llistat de requisits abans de començar el desenvolupament. Aquests requisits van servir de guia durant tot el procés i van permetre verificar, en la fase de proves, que el producte final complia les expectatives establertes.

2.1.1 Requisits funcionals

- El sistema ha de permetre al jugador controlar els elements de cada joc mitjançant teclat i/o ratolí.
- El sistema ha de respondre immediatament a les entrades de l'usuari, sense latència perceptible.
- El sistema ha de detectar correctament les col·lisions entre els elements del joc.
- El sistema ha de identificar les condicions de final de partida i mostrar un missatge informatiu a l'usuari.
- El sistema ha de permetre reiniciar la partida o tornar al menú principal en qualsevol moment.
- El sistema ha d'actualitzar la puntuació en temps real durant la partida.
- El sistema ha de comparar la puntuació actual amb la puntuació màxima guardada i actualitzar-la si la nova és superior.
- El sistema ha de guardar la puntuació màxima de cada joc de manera persistent entre sessions.
- El sistema ha de recuperar les puntuacions màximes guardades en iniciar cada joc.
- El sistema ha de mostrar les puntuacions màximes de tots els jocs a la pantalla de puntuacions.
- En els jocs que ho permeten, el sistema ha d'augmentar progressivament la dificultat a mesura que el jugador avança.
- El sistema ha de permetre navegar entre el menú principal, el menú de selecció de jocs i cada joc de manera senzilla.
- El sistema ha d'adaptar la resolució de la finestra a les dimensions òptimes de cada joc en carregar-lo.

2.1.2 Requisits no funcionals

Estabilitat: el sistema no ha de presentar errors crítics durant l'ús normal ni tancar-se de manera inesperada.

Rendiment: el joc ha de funcionar de manera fluida en maquinari estàndard, sense caigudes de la taxa de fotogrames per segon que afectin la jugabilitat.

Temps de càrrega: el temps de càrrega entre pantalles ha de ser mínim per no interrompre el flux de joc.

Usabilitat: la interfície ha de ser clara i intuïtiva, amb elements interactius fàcilment identificables i informació llegible en tot moment.

Portabilitat: el projecte ha de poder executar-se en qualsevol equip compatible amb Godot Engine 4 sense configuracions addicionals.

Mantenibilitat: el codi ha d'estar ben organitzat, estructurat i comentat per facilitar la detecció d'errors i la incorporació de millores futures.

Accessibilitat: els controls han de ser senzills i aprensibles per a qualsevol tipus d'usuari, independentment del seu nivell d'experiència amb els videojocs.

2.1 Previsió de tasques d'investigació projecte d'investigació]

Abans de començar el desenvolupament es van identificar les tasques d'investigació necessàries per poder prendre decisions tècniques fonamentades:

- Estudiar el funcionament de les mecàniques dels jocs arcade clàssics (Snake, Tetris, Pong, Space Invaders, Busca Mines, Billar) per entendre com adaptar-les al projecte.

- Analitzar i comparar els motors de videojocs disponibles (Godot Engine 4, Unity, Unreal Engine, RPG Maker) per escollir el més adequat segons el tipus de projecte, el cost i la corba d'aprenentatge.

- Aprendre el funcionament bàsic de Godot Engine 4: sistema de nodes, escenes, signals i el cicle de vida dels nodes (_ready, _process, _physics_process).

- Investigar com implementar detecció de col·lisions en Godot Engine 4 per als diferents tipus de joc (col·lisions entre Area2D, RigidBody2D, CharacterBody2D i TileMap).

- Investigar com implementar físiques 2D realistes per al Billar i el Pong, on el comportament dels rebots és crític per a l'experiència de joc.

- Estudiar el sistema de TileMap de Godot Engine 4 per determinar si era adequat per implementar el tauler del Tetris i el Busca Mines amb múltiples capes.

- Investigar com guardar i recuperar dades de manera persistent entre sessions, comparant les opcions disponibles a Godot Engine 4 (ConfigFile, FileAccess, base de dades externa).

- Estudiar el sistema d'Autoload (singleton) de Godot Engine 4 per determinar si era la solució adequada per centralitzar el sistema de puntuacions.

- Investigar com gestionar la resolució de la finestra dinàmicament per adaptar-la a les dimensions òptimes de cada joc.

- Estudiar com organitzar un projecte de Godot Engine 4 amb múltiples jocs independents dins d'un mateix repositori, garantint que les escenes no interfereixin entre elles.

2.2 Tecnologies

2.2.1 Comparativa de les tecnologies valorades

Per escollir el motor de desenvolupament més adequat es van analitzar quatre opcions principals:

Godot Engine 4 és un motor gratuït i de codi obert especialment orientat al desenvolupament de jocs 2D. Utilitza un sistema basat en nodes i escenes que facilita l'organització dels elements del joc. El seu llenguatge GDScript té una sintaxi accessible similar a Python. Ofereix documentació extensa, una comunitat activa i un editor visual intuïtiu. No té limitacions de funcionalitats per versió ni requereix pagament per publicar. És l'opció amb la millor relació entre accessibilitat, funcionalitats i cost per a un projecte educatiu de 2D.

Unity és un motor molt potent i àmpliament utilitzat en la indústria. Suporta el desenvolupament en 2D i 3D i disposa d'una gran quantitat de recursos i tutorials disponibles. No obstant això, el seu llenguatge principal és C#, que té una corba d'aprenentatge més pronunciada que GDScript. A més, algunes funcionalitats avançades estan limitades a les versions de pagament, i els requisits de maquinari són superiors als de Godot.

Unreal Engine és un dels motors més avançats del mercat, dissenyat principalment per a projectes de gran escala amb gràfics 3D d'alta qualitat. Utilitza C++ com a llenguatge principal, amb una complexitat tècnica molt elevada. No és adequat per a un projecte educatiu de les dimensions de Multi Games GUAU.

RPG Maker és una eina específica per a jocs de rol en 2D. Tot i ser accessible, té limitacions tècniques importants: no permet implementar de manera nativa mecàniques com les físiques del Billar, el sistema de tiles del Busca Mines o els shooters, i requereix pagament per llicència.

2.2.2 Tecnologies escollides

Es van escollir les tecnologies següents per al desenvolupament de Multi Games GUAU:

Godot Engine 4 com a motor principal de desenvolupament. Es va triar perquè és completament gratuït, de codi obert, i està especialment orientat al desenvolupament de jocs 2D. GDScript va resultar fàcil d'aprendre i va permetre treballar de manera àgil, provar canvis ràpidament i entendre millor el funcionament intern de cada joc. El sistema de nodes i escenes va facilitar l'organització dels elements de cada joc de manera clara i estructurada.

GitHub Desktop per al control de versions del projecte. Es va triar per la seva interfície gràfica senzilla que no requereix coneixements de línia de comandes, i perquè permet treballar de manera coordinada entre els dos membres de l'equip mantenint un historial complet de tots els canvis realitzats.
Padlet per a la gestió de tasques i el seguiment del progrés. Es va triar per la seva accessibilitat, la seva interfície visual intuïtiva i perquè disposa d'una versió gratuïta suficient per a les necessitats del projecte.

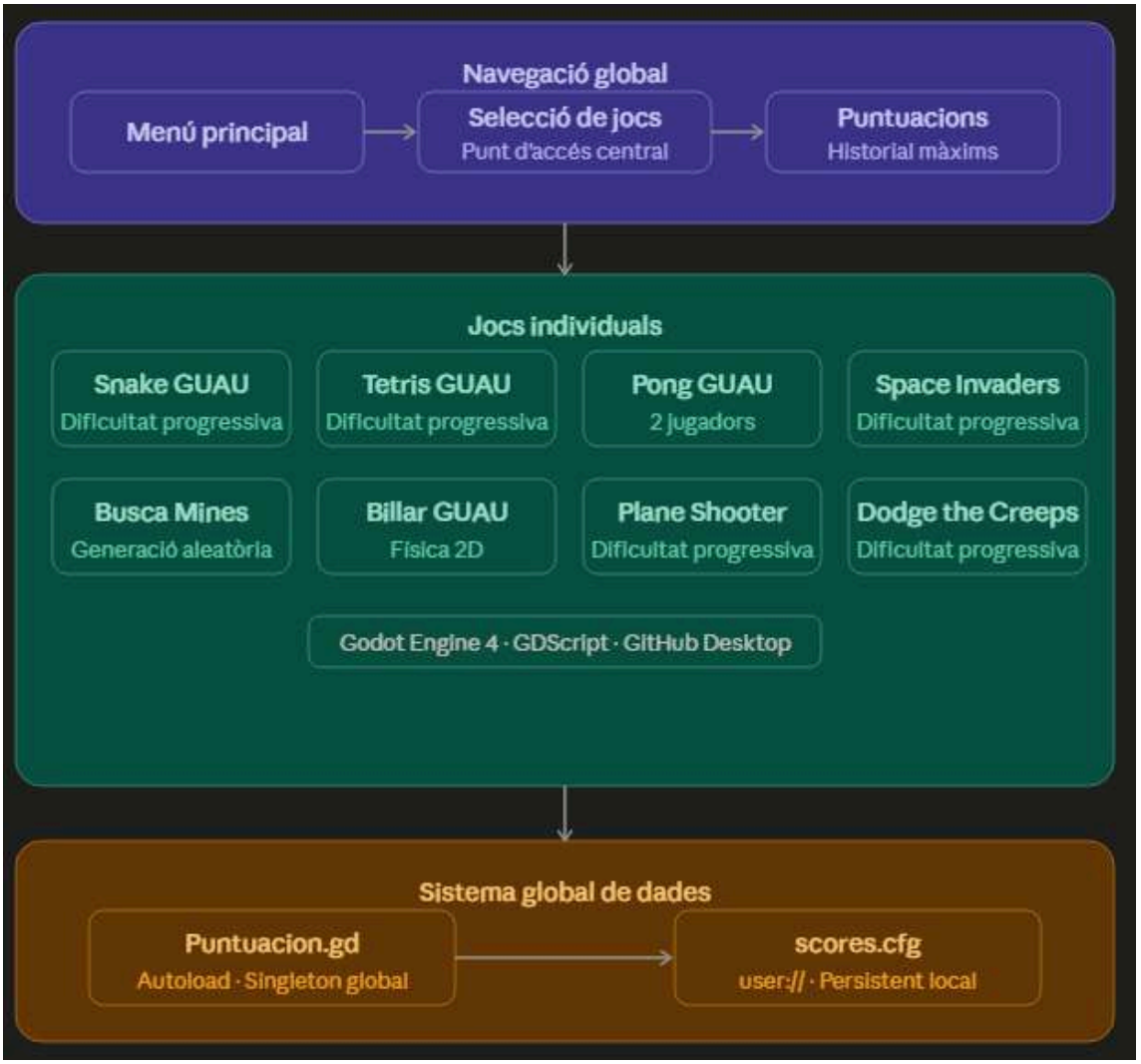
2.3 Estructura del projecte

El projecte s'organitza en un únic repositori de Godot Engine 4 que conté totes les escenes, scripts i recursos. L'estructura es basa en tres capes interconnectades:

La primera capa és la navegació global, formada pel menú principal i el menú de selecció de jocs. Des d'aquí l'usuari accedeix a qualsevol dels vuit jocs i a la pantalla de puntuacions.

La segona capa són els jocs individuals, cadascun amb les seves pròpies escenes, nodes i scripts independents. Tot i la seva independència, tots comparteixen la mateixa convenció d'estructura i es comuniquen amb la capa de dades mitjançant l'Autoload global.

La tercera capa és el sistema de dades global (Puntuacion.gd), implementat com a Autoload de Godot Engine 4. Actua com a singleton accessible des de qualsevol escena i s'encarrega de guardar i recuperar les puntuacions màximes de tots els jocs mitjançant un fitxer ConfigFile local.



2.4 Descripció dels components

2.4.1 Component 1: Interfície d'usuari

La interfície d'usuari és la part visual de Multi Games GUAU amb la qual el jugador interactua durant tota l'experiència de joc. S'organitza en dos nivells.

El primer nivell és la navegació global: el menú principal, que és la primera pantalla que veu l'usuari en obrir el joc, i el menú de selecció de jocs, des d'on pot escollir a quin joc vol jugar. Aquests menús s'han dissenyat perquè la navegació sigui ràpida i fàcil, amb elements visuals clars que identifiquen cada opció disponible.

El segon nivell és la interfície dins de cada joc, que inclou el marcador de puntuació actual, la puntuació màxima assolida, el nombre de vides restants en els jocs que ho implementen, i un botó o tecla per tornar al menú de selecció en qualsevol moment. Tots aquests elements se situen als marges de la pantalla per no obstaculitzar la zona de joc activa.

2.4.2 Component 2: Lògica del sistema

La lògica del sistema és el component central que fa funcionar cada joc. Està implementada en GDScript i s'organitza en scripts associats als nodes de cada escena.

Cada joc té els seus propis scripts independents que gestionen: el moviment dels elements, la detecció de col·lisions, la gestió de la puntuació, el comportament dels elements del joc i, en alguns casos, la progressió de la dificultat. Tot i la independència de cada joc, tots segueixen una estructura comuna que facilita la comprensió del codi i el manteniment del projecte.

2.4.3 Component 3: Sistema de dades

El sistema de dades s'encarrega de guardar i recuperar la informació persistent del projecte, principalment la puntuació màxima de cada joc. Està implementat com a Autoload global (Puntuacion.gd) accessible des de qualsevol escena.

Utilitza la classe ConfigFile de Godot Engine 4 per guardar les dades en un fitxer local (user://scores.cfg) al dispositiu de l'usuari. Quan un joc acaba, compara la puntuació obtinguda amb la màxima guardada i, si és superior, actualitza el fitxer. En iniciar qualsevol joc, el sistema llegeix el fitxer i carrega la puntuació màxima corresponent. Tot el procés és automàtic i transparent per a l'usuari.

2.5 Definició de les tasques [projecte d'investigació]

2.5.1 Prova 1: Funcionament dels jocs

Què es vol comprovar: que tots els jocs de Multi Games GUAU responen correctament als controls del jugador i que les mecàniques bàsiques de cada joc funcionen segons l'esperit del joc original.

Components utilitzats: els scripts principals de cada joc (main.gd o equivalent), el sistema d'Input de Godot Engine 4 i els nodes de cada escena.

Procés seguit: es va provar cada joc de manera independent, verificant que el moviment dels elements responia als controls definits, que els elements del joc es generaven correctament (la serp al Snake, les peces al Tetris, els enemics al Space Invaders) i que el comportament general era el previst. Es van realitzar diverses partides completes de cada joc per detectar comportaments inesperats.

Conclusions: durant aquesta prova es van detectar errors puntuals relacionats amb el moviment i la detecció de col·lisions en alguns jocs, que es van corregir abans de continuar. Un exemple concret va ser la gestió de la direcció contrària al Snake, on inicialment el jugador podia girar 180° i col·lisionar amb el seu

propi cos immediatament. Es va solucionar afegint la comprovació move_direction != up/down/left/right abans d'acceptar el canvi de direcció.

2.5.2 Prova 2: Sistema de puntuació i interacció

Què es vol comprovar: que el sistema de puntuació s'actualitza correctament en temps real, que les interaccions del jugador amb els elements del joc tenen l'efecte esperat sobre la puntuació, i que la dificultat progressiva funciona correctament en els jocs que la implementen.

Components utilitzats: els scripts de puntuació de cada joc, els nodes HUD i ScoreLabel de cada escena, i els paràmetres de dificultat progressiva (velocitat, freqüència d'aparició).

Procés seguit: es va verificar que el marcador s'actualitzava en temps real cada vegada que es produïa una acció que havia de modificar-lo (recollir menjar al Snake, eliminar una fila al Tetris, embocinat una bola al Billar). Es va comprovar que la dificultat augmentava progressivament jugant partides llargues a cada joc, verificant que la velocitat o la freqüència d'aparició d'elements canviava correctament. Es van repetir les proves diverses vegades per assegurar la consistència del comportament.

Conclusions: el sistema de puntuació va funcionar correctament en tots els jocs. Respecte a la dificultat progressiva, es va comprovar que al Space Invaders la reducció del temps entre aparicions era massa agressiva en les primeres versions, fent el joc impossible molt ràpidament. Es va ajustar el valor mínim a 0.2 segons i la reducció per tick a 0.01 segons per obtenir una corba de dificultat més equilibrada.

2.5.3 Prova 3: Guardat i recuperació de dades

Què es vol comprovar: que el sistema de guardat Puntuacion.gd guarda correctament la puntuació màxima en finalitzar cada partida, que les dades es mantenen entre sessions, i que la pantalla de puntuacions les mostra correctament.

Components utilitzats: l'Autoload Puntuacion.gd, el fitxer user://scores.cfg, i la pantalla de puntuacions (puntuaciones.tscn).

Procés seguit: es van realitzar diverses partides de cada joc, finalitzant-les amb puntuacions diferents per verificar que el sistema guardava correctament la puntuació màxima i no la sobreescrivia amb una puntuació inferior. Es va tancar i tornar a obrir el joc diverses vegades per comprovar que les dades es mantenien entre sessions. Es va verificar que la pantalla de puntuacions mostrava els valors correctes per a cada joc.

Conclusions: el sistema va funcionar correctament en tots els casos. Les puntuacions màximes es guardaven i recuperaven sense errors, i la pantalla de puntuacions les mostrava correctament. Es va comprovar també que el fitxer

scores.cfg es creava automàticament la primera vegada que es finalitzava una partida, sense necessitat de cap configuració prèvia per part de l'usuari.

2.5 Definició de les funcionalitats [projecte de desenvolupament]

2.5.1 Funcionalitat 1: Control dels jocs

Què permet fer: controlar els elements de cada joc mitjançant teclat i/o ratolí de manera fluida i precisa.

Com funciona: cada joc registra les entrades de l'usuari a cada frame mitjançant el sistema Input de Godot Engine 4. Alguns jocs registren les accions dinàmicament en temps d'execució via InputMap si no existeixen prèviament al projecte. Els controls varien segons el joc: Snake i Tetris utilitzen les tecles de fletxa i WASD; Pong utilitza W/S per a cada jugador; Space Invaders i Plane Shooter combinen WASD amb el ratolí per apuntar i disparar; Billar utilitza el ratolí per apuntar i carregar el tir; Busca Mines utilitza el clic esquerre per descobrir cel·les i el dret per col·locar banderes; Dodge the Creeps utilitza les fletxes i WASD.

Estat d'implementació: completament implementada en tots els jocs.

2.5.2 Funcionalitat 2: Sistema de joc i progressió

Què permet fer: jugar a cada joc amb les seves mecàniques pròpies i, en alguns casos, experimentar un augment progressiu de la dificultat a mesura que s'avança.

Com funciona: cada joc implementa les seves mecàniques de manera independent. La dificultat progressiva s'implementa de maneres diferents segons el joc: al Tetris, la velocitat de caiguda de les peces augmenta en 0.25 unitats per cada fila eliminada; al Space Invaders, el temps entre aparicions d'enemics es redueix en 0.01 segons per tick fins a un mínim de 0.2 segons; al Plane Shooter, els nivells successius incorporen més enemics i de tipus més difícils; al Snake, la serp s'allarga i l'espai disponible es redueix a mesura que el jugador menja.

Estat d'implementació: completament implementada. La dificultat progressiva està implementada als jocs que ho permeten per la seva naturalesa; els jocs com el Billar o el Pong no l'incorporen perquè el seu disseny no ho requereix.

2.5.3. Funcionalitat 3: Guardat de dades

Què permet fer: guardar automàticament la puntuació màxima de cada joc entre sessions, de manera que el jugador pugui consultar-la i intentar superar-la en futures partides.

Com funciona: en finalitzar cada partida, el joc crida Puntuacion.save_score(nom_joc, puntuació). L'Autoload compara la nova puntuació amb la guardada al fitxer user://scores.cfg i, si és superior, actualitza el valor. En iniciar qualsevol joc, el sistema llegeix el fitxer i carrega la puntuació màxima corresponent. La pantalla de puntuacions mostra les puntuacions màximes de tots els jocs cridant Puntuacion.get_score(nom_joc) per a cada un.

Estat d'implementació: completament implementada per a tots els jocs que disposen de sistema de puntuació (Snake, Tetris, Space Invaders, Busca Mines, Plane Shooter i Dodge the Creeps). El Pong i el Billar no guarden puntuació màxima per la seva naturalesa competitiva i basada en vides respectivament.

3. Altres capítols

3.1 Decisions tècniques durant el desenvolupament

Al llarg del projecte vam haver de prendre diverses decisions tècniques rellevants. En cada cas vam valorar les alternatives disponibles i vam escollir l'opció més adequada segons criteris tècnics i pràctics.

La primera gran decisió va ser el canvi de projecte. Inicialment volíem desenvolupar Pokemon Classic, un joc de rol en 2D amb mapa del món, batalles per torns i progressió de personatge. Després d'analitzar la complexitat tècnica real del projecte, vam comprovar que implementar un sistema de batalles, diàlegs, mapa transitable i estadístiques de personatge requeria un volum de treball molt superior al temps disponible. L'alternativa era reduir l'abast i fer un projecte més petit però complet. Vam optar per aquesta segona opció i vam crear Multi Games GUAU, cosa que ens va permetre lliurar un producte funcional i de qualitat dins del termini establert.

3.2 Decisió sobre la reutilització de codi extern

Una altra decisió important va ser si partir de codi font existent disponible en repositoris públics o desenvolupar tots els jocs des de zero. Reutilitzar codi hauria accelerat el procés, però ens hauria impedit entendre completament el funcionament intern de cada joc i hauria dificultat la integració dins d'un entorn comú. Vam decidir programar tots els jocs des de zero en GDScript, cosa que ens va permetre aprendre molt més i tenir control total sobre l'estructura del codi i les mecàniques de cada joc.

3.3 Decisió sobre l'estructura del projecte

Vam valorar dues maneres d'organitzar els jocs dins de Godot Engine 4: tenir un únic projecte amb tots els jocs integrats, o crear projectes separats i exportar-los individualment. L'opció de projecte únic facilitava compartir recursos comuns com el sistema de puntuacions (Puntuacion.gd com a Autoload global), el menú principal i la pantalla de puntuacions, i simplificava molt la navegació entre jocs. Vam escollir aquesta opció per la seva coherència tècnica i per la millor experiència d'usuari que oferia.

3.4 Decisió sobre el sistema de guardat de puntuacions

Per guardar les puntuacions màximes vam valorar tres opcions: guardar les dades en variables globals (que es perden en tancar el joc), usar una base de dades externa, o usar el sistema de fitxers propi de Godot amb ConfigFile. La primera opció era massa limitada. La segona era excessivament complexa per a les dimensions del projecte. Vam escollir ConfigFile perquè és la solució nativa de Godot, no requereix cap dependència externa, guarda les dades de manera persistent a user://scores.cfg i és molt senzilla d'implementar i mantenir. A més, en centralitzar-la en un Autoload global, tots els jocs poden accedir-hi amb una sola línia de codi.

3.5 Decisió sobre la resolució i la finestra de joc

Cada joc té unes dimensions òptimes diferents. Vam valorar si fer tots els jocs amb la mateixa resolució fixa o permetre que cada joc adaptés la mida de la finestra. Fixar una resolució única hauria simplificat el desenvolupament, però hauria perjudicat l'experiència visual d'alguns jocs. Vam optar per canviar dinàmicament la mida de la finestra en carregar cada joc des del menú de selecció, assignant a cada un la resolució més adequada per al seu disseny (per exemple, Snake a 1000×1050, Tetris a 850×750 o Billar a 1200×775).

4 Conclusions

4.1 Conclusions generals del projecte

El desenvolupament de Multi Games GUAU ha estat una experiència molt completa tant a nivell acadèmic com personal. El projecte ens ha permès aplicar de manera pràctica un conjunt ampli de coneixements adquirits durant el curs, que van molt més enllà de la programació pura: gestió de projectes, presa de decisions tècniques sota pressió, organització del treball en equip i adaptació als canvis imprevistos.

Acadèmicament, hem après a utilitzar un motor de videojocs professional com Godot Engine 4, a programar en GDScript, a estructurar un projecte amb múltiples escenes i scripts interconnectats, i a implementar sistemes reutilitzables com l'Autoload de puntuacions. També hem après a fer servir eines de control de versions amb GitHub Desktop, cosa que ens ha donat una primera aproximació real al flux de treball professional en el desenvolupament de programari.

A nivell personal, el canvi de Pokemon Classic a Multi Games GUAU va ser el moment més exigent del projecte. Reconèixer que un plantejament inicial no és viable i saber reorientar el treball de manera ràpida i organitzada és una competència fonamental en qualsevol àmbit professional del sector informàtic, i aquest projecte ens ha obligat a exercitar-la en un context real.

El resultat final és un producte complet, funcional i accessible que demostra que és possible desenvolupar un projecte de qualitat amb recursos limitats, eines gratuïtes i una bona organització del treball.

4.2 Consecució dels objectius

Dissenyar i implementar els jocs del projecte amb un funcionament correcte i sense errors greus. Objectiu assolit. Els vuit jocs (Snake, Tetris, Pong, Space Invaders, Busca Mines, Billar, Plane Shooter i Dodge the Creeps) funcionen correctament i han superat les proves de funcionament sense errors crítics.

Programar sistemes de joc com el moviment, la detecció de col·lisions, la puntuació i altres mecàniques bàsiques. Objectiu assolit. Cada joc té implementades les seves mecàniques pròpies: detecció de col·lisions, moviment de personatges i elements, i sistemes de puntuació individuals.

Afegir sistemes de dificultat progressiva en els jocs que ho permetin. Objectiu assolit parcialment. S'ha implementat dificultat progressiva al Tetris (augment de velocitat per cada fila eliminada), al Space Invaders (reducció del temps entre aparicions d'enemics) i al Plane Shooter (nivells amb més enemics). Alguns jocs com el Billar o el Pong no incorporen dificultat progressiva per la seva naturalesa.

Desenvolupar un menú principal i un menú de selecció de jocs amb una interfície clara i intuïtiva. Objectiu assolit. Els dos menús estan implementats amb una navegació clara, imatges representatives de cada joc i accés directe a qualsevol joc amb un sol clic.

Implementar un sistema de guardat de la puntuació màxima persistent entre sessions. Objectiu assolit. El sistema Autoload Puntuacion.gd guarda i recupera correctament les puntuacions màximes de tots els jocs mitjançant un fitxer ConfigFile local.

Garantir l'estabilitat del projecte i evitar problemes tècnics greus. Objectiu assolit. Durant les fases de proves no es van detectar errors crítics que interrompessin les partides. Els errors puntuals detectats es van corregir abans de la fase d'integració final.

Organitzar el projecte de manera eficient, repartint les tasques i fent ús d'eines de planificació. Objectiu assolit. Es van utilitzar Padlet per a la gestió de tasques, el diagrama de Gantt per a la planificació temporal i GitHub Desktop per al control de versions, mantenint el projecte organitzat durant tot el desenvolupament.

4.3 Valoració de la metodologia i planificació

La metodologia àgil escollida va resultar ser l'encert més important del projecte. Gràcies a la seva flexibilitat, vam poder gestionar el canvi de Pokemon Classic a Multi Games GUAU sense perdre l'organització ni el control del progrés. Una metodologia tradicional, amb tota la planificació tancada des del principi, hauria fet molt més difícil absorir un canvi tan profund a mitja durada del projecte.

La planificació inicial es va veure afectada significativament per aquest canvi, que va obligar a replanejar completament les tasques i els temps en un termini molt curt. Tot i això, les eines que fèiem servir van facilitar molt l'adaptació: el Padlet ens va permetre redistribuir les tasques ràpidament, el diagrama de Gantt ens va ajudar a recalcular els temps restants, i GitHub Desktop ens va garantir que no perdíem cap feina feta durant la transició.

Com a aspecte crític, cal reconèixer que la planificació inicial va ser excessivament optimista respecte a la complexitat de Pokemon Classic. Una anàlisi tècnica més detallada a l'inici del projecte hauria permès detectar abans la inviabilitat del plantejament original i hauria donat més marge de temps per al desenvolupament de Multi Games GUAU. Aquesta és una lliçó important: en qualsevol projecte de programari, la fase d'anàlisi de viabilitat tècnica és tan important com el desenvolupament en si.

4.4 Visió de futur

Multi Games GUAU és un projecte que té un recorregut clar de millora i ampliació en diverses direccions.

A curt termini, la millora més immediata seria afegir nous jocs a la col·lecció. Durant el desenvolupament es van valorar altres títols clàssics com Arkanoid, Pac-Man o Flappy Bird que no es van poder implementar per limitacions de temps. Incorporar-los ampliaria significativament el catàleg i enriquia l'experiència general.

També seria possible millorar la qualitat visual i sonora dels jocs existents. Alguns jocs com el Space Invaders o el Dodge the Creeps utilitzen gràfics molt simples o procedurals. Incorporar sprites i efectes de so de major qualitat milloraria considerablement la presentació del producte.

A mig termini, una millora molt motivadora seria implementar un sistema de puntuacions en línia, que permetés als jugadors comparar les seves puntuacions màximes amb les d'altres usuaris. Godot Engine 4 ofereix suport per a peticions HTTP que farien possible connectar el joc a un servidor extern o a una base de dades en núvol.

A llarg termini, l'adaptació del projecte per a dispositius mòbils seria la millora amb més impacte en termes d'abast d'usuaris. Godot Engine 4 permet exportar projectes per a Android i iOS, però caldria redissenyar els controls per a pantalles tàctils i adaptar les interfícies a resolucions de mòbil. Aquesta adaptació obriria Multi Games GUAU a un públic molt més ampli i convertiria el projecte en un producte distribuïble de manera gratuïta a través de les botigues d'aplicacions.

5. Glossari

Arcade: gènere de videojocs caracteritzat per mecàniques simples, partides curtes i dificultat progressiva. Els jocs arcade clàssics com Snake, Tetris o Space Invaders van ser molt populars durant les dècades dels 80 i 90 i continuen sent una referència en el món dels videojocs.

Autoload: funcionalitat de Godot Engine 4 que permet carregar un script o escena automàticament en iniciar el joc i mantenir-lo accessible des de qualsevol altra escena durant tota l'execució. S'utilitza per a sistemes globals com el gestor de puntuacions (Puntuacion.gd).

Col·lisió: detecció del contacte físic entre dos elements del joc. Pot tenir efectes diferents segons el context: acabar la partida, sumar punts, rebottar un element o activar una acció concreta.

CollisionShape2D: node de Godot Engine 4 que defineix la forma geomètrica d'un objecte per a la detecció de col·lisions. Pot tenir forma de rectangle, cercle, càpsula o polígon personalitzat.

ConfigFile: classe de Godot Engine 4 que permet llegir i escriure fitxers de configuració en format de seccions i claus. Al projecte s'utilitza per guardar les puntuacions màximes de cada joc de manera persistent al fitxer user://scores.cfg.

Control de versions: sistema que registra tots els canvis realitzats en el codi d'un projecte al llarg del temps, permetent consultar l'historial de modificacions i recuperar versions anteriors en cas d'error. Al projecte s'ha utilitzat GitHub Desktop com a eina de control de versions.

Dificultat progressiva: sistema de joc que incrementa el nivell de dificultat a mesura que el jugador avança. Pot manifestar-se com un augment de la velocitat dels elements, una major freqüència d'aparició d'obstacles o qualsevol altre paràmetre que faci el joc més exigent amb el temps.

Escena: unitat bàsica d'organització a Godot Engine 4. Cada escena és un arxiu independent que conté una jerarquia de nodes i representa una part del joc, com ara un menú, un nivell o un personatge. Al projecte, cada joc té les seves pròpies escenes independents.

GDScript: llenguatge de programació propi de Godot Engine 4. Té una sintaxi clara i accessible, similar a Python, dissenyat específicament per al desenvolupament de videojocs. S'ha utilitzat per programar tota la lògica de Multi Games GUAU.

GitHub Desktop: aplicació d'escriptori que proporciona una interfície gràfica per gestionar repositoris Git. Permet fer commits, consultar l'historial de canvis i sincronitzar el codi entre membres de l'equip sense necessitat d'usar la línia de comandes.

Godot Engine 4: motor de videojocs gratuït i de codi obert utilitzat per al desenvolupament de Multi Games GUAU. Ofereix un editor visual integrat, un sistema de nodes i escenes, i suport per a exportació a múltiples plataformes.

HUD: acrònim de Heads-Up Display. És la capa d'informació visual que es mostra per sobre de la zona de joc durant la partida, incloent elements com el marcador de puntuació, les vides restants o la barra de vida del jugador.

Impuls: força aplicada de manera instantània a un objecte físic dins del motor de joc. Al Billar s'utilitza apply_central_impulse() per donar moviment a la bola blanca quan el jugador realitza un tir.

InputMap: sistema de Godot Engine 4 que permet definir i gestionar les accions d'entrada de l'usuari (teclat, ratolí, controlador). Al projecte s'ha utilitzat per registrar dinàmicament els controls de cada joc en temps d'execució.

Lògica de joc: conjunt de regles i instruccions programades que determinen com funciona un joc: el moviment dels elements, les interaccions entre objectes, el sistema de puntuació i el comportament general de cada mecànica.

Metodologia àgil: enfocament de gestió de projectes basat en cicles curts de treball iteratius. Permet adaptar-se als canvis i corregir errors de manera primerenca sense que afectin la totalitat del projecte.

Node: unitat bàsica d'organització dins de Godot Engine 4. Cada element del joc (personatge, obstacle, botó, temporitzador) es representa com un node amb propietats i comportaments propis. Els nodes s'organitzen en jerarquies per formar les escenes.

Padlet: eina de gestió visual de tasques en format de tauler digital. S'ha utilitzat per organitzar les tasques de cada membre de l'equip, compartir decisions i fer un seguiment del progrés del projecte.

PackedScene: tipus de dada de Godot Engine 4 que representa una escena empaquetada i llesta per ser instanciada dinàmicament durant l'execució. S'utilitza per generar elements repetitius com bales, enemics o segments de la serp.

Puntuació màxima: la millor puntuació assolida per un jugador en un joc concret. Es guarda de manera persistent mitjançant el sistema de fitxers i es mostra a la interfície perquè el jugador pugui intentar superar-la en futures sessions.

RigidBody2D: node de Godot Engine 4 que simula física realista, incloent gravetat, fricció i rebots. Al Pong s'utilitza per a les pales, i al Billar per a les boles.

Senyal: mecanisme de comunicació entre nodes a Godot Engine 4, equivalent al patró observer de programació. Permet que un node notifiqui altres nodes

quan es produeix un esdeveniment concret, com ara el final d'una partida o la recollida d'un objecte.

Singleton: patró de disseny de programari que garanteix que una classe o script té una única instància accessible globalment. A Godot Engine 4 s'implementa mitjançant l'Autoload. Al projecte, Puntuacion.gd és un singleton que gestiona les puntuacions de tots els jocs.

TileMap: node de Godot Engine 4 que permet construir escenaris a partir de tessel·les (tiles) organitzades en una graella. S'ha utilitzat al Tetris per gestionar les peces i el tauler, i al Busca Mines per gestionar les capes de mines, números, gespa i banderes.

Timer: node de Godot Engine 4 que emet un senyal quan transcorre un temps determinat. S'utilitza per controlar la velocitat de moviment de la serp al Snake, la caiguda de les peces al Tetris, l'aparició d'enemics al Space Invaders i al Dodge the Creeps, i molts altres aspectes temporals dels jocs.

6. Bibliografia

1. GDQuest. Learn GDScript From Zero. [En línia]. Disponible a: https://gdquest.github.io/learn-gdscript. [Consultat: maig 2025].

2. Godot Engine. Godot Engine Documentation — Version 4.x. [En línia]. Disponible a: https://docs.godotengine.org/en/stable. [Consultat: febrer – maig 2025].

3. GitHub. GitHub Desktop Documentation. [En línia]. Disponible a: https://docs.github.com/en/desktop. [Consultat: febrer 2025].

4. Wikimedia Foundation. Desenvolupament de videojocs. Viquipèdia. [En línia]. Disponible a: https://ca.wikipedia.org/wiki/Desenvolupament_de_videojocs. [Consultat: febrer 2025].

5. Godot Engine. Your first 2D game — Official tutorial. [En línia]. Disponible a: https://docs.godotengine.org/en/stable/getting_started/first_2d_game/index.html. [Consultat: març 2025].

6. Godot Engine. Using TileMap — Official documentation. [En línia]. Disponible a: https://docs.godotengine.org/en/stable/tutorials/2d/using_tilemaps.html. [Consultat: març 2025].

7. Godot Engine. Singletons (Autoload) — Official documentation. [En línia]. Disponible a: https://docs.godotengine.org/en/stable/tutorials/scripting/singletons_autoload.html. [Consultat: abril 2025].

8. Godot Engine. Saving games — Official documentation. [En línia]. Disponible a: https://docs.godotengine.org/en/stable/tutorials/io/saving_games.html. [Consultat: abril 2025].

9. Godot Engine. Physics introduction — Official documentation. [En línia]. Disponible a: https://docs.godotengine.org/en/stable/tutorials/physics/physics_introduction.html. [Consultat: març 2025].

10. Padlet. Padlet — Aplicació de gestió visual de tasques. [En línia]. Disponible a: https://padlet.com. [Consultat: febrer – maig 2025].

7 Annexos

1. **Compromís de treball en grup**
   Document signat per el membre del projecte (Aitor Carrillo Mata) on s'estableixen els compromisos de treball, la distribució de responsabilitats i les normes de funcionament del grup durant el desenvolupament del projecte.
   Compromís de treball en grup

2. **Proposta del projecte**
   Document inicial de proposta del projecte presentat al professor, on s'explica la idea original de Pokemon GUAU, la descripció del projecte, els objectius inicials i les tecnologies previstes (Godot, MySQL). Inclou també la justificació del canvi cap a Multi Games GUAU.
   Proposta del projecte

3. **Ficha proyecto SMX**
   Document oficial del centre on es recull la informació del projecte: nom, participants, tutor, objectius, assignatures relacionades i eines utilitzades.
   Ficha proyecto SMX

4. **Document funcional**
   Document que recull les decisions funcionals preses durant el desenvolupament del projecte, incloent la definició de les mecàniques de cada joc, l'estructura de les escenes i les decisions tècniques més rellevants.
   Document funcional

5. **Diagrama de Gantt**
   Planificació temporal completa del projecte en format de diagrama de Gantt, on es mostren totes les fases del desenvolupament (investigació, disseny, implementació, proves i integració), les tasques de cada fase, la seva durada i la distribució entre els dos membres de l'equip.
   Diagrama de Gantt

6. **Seguiment setmanal**
   Taula de seguiment setmanal del projecte on es registra el progrés de cada setmana, les tasques completades, les incidències detectades i els ajustos realitzats sobre la planificació inicial.
   Seguiment setmanal