Liridon
Dumea Emilian
Grupa: 1211B

Povestea jocului

Într-o lume vastă și plina de mister, trăiește un personaj curajos și aventuros, numit Aiden. Aiden locuiește într-un mic sat pitoresc situat în mijlocul unui unui peisaj încântător, dar înconjurat de pericole. Într-o zi, Aiden găsește un vechi pergament în podul casei sale. Acesta dezvăluie existența unui artefact misterios, cunoscut sub numele de Liridon, o sferă luminoasă care se spune că deține puteri neînțelese și care poate aduce echilibru și lumină în lume

Hotărât să descopere acest artefact legendar, Aiden pornește într-o călătorie epică. Îl așteaptă o cale lungă și plină de obstacole, unde fiecare pas înainte este o provocare.

În călătoria sa, Aiden se confruntă cu păduri misterioase, deșerturi aride și încărcate de căldură sufocantă, tărâmuri mistice, aducându-i provocări unice și pericole neașteptate. De-a lungul acestei călătorii, fel de fel de creaturi ciudate și monștrii sălbatici îl pândesc la fiecare colț, gata să îl oprească. Cu toate acestea, el își folosește curajul și înțelepciunea pentru a înfrunta fiecare obstacol și a progresa spre scopul sau.

În cele din urmă, Aiden ajunge în Pădurea Pierdută, un loc fermecat și plin de mistere. Aici, el se confruntă cu provocări de neimaginat și trebuie să își pună la încercare toate abilitățile pentru a depăși obstacolele. Dar, când ajunge în inima pădurii, găsește Liridon-ul strălucind în lumina difuză, emanând o energie blândă și misterioasă. Cu Liridon-ul în mână, Aiden simte o putere nouă palpitând în jurul său. Înconjurat de lumină, se întoarce în satul său, aducând cu el pace și prosperitate.

Prezentare joc

Joc single player, în care personajul principal trebuie să învingă inamicii și să ajungă la o ieșire pentru a trece la nivelul următor, mai puțin în cazul ultimului nivel, unde trebuie să ia artefactul Liridon după ce a înfrânt toți inamicii din calea sa Fiecare tărâm pe care ajunge Aiden reprezintă un alt nivel în care acesta trebuie să înfrunte noi dușmani până când vă găsi Liridon-ul și va readuce lumina și prosperitatea în lume.

Reguli joc

Jocul implică diverse elemente și locații pe traseul lui Aiden care diferă în funcție de nivel. Scopul jocului este de a parcurge toate nivelele și de a găsi Liridon-ul. Aiden este obligat să învingă inamicii care îi ies în cale pentru a putea înainta către nivelul următor. Pe măsură ce avansează în joc, Aiden se va confrunta cu inamici diferiți, care vor fi mai greu de eliminat și care vor avea atacuri diverse. Aiden va pierde un punct din viață de fiecare dată când va fi nimerit de un atac al inamicilor și de asemenea când va intra în coliziune cu unul dintre aceștia. Aventurierul are posibilitatea de a lansa diverse atacuri cu sabia, iar după primul nivel va avea posibilitatea de a trage cu arcul. Acesta va porni jocul cu 5 vieți, însă pe parcursul nivelelor, în anumite zone, se vor afla pe hartă vieți pe care le va putea colecta. Dacă va rămâne fără vieți, acesta va pierde jocul și va trebui să reia nivelul de la început.

Personajele jocului

- Aiden este personajul principal, un aventurier curajos, dornic de cunoaștere
Acesta este controlat de jucător. Scopul lui este de trece prin toate tărâmurile până când vă găsi Liridon-ul.

-Inamicii vor avea fiecare o înfățișare specifica pentru nivelul din care fac parte iar scopul lor este de a-l împiedica pe Aiden de a ajunge la Liridon

Tabla de joc

- Componente pasive:
    > Solul acoperit de iarbă - tile principal pe care se vor deplasa caracterele
    > Pietre, tufișuri, copaci, cactuși - elemente decorative ce oferă un aspect plăcut jocului

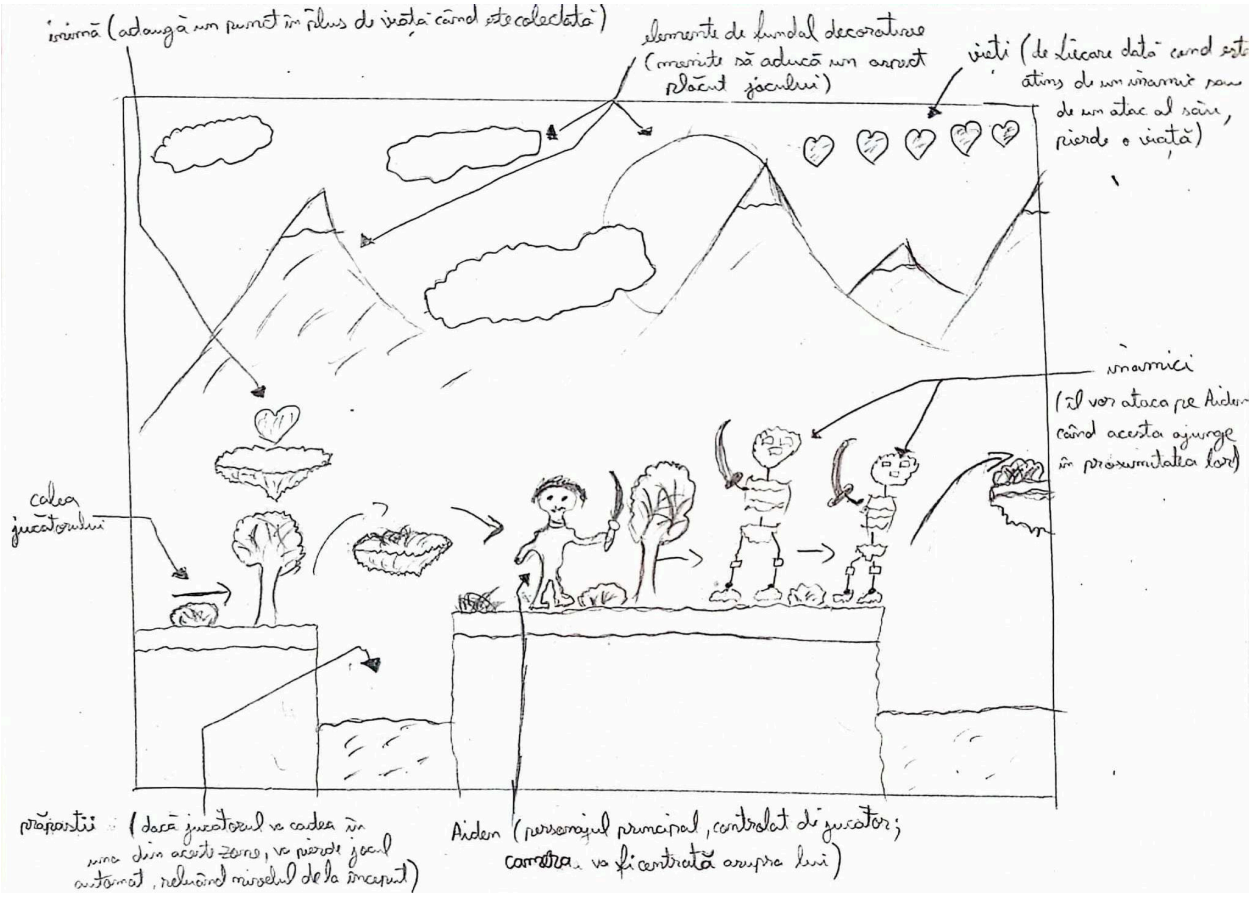
- Componente active:
    > vieți (Tile-uri cu aspect de inimă ce pot fi colectate pentru a câștiga un punct de viață în plus)
    > inamici (NPC-uri care vor ataca personajul principal)
    > monede de aur (vor mări scorul cu 50 de puncte cand sunt colectate)
    > monede de argint (vor mări scorul cu 30 de puncte cand sunt colectate)

- Structura tablei de joc: Tabla de joc este în principal caracterizată de o suprafață plana și uniforma, oferind jucătorului o bază stabilă pentru acțiune. Cu toate acestea, terenul vă prezenta și variații alte altitudinii, observandu-se sub forma unor pante mai ușoare sau mai abrupte, distribuite în mod sporadic pe suprafața tablei de joc, oferind o notă de diversitate și realism peisajului.

De asemenea, pentru a crește nivelul de dificultate, inamicii pot fi plasați uneori la o altitudine mai mare fata de cea a jucătorului pentru a-l pune la încercare, fiind nevoit să se ferească de atacurile lansate de la distanță ale inamicului pe măsură ce avansează către acesta. Pe lângă asta, viețile suplimentare vor fi plasate strategic în locuri mai dificil de ajuns, obligând jucătorul să sară pe mai multe platforme, cu riscul de a cădea în anumite prăpastii din tabla de joc, fapt ce va duce la pierderea instantă a progresului pe acel nivel.

Tabla de joc este construită din mai multe mape, fiecare dintre acestea reprezentând un nivel.

În colțul din dreapta sus va fi afișat numărul de vieți rămase.
Tabla de joc va fi centrată asupra lui Aiden, personajul controlat de jucător.



Mecanica jocului

> Nivele și traseu: Jocul este structurat pe nivele, iar Aiden trebuie să parcurgă fiecare nivel pentru a progresa și a găsi pe Liridon-ul. Fiecare nivel prezintă un traseu diferit, cu diverse elemente și locații.

> Inamici și atacuri: În timp ce parcurge traseul, Aiden se va confrunta cu diferiți inamici. Acești inamici au atacuri diverse și vor deveni progresiv mai puternici pe măsură ce jocul avansează. Aiden pierde un punct din viață atunci când este lovit de un atac sau intră în coliziune cu un inamic.

> Arme: Aiden poate folosi doar sabia pentru atacuri. De asemenea, el are si un atac special. Acest lucru oferă jucătorului opțiuni strategice suplimentare în luptă.

> Puncte de viață colectabile: Aiden începe jocul cu un număr de 100 de puncte de viață, dar poate găsi și colecta puncte de viață suplimentare pe parcursul nivelelor. Aceste puncte îi vor crește la loc viața și îi vor oferi o șansă suplimentară în cazul în

care nu mai are multe puncte.

> Keybinds:

- Tastele A si D - moving keys (la apăsarea acestor taste personajul se va deplasa în stânga, respectiv dreapta);
- Tasta W- jump key (la apăsarea acestei taste personajul va sări);
- Tasta R - attack key (la apăsarea acestei taste personajul va lansa un atac cu sabia);
- Tasta Q - power attack key (la apăsarea acestei taste personajul va lansa un atac special cu sabia, ce îl va costa o anumită cantitate de energie)
- Tasta O - salvează datele nivelului curent în baza de date

Game sprites

Descriere fiecare nivel

Jocul va avea cel puțin 3 nivele prin care jucătorul va trebui să treacă pentru a găsi Liridon-ul.

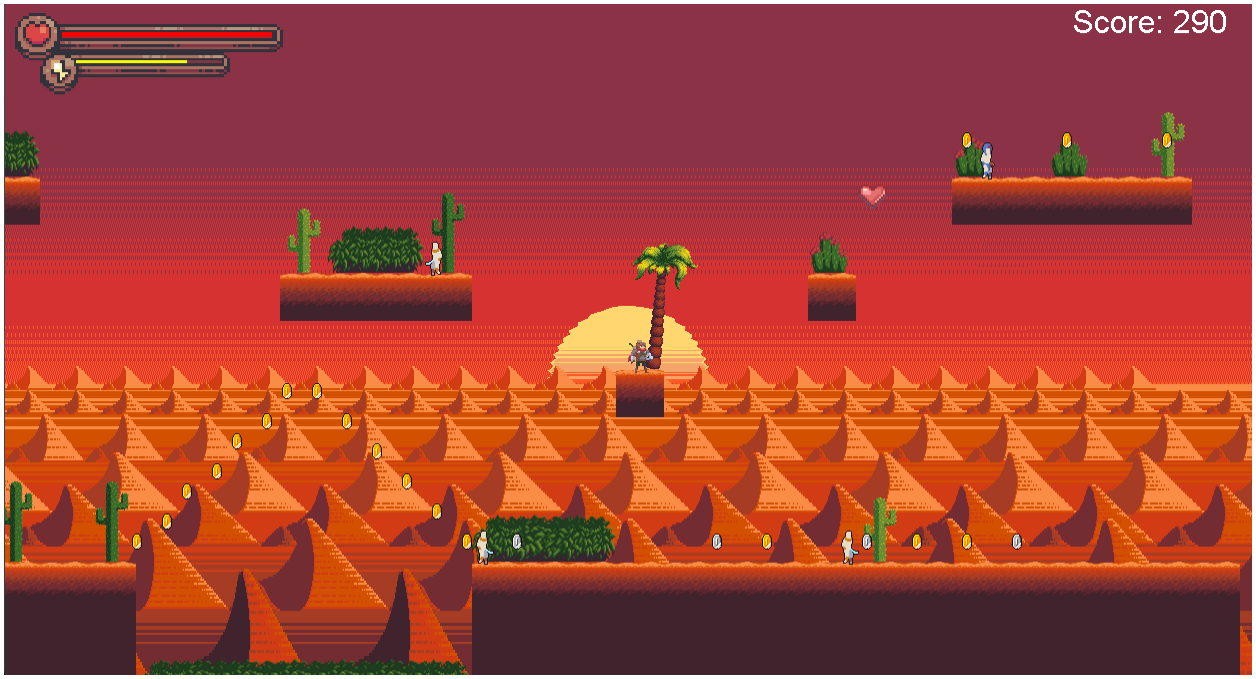
Primul nivel prezintă un grad de dificultate mai scăzut, astfel încât jucătorul să se acomodeze cu controalele și mecanica jocului. Acesta se va confrunta cu inamici pe care îi va dobora cu ușurință. Totuși, pe măsură ce avansează în acest nivel, numărul inamicilor va crește, pentru a mări într-o oarecare măsură dificultatea

Acest nivel se va desfășura într-o zonă magică, pe un platou înalt, străjuit de stânci înalte și sculptate în forme ciudate. Iarba moale și strălucitoare crește din crăpăturile stâncilor, vibrând în nuanțe de verde sub lumina soarelui. Pe acest platou magic se înalță și câțiva copaci înalți și subțiri, cu frunze strălucitoare.

Al doilea nivel are un grad de dificultate mai ridicat, determinat de poziționarea inamicilor pe hartă, dar și de numărul acestora. Pentru a ajunge la unii inamici, jucătorul va trebui să urce pe mai multe platforme până la aceștia, fapt ce aduce un dezavantaj semnificativ. Inamicii vor fi mai puternici și mai rapizi, și de asemenea în număr mai mare.

În acest nivel, Aiden se găsește prins în mijlocul unui vast deșert de nisipuri aurii

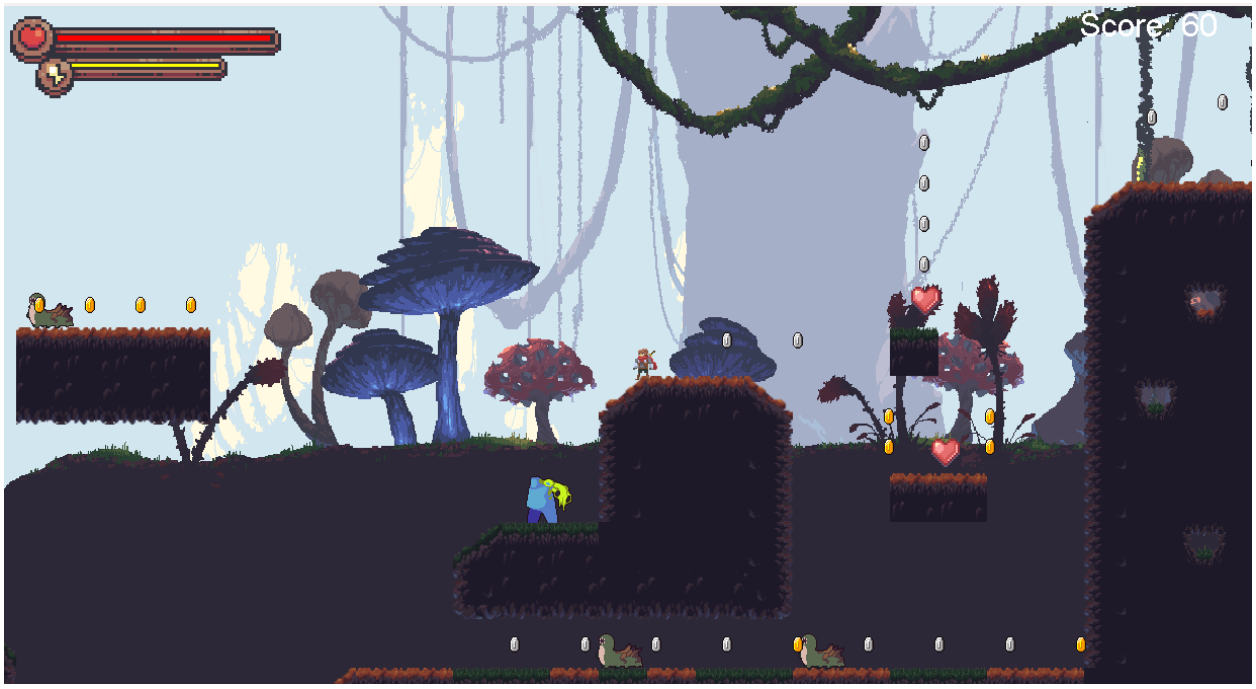
și dune nesfârșite. Pe lângă valurile nesfârșite de nisip, vor exista și mici lacuri cu apă cristalină, înconjurate de cactuși înalți și viguroși



Ultimul nivel aduce noi provocări. Jucătorul este întemeiat de noi inamici, care

sunt mai rezistenți și mai puternici, incluzând și mecanismul de poziționare, crescând astfel și mai mult nivelul de dificultate.

În acest nivel, Aiden se află în mijlocul unei păduri misterioase și fermecate, unde se găsesc ciuperci gigantice, copaci ce se înalță spre cer și o mlaștină întunecată.



Descriere meniu

- Play - începe un joc nou
- Load - încarcă un nivel salvat anterior
- Quit - ieșire joc

Documentație tehnică

Jocul conține mai multe șabloane de proiectare printre care:

Singleton - Implementarea acestuia în clasa Game asigură că jocul are o singură instanță pe durata execuției, oferind un punct central de acces și gestionare. Acest model de proiectare este esențial pentru a preveni inconsistențele și problemele de sincronizare care pot apărea atunci când mai multe instanțe ale unei clase critice sunt create și utilizate simultan.

Abstract Factory - oferă o interfață pentru crearea familiilor de obiecte înrudite sau dependente fără a specifica clasele lor concrete. Acest șablon este utilizat pentru a crea diferite tipuri de inamici pentru diferite niveluri. Acesta permite ca procesul de creare a inamicilor să fie consistent și ușor de extins atunci când sunt adăugate noi niveluri sau tipuri de inamici.

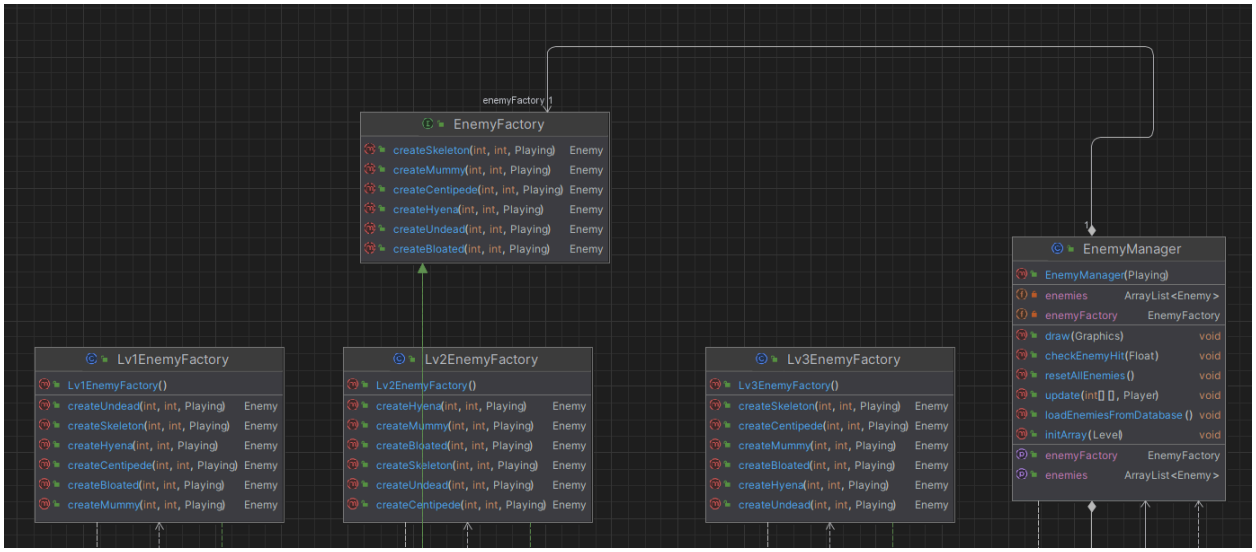
Componente cheie:

> Interfața și clasele concrete pentru inamici: Definirea comportamentelor și atributelor comune pentru toți inamicii.

> Interfața EnemyFactory: Declară metodele de creare pentru diferitele tipuri de inamici

> Clasele Concrete Factory (Lv1EnemyFactory, Lv2EnemyFactory, etc.): Implementează interfața EnemyFactory și furnizează logica specifică de instanțiere pentru fiecare tip de inamic pentru fiecare nivel.

> Clasa EnemyManger: Gestionează crearea, actualizarea și randarea inamicilor în joc, utilizând fabrica corespunzătoare în funcție de nivelul curent.



State - Acesta este utilizat pentru a gestiona diferitele stări ale jocului, cum ar fi PLAYING, MENU, QUIT, și LOAD. Acest șablon oferă o modalitate organizată și flexibilă de a separa comportamentele și logica specifică fiecărei stări, permițând astfel o gestionare eficientă a tranzițiilor dintre ele.

- Menu: Aceasta este starea în care jucătorul navighează prin opțiunile jocului, cum ar fi începerea unui nou joc, încărcarea unui joc salvat, accesarea setărilor sau ieșirea din joc.
- Playing: aceasta reprezintă starea principală a jocului, când jocul este efectiv în desfășurare. Gestionează toate aspectele jocului în timpul rulării, inclusiv actualizarea și desenarea scenei de joc, interacțiunea cu jucătorul, inamicii, etc.
- Load: Aceasta este starea în care jocul încarcă datele necesare pentru a începe sau continua o sesiune de joc. Implică inițializarea resurselor pentru un nou joc.
- Quit: Aceasta este starea în care jocul finalizează procesele și închide aplicația.

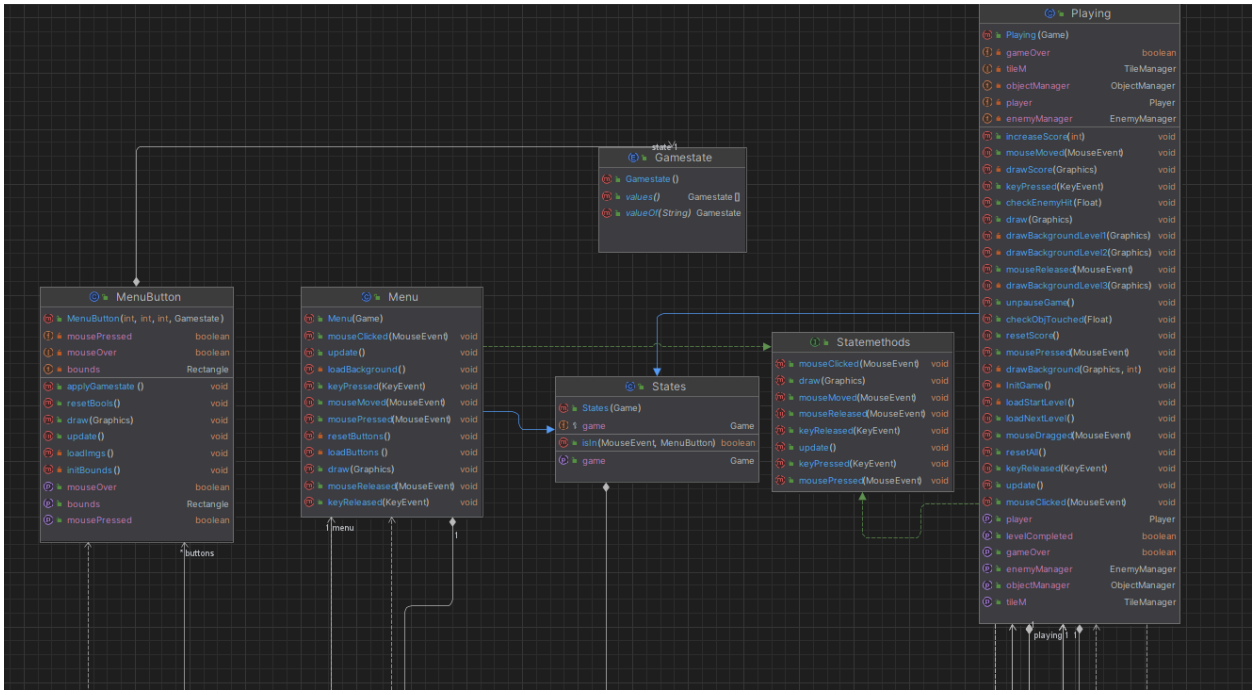
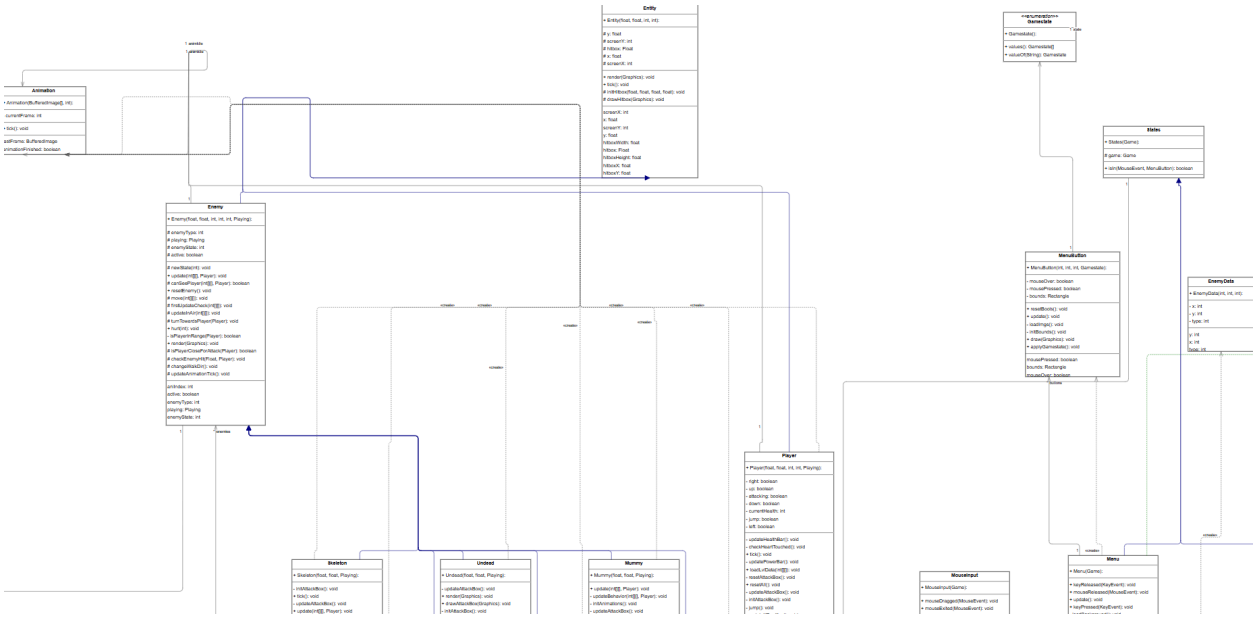
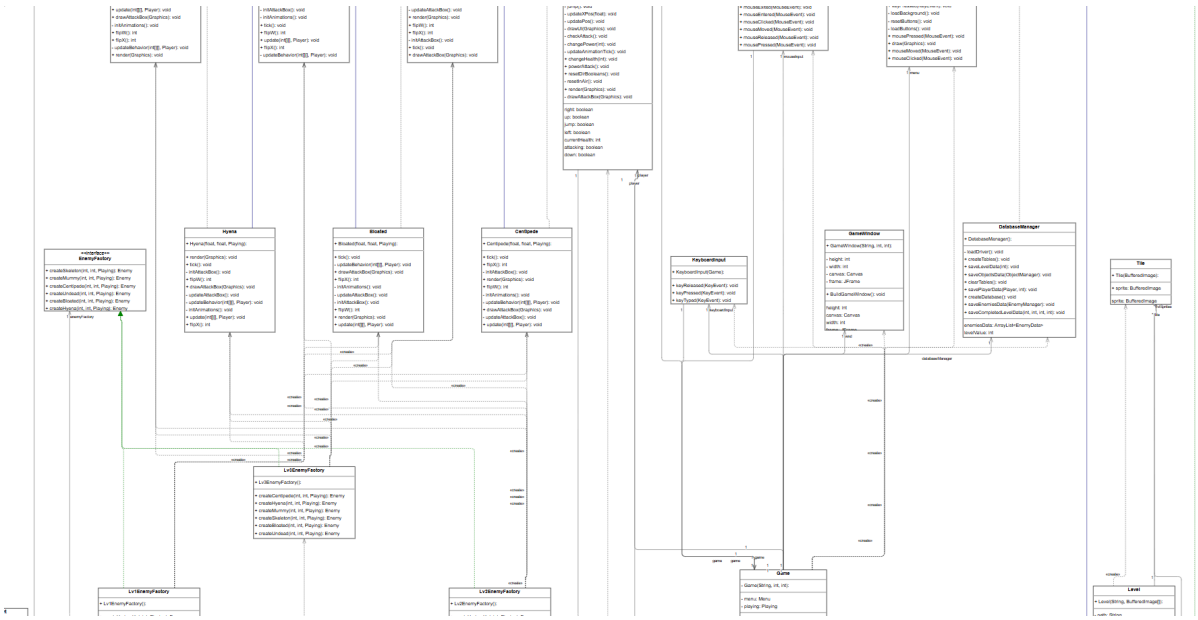
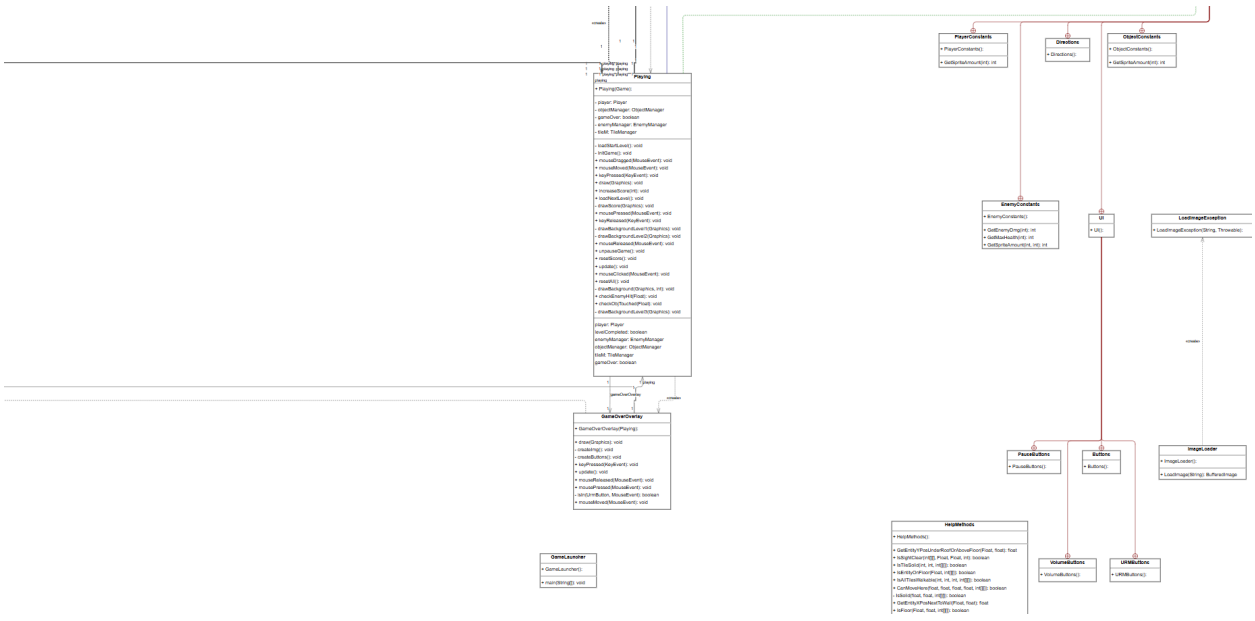
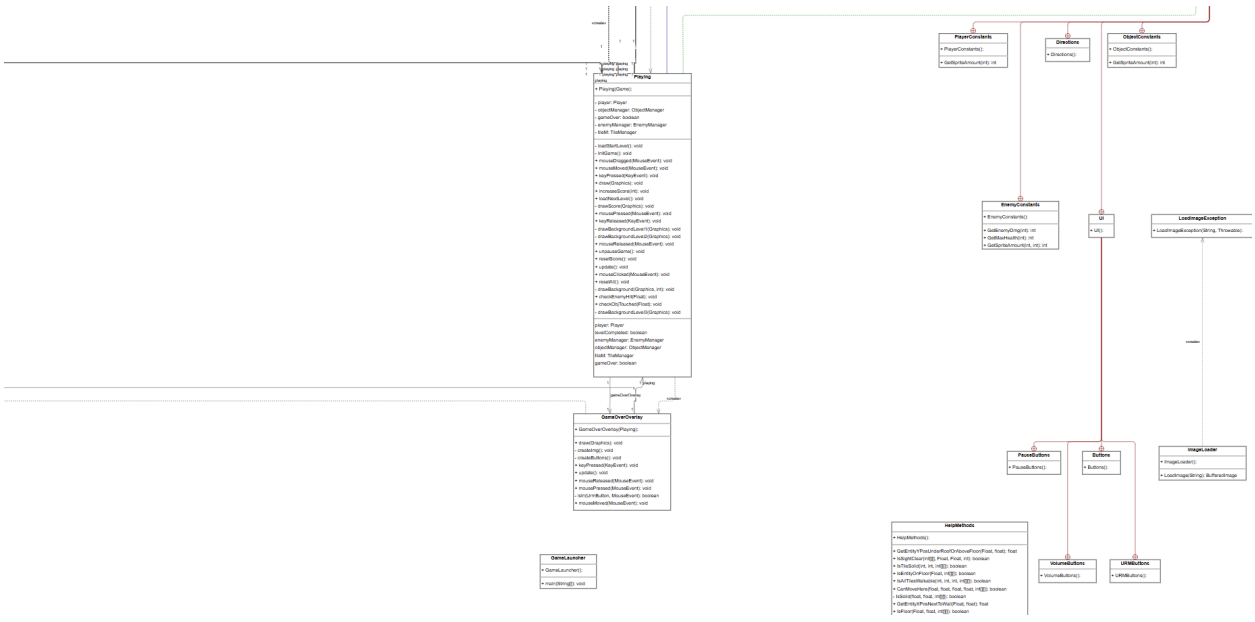


Diagrama de clase




«interface»
EnemyFactory
+ createMeleeEnemy(int, int, Playing): Enemy
+ createMummy(int, int, Playing): Enemy
+ createCentipede(int, int, Playing): Enemy
+ createUndead(int, int, Playing): Enemy
+ createBoss(int, int, Playing): Enemy
+ createSkeleton(int, int, Playing): Enemy
Hyena
+ Hyena(float, float, Playing)
+ renderGraphics(): void
+ tick(): void
- initMaskBox(): void
+ flip(): int
+ drawMaskBoxGraphics(): void
- updateMaskBox(): void
- updateBehavior(int[][], Player): void
- initAnimations(): void
+ update(int[][], Player): void
+ flip(): int
Bloated
+ Bloated(float, float, Playing)
+ tick(): void
- updateBehavior(int[][], Player): void
- drawMaskBoxGraphics(): void
+ flip(): int
- initAnimations(): void
- updateMaskBox(): void
- initMaskBox(): void
+ flip(): int
+ renderGraphics(): void
+ update(int[][], Player): void
CentSpade
+ Centspade(float, float, Playing)
+ tick(): void
+ flip(): int
- initMaskBox(): void
+ renderGraphics(): void
+ flip(): int
- initAnimations(): void
- updateBehavior(int[][], Player): void
- drawMaskBoxGraphics(): void
+ update(int[][], Player): void
Lvl1EnemyFactory
+ Lvl1EnemyFactory()
+ createCentipede(int, int, Playing): Enemy
+ createHyena(int, int, Playing): Enemy
+ createMummy(int, int, Playing): Enemy
+ createSkeleton(int, int, Playing): Enemy
+ createBloated(int, int, Playing): Enemy
+ createUndead(int, int, Playing): Enemy
Lvl1EnemyFactory
+ Lvl1EnemyFactory()
Lvl2EnemyFactory
+ Lvl2EnemyFactory()
Enemy
- updateAttackBox(): void
+ render(Graphics): void
+ flip(): int
- initMaskBox(): void
+ tick(): void
+ drawMaskBoxGraphics(): void
- updateAttackBox(): void
- updatePos(): void
- drawAttack(Graphics): void
- checkAttack(): void
- changePower(int): void
- updateAnimationTick(): void
+ changeHealth(int): void
+ powerAttack(): void
+ resetEnd(Boolean): void
- resetAni(): void
+ render(Graphics): void
- drawMaskBox(Graphics): void
right: boolean
up: boolean
jump: boolean
left: boolean
currentHealth: int
attacking: boolean
down: boolean
player
KeyboardInput
+ KeyboardInput(Game)
+ keyReleased(KeyEvent): void
+ keyPressed(KeyEvent): void
+ keyTyped(KeyEvent): void
keyboardInput
MouseInput
+ mouseClicked(MouseEvent): void
+ mouseMoved(MouseEvent): void
+ mouseReleased(MouseEvent): void
+ mousePressed(MouseEvent): void
mouseInput
GameWindow
+ GameWindow(String, int, int)
- height: int
- width: int
- canvas: Canvas
- frame: JFrame
+ BuildGameWindow(): void
height: int
canvas: Canvas
width: int
frame: JFrame
Game
DatabaseManager
+ DatabaseManager()
- loadDriver(): void
+ createTables(): void
- saveLevelData(int): void
+ saveObjectsData(ObjectManager): void
- clearTables(): void
+ savePlayerData(Player, int): void
- createDatabase(): void
+ saveEnemiesData(EnemyManager): void
+ saveCompletedLevelData(int, int, int, int): void
enemiesData: ArrayList<EnemyData>
levelValue: int
databaseManager
Tile
+ Tile(BufferedImage)
+ sprite: BufferedImage
sprite: BufferedImage
tile
Level
+ Level(String, BufferedImage[])
Lvl1EnemyFactory
+ Lvl1EnemyFactory()
+ createUndead(int, int, Playing): Enemy
+ createHyena(int, int, Playing): Enemy
+ createCentipede(int, int, Playing): Enemy
+ createBloated(int, int, Playing): Enemy
+ createMummy(int, int, Playing): Enemy
Lvl2EnemyFactory
+ Lvl2EnemyFactory()
+ createUndead(int, int, Playing): Enemy
+ createHyena(int, int, Playing): Enemy
+ createSkeleton(int, int, Playing): Enemy
+ createMummy(int, int, Playing): Enemy
+ createBloated(int, int, Playing): Enemy
+ createCentipede(int, int, Playing): Enemy
EnemyManager
+ EnemyManager(Playing)
- enemyFactory: EnemyFactory
- enemies: ArrayList<Enemy>
+ update(int[][], Player): void
+ checkEnemyHit(Float): void
+ loadEnemiesFromDatabase(): void
+ initArrayLevel(): void
+ draw(Graphics): void
+ resetAllEnemies(): void
enemyFactory: EnemyFactory
enemies: ArrayList<Enemy>
enemyManager
Game
- Game(String, int, int)
- menu: Menu
- playing: Playing
+ startGame(): void
- Update(): void
- Draw(): void
+ run(): void
+ getInstance(String, int, int): Game
- initGame(): void
menu: Menu
playing: Playing
game
Level
+ Level(String, BufferedImage[])
- path: String
- mapTile: int[][]
+ printMap(): void
+ loadMap(String): void
mapTile: int[][]
levels
TileManager
+ TileManager(Game)
- tvIndex: int
+ printMap(): void
+ drawGraphics(): void
+ loadNextLevel(): void
- buildAllLevels(): void
currentLevel: Level
amountOfLevels: int
tvIndex: int
tileM
Constants
+ Constants()
ObjectManager
Playing
+ Playing(Game)
- player: Player
- objectManager: ObjectManager
- gameOver: boolean
- enemyManager: EnemyManager
- tileM: TileManager
PlayerConstants
+ PlayerConstants()
+ GetSpriteAmount(int): int
Directions
+ Directions()
ObjectConstants
+ ObjectConstants()
+ GetSpriteAmount(int): int


Playing
- player: Player
- objectManager: ObjectManager
- gameOver: boolean
- enemyManager: EnemyManager
- tileMl: TileManager
- loadNextLevel(): void
- initGame(): void
+ mouseDragged(MouseEvent): void
+ mouseMoved(MouseEvent): void
+ keyPressed(KeyEvent): void
+ draw(Graphics): void
+ increaseScore(int): void
+ loadNextLevel(): void
- drawScore(Graphics): void
+ mousePressed(MouseEvent): void
+ keyReleased(KeyEvent): void
- drawBackgroundLevel1(Graphics): void
- drawBackgroundLevel2(Graphics): void
+ mouseReleased(MouseEvent): void
+ mouseGame(): void
+ resetScore(): void
+ update(): void
+ mouseClicked(MouseEvent): void
+ resetAll(): void
- drawBackgroundLevel3(Graphics, int): void
+ checkEnemyHit(Float): void
+ checkObjTouched(Float): void
- drawBackgroundLevel3(Graphics): void
player: Player
levelCompleted: boolean
enemyManager: EnemyManager
objectManager: ObjectManager
tileMl: TileManager
gameOver: boolean
GameOverOverlay
+ GameOverOverlay(Playing)
+ draw(Graphics): void
- creating(): void
- createButtons(): void
+ keyPressed(KeyEvent): void
+ update(): void
+ mouseReleased(MouseEvent): void
+ mousePressed(MouseEvent): void
- isIn(UrmButton, MouseEvent): boolean
+ mouseMoved(MouseEvent): void
GameLauncher
+ GameLauncher():
+ main(String[]): void
PlayerConstants
+ PlayerConstants():
+ GetSpriteAmount(int): int
Directions
+ Directions():
ObjectConstants
+ ObjectConstants():
+ GetSpriteAmount(int): int
EnemyConstants
+ EnemyConstants():
+ GetEnemyDmg(int): int
+ GetMaxHealth(int): int
+ GetSpriteAmount(int, int): int
UI
+ UI():
LoadImageException
+ LoadImageException(String, Throwable):
PauseButtons
+ PauseButtons():
Buttons
+ Buttons():
ImageLoader
+ ImageLoader():
+ LoadImage(String): BufferedImage
HelpMethods
+ HelpMethods():
+ GetEntityYPosUnderRoofOrAboveFloor(Float, float): float
+ IsRightClear(int[][], Float, Float, int): boolean
+ IsTileSolid(int, int, int[][]): boolean
+ IsWidthOrFloor(Float, int[][]): boolean
+ IsAllTilesWalkable(int, int, int, int[][]): boolean
+ CanMoveHere(float, float, float, float, int[][]): boolean
- IsSolid(float, float, int[][]): boolean
+ GetEntityXPosNextToWall(Float, float): float
+ IsFloor(Float, float, int[][]): boolean
VolumeButtons
+ VolumeButtons():
UrmButtons
+ UrmButtons():


Playing
- player: Player
- objectManager: ObjectManager
- gameOver: boolean
- enemyManager: EnemyManager
- tileMl: TileManager
- loadNextLevel(): void
- initGame(): void
+ mouseDragged(MouseEvent): void
+ mouseMoved(MouseEvent): void
+ keyPressed(KeyEvent): void
+ draw(Graphics): void
+ increaseScore(int): void
+ loadNextLevel(): void
- drawScore(Graphics): void
+ mousePressed(MouseEvent): void
+ keyReleased(KeyEvent): void
- drawBackgroundLevel1(Graphics): void
- drawBackgroundLevel2(Graphics): void
+ mouseReleased(MouseEvent): void
+ mouseGame(): void
+ resetScore(): void
+ update(): void
+ mouseClicked(MouseEvent): void
+ resetAll(): void
- drawBackgroundLevel3(Graphics, int): void
+ checkEnemyHit(Float): void
+ checkObjTouched(Float): void
- drawBackgroundLevel3(Graphics): void
player: Player
levelCompleted: boolean
enemyManager: EnemyManager
objectManager: ObjectManager
tileMl: TileManager
gameOver: boolean
GameOverOverlay
+ GameOverOverlay(Playing)
+ draw(Graphics): void
- creating(): void
- createButtons(): void
+ keyPressed(KeyEvent): void
+ update(): void
+ mouseReleased(MouseEvent): void
+ mousePressed(MouseEvent): void
- isIn(UrmButton, MouseEvent): boolean
+ mouseMoved(MouseEvent): void
GameLauncher
+ GameLauncher():
+ main(String[]): void
PlayerConstants
+ PlayerConstants():
+ GetSpriteAmount(int): int
Directions
+ Directions():
ObjectConstants
+ ObjectConstants():
+ GetSpriteAmount(int): int
EnemyConstants
+ EnemyConstants():
+ GetEnemyDmg(int): int
+ GetMaxHealth(int): int
+ GetSpriteAmount(int, int): int
UI
+ UI():
LoadImageException
+ LoadImageException(String, Throwable):
PauseButtons
+ PauseButtons():
Buttons
+ Buttons():
ImageLoader
+ ImageLoader():
+ LoadImage(String): BufferedImage
HelpMethods
+ HelpMethods():
+ GetEntityYPosUnderRoofOrAboveFloor(Float, float): float
+ IsRightClear(int[][], Float, Float, int): boolean
+ IsTileSolid(int, int, int[][]): boolean
+ IsWidthOrFloor(Float, int[][]): boolean
+ IsAllTilesWalkable(int, int, int, int[][]): boolean
+ CanMoveHere(float, float, float, float, int[][]): boolean
- IsSolid(float, float, int[][]): boolean
+ GetEntityXPosNextToWall(Float, float): float
+ IsFloor(Float, float, int[][]): boolean
VolumeButtons
+ VolumeButtons():
UrmButtons
+ UrmButtons():

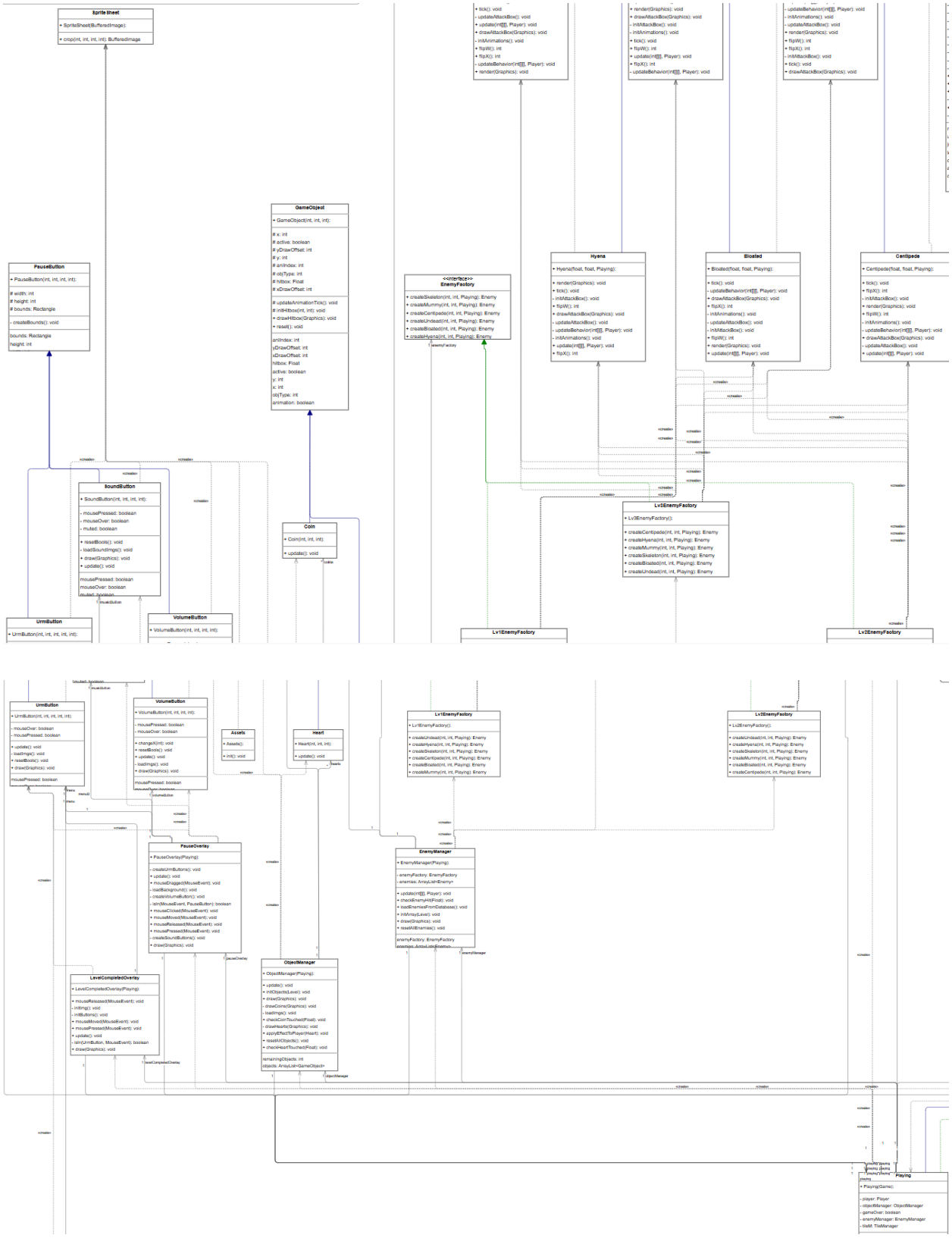


Diagrama de clase este de asemenea anexată în fișierul src.uml și în fișierul diagrama_uml.drawio.svg din interiorul proiectului, ce pot fi accesate folosind

IntelliJ Idea

GameLauncher - Această clasă conține metoda main, care este punctul de intrare în aplicație. Ea creează și pornește jocul.

GameState - Această clasă este o interfață care definește comportamentul stărilor jocului, cum ar fi meniul principal, jocul propriu-zis și alte stări pe care jocul le poate avea.

Entity - Această clasă abstractă este o clasă de bază pentru toate entitățile din joc, cum ar fi jucătorii și inamicii. Ea conține informații despre poziție, dimensiune și alte atribute comune ale entităților. In ea se regasesc campuri pentru coordonatele entitatii in joc, coordonatele pozitiei pe ecran, dimensiunile, timpul de animatie si indexul curent al animatiei, precum si cutia de coliziune.

Contine metode specifice pentru:

- desenarea cutiei de coliziuni (pentru debug) si initializare a acesteia, precum si functii ce returneaza coordonatele si dimensiunile ei.
- returnarea coordonatelor pozitiei in joc, respectiv pe ecran
- returnarea cutiei de coliziuni
- tick: Metoda abstracta pentru actualizarea logicii entitatii
- render: Metoda abstracta pentru desenarea entitatii

Player

- este o componentă esențială a jocului, responsabilă de gestionarea tuturor aspectelor legate de jucător și comportamentul său în joc.
- În codul furnizat, sunt definite variabilele și constructorul care inițializează animațiile, starea jucătorului, viteza, sănătatea, bara de putere etc. Metodele de actualizare (tick) și desenare (render) sunt responsabile pentru actualizarea stării jucătorului și desenarea acestuia pe ecran în fiecare cadru al jocului.
- De asemenea, există metode care controlează mișcarea și coliziunea jucătorului, atacurile și interacțiunile cu alte obiecte din joc și desenarea interfeței utilizatorului (UI).
- În esență, clasa Player integrează toate funcționalitățile necesare pentru a gestiona comportamentul și interacțiunea jucătorului în cadrul jocului.

Enemy - Clasa Enemy definește comportamentul de bază și proprietățile pentru inamicii din joc. Metodele incluse gestionează mișcarea, actualizarea stării, verificarea coliziunilor, animațiile și interacțiunea cu jucătorul. Clasele derivate vor implementa metodele abstracte update și render pentru a defini comportamentul și redarea specifică a fiecărui tip de inamic.

- firstUpdateCheck - Verifică dacă este prima actualizare a inamicului și stabilește dacă acesta se află în aer.
- updateInAir - Actualizează poziția și starea inamicului atunci când este în aer.
- move - Deplasarea inamicului pe orizontală.
- canSeePlayer - Verifică dacă inamicul poate vedea jucătorul.
- isPlayerInRange - Verifică dacă jucătorul se află în raza de atac a inamicului.
- isPlayerCloseForAttack - Verifică dacă jucătorul este suficient de aproape pentru

a fi atacat.
- hurt - Aplică daune inamicului și actualizează starea acestuia în consecință.
- checkEnemyHit - Verifică dacă atacul inamicului a lovit jucătorul și aplică daune dacă este cazul.
-

Hyena, Skeleton, Mummy, Player, Undead, Centipede, Bloated - Fiecare dintre aceste clase reprezintă o entitate specifică a unui inamic în joc. Metodele sunt asemănătoare, însă ceea ce le deosebește este compartamentul și aspectul specific al fiecărui tip de inamic. Ele includ logica mișcării, detecția jucătorului, atacul și afișarea graficelor corespunzătoare.

- De asemenea, aceste clase extind clasa abstractă Enemy, ceea ce înseamnă că moștenesc comportamentul general al inamicului. Însă ele adaugă și modifică funcționalitatea pentru a se potrivi specificațiilor inamicului.
- Conțin metode pentru initializarea animațiilor specifice, initializarea hitbox-ului pentru atac, actualizarea comportamentului în funcție de starea curentă a jocului și poziția jucătorului, actualizarea animației în funcție de starea inamicului și trecerea timpului, desenarea inamicului în poziția și starea sa curentă.

EnemyManager

- este o clasă centrală în joc responsabilă de gestionarea inamicilor. Aceasta gestionează crearea, actualizarea și desenarea inamicilor în funcție de starea jocului și acțiunile jucătorului.
- Folosește o listă de inamici pentru a-i urmări și a le aplica actualizările necesare în fiecare cadru de joc. De asemenea, se ocupă de verificarea și gestionarea coliziunilor între inamici și jucător și oferă funcționalitatea de a reseta starea inamicilor la valorile inițiale. Prin intermediul acestei clase, jocul poate menține o interacțiune dinamică și eficientă cu inamicii, oferind o experiență captivantă și plină de acțiune pentru jucător.

Animation

- este o clasă responsabilă de gestionarea animațiilor în joc. Aceasta stochează o serie de cadre (frames) sub formă de imagini și controlează afișarea lor pentru a crea iluzia mișcării.
- Clasa este proiectată pentru a itera prin cadrele de animație în funcție de un anumit interval de timp specificat de utilizator. Atunci când este apelată metoda tick(), clasa calculează timpul scurs între apelurile succesive și, în funcție de acest timp și de viteza specificată, trece la următorul cadru din animație.

HelpMethods - Această clasă conține funcții de ajutor pentru gestionarea mișcării și coliziunilor entităților în joc. Funcțiile utilizează datele nivelului pentru a determina dacă anumite acțiuni sunt permise sau pentru a calcula pozițiile precise ale entităților în raport cu pereții, podeaua sau alte obstacole.

> CanMoveHere - Verifică dacă fiecare colț al dreptunghiului definit de poziție și dimensiuni (colțul stânga sus, colțul dreapta jos, colțul dreapta sus și colțul stânga jos)

nu este solid.

> IsTileSolid - Verifică dacă un tile specific este solid. Returnează true dacă valoarea in matrice corespunde unui tile solid, altfel false

> IsSolid - Verifică dacă un punct specific (x, y) din lumea jocului este solid sau nu.

> GetEntityXPosNextToWall - Calculează poziția exactă pe axa X unde o entitate ar trebui să se oprească atunci când se lovește de un perete.

> GetEntityYPosUnderRoofOrAboveFloor - Calculează poziția exactă pe axa Y unde o entitate ar trebui să se oprească atunci când se lovește de tavan sau podea.

> IsEntityOnFloor - Verifică punctele de coliziune de la colțurile stânga jos și dreapta jos ale hitbox-ului pentru a vedea dacă sunt pe un tile solid.

> IsFloor - Verifică dacă punctul din colțul stânga jos sau dreapta jos, în funcție de viteza pe axa x, este pe un tile solid.

> IsAllTilesWalkable - Parcurge și verifică dacă toate tile-urile dintre două puncte pe axa X și la un anumit nivel pe axa Y sunt solide.

> IsSightClear - Verifică dacă drumul (linia vizuală) dintre două entități este liber de obstacole solide. Determină tile-urile pe axa X pentru cele două hitbox-uri si apoi apelează IsAllTilesWalkable pentru a verifica dacă toate tile-urile dintre aceste două puncte sunt solide.

Constants

- centralizează toate constantele folosite în joc, făcându-le accesibile și ușor de gestionat. Aceasta include valori pentru gravitație (GRAVITY), detalii despre diverse obiecte de joc (în cadrul clasei interne ObjectConstants), cum ar fi monedele și inimile, specificații pentru elementele UI (UI), și caracteristici ale inamicilor (EnemyConstants) și ale jucătorului (PlayerConstants).
- ObjectConstants definește tipurile de obiecte (monede, inimi), dimensiunile lor implicite și scalate, precum și numărul de sprite-uri pentru animații. EnemyConstants și PlayerConstants definesc stările, dimensiunile, daunele, sănătatea maximă și numărul de cadre pentru animațiile inamicilor și jucătorului.
- Prin centralizarea acestor constante, codul devine mai clar, mai ușor de întreținut și de ajustat.

Assets - este responsabilă pentru gestionarea încărcării și stocării resurselor grafice utilizate în joc. Ea conține o serie de tablouri de imagini, fiecare corespunzând unui anumit element grafic din joc, cum ar fi jucătorul, inamicii, medii de fundal etc.

ImageLoader

- este responsabilă de încărcarea imaginilor din fișierele sursă și transformarea acestora în obiecte BufferedImage. Metoda LoadImage(String path) primește ca argument calea relativă către fișierul imagine și returnează un obiect BufferedImage care conține imaginea încărcată.
-

LoadImageException

- este o clasă de excepție personalizată, derivată din clasa Exception. Ea este folosită pentru a gestiona situațiile în care apare o eroare în timpul încărcării imaginilor din fișierele sursă. Constructorul acestei clase primește un mesaj care

explică problema și o cauză (Throwable) care indică excepția originală care a provocat eroarea. Prin intermediul acestei clase, se poate gestiona și trata în mod corespunzător orice eroare legată de încărcarea imaginilor.

SpriteSheet - este responsabilă pentru gestionarea unei imagini formate din dale (sprite sheet) și pentru extragerea dalelor individuale din această imagine.

GameObject

- servește drept schelet de bază pentru toate obiectele din joc în aplicația noastră. Ea gestionează aspecte esențiale cum ar fi poziția, hitbox-ul pentru coliziuni, starea de activitate și animație a obiectului. Prin intermediul acestei clase, obiectele din joc sunt standardizate în ceea ce privește comportamentul și caracteristicile lor de bază.
- de exemplu, prin metoda reset() se poate reinițializa starea obiectului, iar prin initHitbox() se poate inițializa hitbox-ul. Această abordare ajută la menținerea unui cod organizat și ușor de gestionat, oferind totodată flexibilitate pentru dezvoltarea și extinderea funcționalităților specifice ale obiectelor jocului.

Coin - este o subclasă a clasei GameObject și reprezintă un obiect de tip monedă în joc. Ea moștenește comportamentul de bază al clasei GameObject și își definește propriile caracteristici specifice monedei, cum ar fi dimensiunea și offset-urile de desenare.

Heart - este o altă subclasă a clasei GameObject și reprezintă un obiect de tip inimă în joc. Similar clasei Coin, ea moștenește comportamentul de bază al clasei GameObject, dar își definește propriile caracteristici specifice inimii, cum ar fi dimensiunea și offset-urile de desenare, precum și o viteză de animație diferită.

ObjectManager

- Este responsabilă pentru gestionarea și coordonarea obiectelor din joc în cadrul aplicației noastre. Aici sunt implementate funcționalități esențiale precum încărcarea imaginilor pentru obiecte, inițializarea și actualizarea acestora în funcție de progresul jocului și interacțiunile cu jucătorul.
- Principalul său scop este de a gestiona două tipuri principale de obiecte din joc: monede și inimi. Ea se ocupă de următoarele aspecte:
  > Inițializare și gestionarea obiectelor: Clasa încarcă imaginile necesare pentru monede și inimi folosind un obiect SpriteSheet, după care creează și inițializează listele de obiecte corespunzătoare. Metoda initObjects este folosită pentru a adăuga monede și inimi în funcție de nivelul curent.
  > Metodele checkCoinTouched și checkHeartTouched verifică dacă jucătorul a interacționat cu o monedă sau o inimă și aplică efectele corespunzătoare în cazul unei coliziuni.
  > Metoda update este responsabilă pentru actualizarea stării obiectelor în fiecare cadru al jocului. Aceasta se asigură că animațiile sunt actualizate corect și că obiectele sunt mutate în funcție de acțiunile jucătorului.
  > Metoda draw este utilizată pentru a desena obiectele pe ecran în fiecare cadru al jocului. Aceasta se ocupă de calculul poziției fiecărui obiect în funcție de

poziția jucătorului și desenarea lor pe ecran.
> Metoda resetAllObjects este folosită pentru a reseta starea tuturor obiectelor, aducându-le înapoi în starea lor inițială pentru un nou joc sau un nivel nou.

Level

- Este componenta responsabilă cu încărcarea și gestionarea hărții nivelului în joc. Ea primește calea către fișierul de hartă și o serie de imagini de dale, pe care le utilizează pentru a construi harta nivelului.
- Prin intermediul metodei loadMap(), citeste fișierul de hartă și creează o matrice de numere întregi pentru a reprezenta configurarea hărții. Această matrice este utilă pentru alte componente ale jocului pentru a ști locația și tipul dalelor și obiectelor în cadrul nivelului. De asemenea, clasa Level oferă metoda printMap() pentru debugare și acces la calea către fișierul de hartă prin getPath().

TileManager

- Este o clasă responsabilă pentru gestionarea dalelor și a nivelurilor în joc. Ea încarcă și stochează nivelurile, oferă funcționalitate pentru schimbarea nivelurilor, și desenează dalele corespunzătoare pe ecran în funcție de poziția și vizibilitatea jucătorului.
- Metoda loadNextLevel este responsabilă cu încărcarea următorului nivel. Ea actualizează indexul nivelului curent și, dacă se ajunge la finalul listei de niveluri, se revine la primul nivel și se comută starea jocului la meniu. De asemenea, actualizează datele jucătorului și ale inamicilor pentru noul nivel.
- Metoda draw desenează dalele nivelului curent pe ecran. Parcurge matricea de dale și, pentru fiecare dintre ele, determină dacă trebuie să fie desenată pe ecran în funcție de poziția și vizibilitatea jucătorului. Apoi, desenează dalele respective utilizând grafica furnizată și dimensiunile dalei din cadrul jocului.

PauseButton - este o clasă simplă care definește comportamentul unui buton în interfața utilizatorului. Cu ajutorul coordonatelor și dimensiunilor sale, acesta poate fi plasat și vizualizat pe ecran. Utilizând un obiect Rectangle pentru a defini zona de coliziune, butonul poate detecta interacțiunile cu mouse-ul. Această clasă furnizează, de asemenea, metode pentru a obține dimensiunile butonului și dreptunghiul său de coliziune.

VolumeButton - extinde funcționalitatea PauseButton pentru a gestiona setările de volum ale jocului. Această clasă include imagini pentru stările de volum activat și dezactivat. Prin metoda update(), starea butonului este actualizată în funcție de interacțiunea cu mouse-ul. Astfel, utilizatorul poate controla volumul jocului printr-un simplu clic.

SoundButton - este o extensie a clasei PauseButton, specializată în controlul audio al jocului. Cu ajutorul imaginilor corespunzătoare stării audio, acest buton permite utilizatorului să activeze sau să dezactiveze sunetul. Metoda update() actualizează imaginea butonului în funcție de interacțiunea cu mouse-ul, oferind o experiență interactivă plăcută.

UrmButton - adaugă funcționalitatea de navigare în interfața utilizatorului, extinzând clasa PauseButton. Cu ajutorul imaginilor pentru diferite stări ale butonului, acesta facilitează navigarea utilizatorului în cadrul jocului. Prin intermediul metodei update(), starea butonului este actualizată în funcție de interacțiunea cu mouse-ul, permițând o navigare fluidă și intuitivă.

PauseOverlay

- gestionează afișarea și interacțiunile asociate meniului de pauză. În constructor, se încarcă imaginea de fundal și se creează butoanele pentru controlul sunetului și navigarea în meniu. Metoda update() actualizează stările butoanelor în funcție de interacțiunile utilizatorului, iar metoda draw() desenează toate elementele meniului de pauză pe ecran
- Metodele pentru gestionarea evenimentelor mouse-ului, cum ar fi mousePressed() sau mouseReleased(), sunt responsabile pentru tratarea interacțiunilor cu butoanele și ajustarea stărilor jocului în consecință.
- Prin intermediul acestor funcționalități, clasa PauseOverlay oferă o interfață intuitivă și eficientă pentru utilizator, facilitând controlul și navigarea în cadrul jocului.

GameOverOverlay

- este responsabilă de afișarea și gestionarea meniului care apare atunci când jocul se încheie. În constructor, imaginea de fundal și butoanele necesare pentru navigarea în meniu sunt create și inițializate. Metoda draw() desenează fundalul opac și imaginea meniului game over pe ecran, împreună cu butoanele corespunzătoare.
- Butonul "Menu" și butonul "Play Again" sunt create și afișate în pozițiile corespunzătoare. Metoda update() este responsabilă pentru actualizarea stării butoanelor, iar metodele mouseMoved(), mousePressed() și mouseReleased() gestionează interacțiunile utilizatorului cu aceste butoane

LevelCompletedOverlay

- se ocupă de afișarea și gestionarea meniului care apare atunci când un nivel este completat. În constructor, imaginea de fundal și butoanele necesare pentru navigarea în meniu sunt create și inițializate.
- Metoda draw() desenează imaginea de fundal a nivelului completat și butoanele "Menu" și "Next Level" pe ecran, în pozițiile corespunzătoare. Metoda update() este responsabilă pentru actualizarea stării butoanelor în funcție de acțiunile utilizatorului.
- Metodele mouseMoved(), mousePressed() și mouseReleased() gestionează interacțiunile utilizatorului cu aceste butoane.

MenuButton

- reprezintă un buton din meniul principal al jocului și gestionează afișarea, interacțiunea și funcționalitatea acestuia. În constructor, se specifică poziția butonului, indicele său (pentru a determina imaginea corespunzătoare) și starea

de joc asociată cu acțiunea butonului.

- Metoda loadImgs() încarcă imaginile necesare pentru starea activă și inactivă a butonului, în funcție de indicele butonului. Metoda initBounds() inițializează limitele butonului în funcție de poziția sa și dimensiunile imaginii.
- Metoda draw(Graphics g) desenează butonul pe ecran, utilizând imaginea corespunzătoare stării sale active sau inactive. Metoda update() este responsabilă pentru actualizarea stării butonului în funcție de acțiunile utilizatorului, cum ar fi suprapunerea cursorului sau apăsarea butonului.

Menu

- reprezintă starea jocului asociată cu meniul principal și gestionează afișarea și interacțiunea cu butoanele din acesta. Constructorul încarcă imaginile necesare și inițializează butoanele și fundalul meniului. Metoda loadBackground() determină dimensiunile și poziția fundalului, iar metoda loadButtons() creează butoanele și le atribuie stările de joc corespunzătoare.
- Metodele update() și draw(Graphics g) sunt responsabile pentru actualizarea și desenarea stării meniului. În update(), fiecare buton este actualizat pentru a reflecta starea sa curentă, iar în draw(Graphics g) se desenează fundalul și fiecare buton.
- Metodele mousePressed(MouseEvent e) și mouseReleased(MouseEvent e) gestionează evenimentele de apăsare și eliberare a mouse-ului. Ele determină dacă vreun buton a fost apăsat și, dacă da, aplică starea de joc asociată cu acțiunea butonului respectiv și resetează stările butoanelor.
- Metoda mouseMoved e responsabilă pentru detectarea suprapunerii cursorului mouse-ului peste butoane. Ea setează starea de suprapunere (mouseOver) pentru fiecare buton în funcție de poziția cursorului
- Metodele keyPressed și keyReleased gestionează evenimentele de apăsare și eliberare a tastelor. În acest caz, se detectează dacă a fost apăsată tasta ESC și, dacă da, se revine la starea de joc PLAYING.

Playing:

- este o componentă esențială a sistemului de stat al jocului și este responsabilă pentru gestionarea logicii și afișarea elementelor de joc în timpul desfășurării acestuia. Aceasta acționează ca o stare de joc activă în care jucătorul interacționează cu mediul și se desfășoară acțiunea.

- Inițializarea jocului: Constructorul Playing(Game game) initializează jocul prin încărcarea tuturor resurselor necesare și inițializarea obiectelor precum jucătorul, gestionarul de dale, managerul de inamici. Metoda InitGame este responsabilă pentru această inițializare și încărcarea nivelului inițial.

- Actualizarea și desenarea jocului: Metoda update este apelată pentru a actualiza starea jocului în fiecare cadru, iar metoda draw este responsabilă pentru desenarea tuturor elementelor de joc pe ecran. Aceasta include desenarea dalelor, a jucătorului, a inamicilor, a obiectelor și a altor elemente grafice precum scorul și suprapunerile.

- Interacțiunea cu input-ul utilizatorului: Metodele keyPressed și keyReleased gestionează evenimentele de apăsare și eliberare a tastelor de la tastatură, iar metodele mousePressed, mouseReleased și mouseMoved se ocupă de evenimentele generate de mouse. Aceste metode reacționează la inputul utilizatorului și actualizează starea jocului în funcție de acțiunile acestuia.

- Gestionarea nivelurilor și a stărilor de joc: Clasa gestionează tranzițiile între diferitele stări ale jocului, precum pauza, finalizarea nivelului și sfârșitul jocului. Metoda loadNextLevel încarcă următorul nivel, iar metoda resetAll resetează starea jocului pentru a începe din nou sau pentru a trece la alt nivel. De asemenea, clasele asociate precum GameOverOverlay, LevelCompletedOverlay și PauseOverlay sunt utilizate pentru a gestiona suprapunerile și afișările specifice acestor stări.
- Manipularea elementelor de joc: Metodele checkEnemyHit și checkObjTouched sunt responsabile pentru detectarea coliziunilor dintre jucător și inamici sau obiecte. Acestea gestionează logica de joc și interacțiunea dintre diferitele entități din mediu.
- Desenarea fundalului: Metodele drawBackgroundLevel1, drawBackgroundLevel2 și drawBackgroundLevel3 sunt utilizate pentru a desena fundalul specific fiecărui nivel. Acestea sunt responsabile pentru crearea atmosferei și a ambientului în funcție de tema și setările nivelului.

KeyboardInput - este responsabilă pentru captarea evenimentelor de la tastatură și direcționarea lor către starea corespunzătoare a jocului. Prin implementarea interfeței KeyListener, această clasă poate detecta apăsările de taste și poate executa acțiuni precum mutarea jucătorului sau activarea/dezactivarea pauzelor.

MouseInput - gestionează interacțiunile cu mouse-ul, implementând interfețele MouseListener și MouseMotionListener. Această clasă poate detecta clicuri de mouse, mutări și trageri, și poate reacționa în consecință, permițând jucătorului să interacționeze cu elementele din joc, cum ar fi meniurile sau obiectele din mediul de joc.

Game
- reprezintă nucleul jocului, coordonând toate componentele și acțiunile din cadrul acestuia.
- Este responsabilă de gestionarea ferestrei de joc, a inputului utilizatorului și a stării curente a jocului. Prin intermediul metodei InitGame(), se inițializează toate componentele necesare, precum managerul bazei de date, meniul și mediul de joc.
- Metoda run() conține bucla principală a jocului, în care sunt actualizate și desenate cadrele în funcție de starea jocului și de evenimentele de input.
- Metodele StartGame() și Update() sunt responsabile pentru pornirea și actualizarea jocului, respectiv, în timp ce Draw() se ocupă de desenarea graficii pe ecran.
- Clasa gestionează, de asemenea, toate stările posibile ale jocului, inclusiv

meniul, mediul de joc și alte stări specifice, cum ar fi încărcarea sau ieșirea din joc.
- metoda Update gestionează logica de actualizare a jocului în funcție de starea sa curentă. Aceasta include stările de meniu, joc activ și încărcare. Când starea jocului este LOAD, metoda Update() are responsabilitatea de a încărca indexul nivelului salvat în baza de date și de a inițializa nivelul corespunzător în joc.

EnemyData - este o simplă clasă de date care reprezintă informațiile despre un inamic din joc, inclusiv coordonatele x și y și tipul acestuia. Această clasă este folosită pentru a organiza și transmite datele despre inamici între DatabaseManager și alte părți ale codului jocului.

DatabaseManager
- gestionează interacțiunea cu baza de date a jocului, oferind metode pentru crearea tabelelor, salvarea datelor despre jucător, inamici, obiecte și niveluri, precum și alte operații de manipulare a datelor.
- Metoda createDatabase() inițializează baza de date și apelează metoda createTables() pentru a crea toate tabelele necesare.
- Metoda createTables() se ocupă de crearea tabelelor pentru jucători, inamici, obiecte, niveluri și niveluri completate, asigurându-se că acestea există în baza de date.
- Metodele savePlayerData(), saveEnemiesData(), saveObjectsData(), saveLevelData() și saveCompletedLevelData() sunt responsabile pentru salvarea datelor jucătorului, inamicilor, obiectelor, nivelurilor și nivelurilor completate în baza de date, respectiv.
- Metoda clearTables() permite golirea tuturor tabelelor din baza de date, utilă în anumite situații, cum ar fi resetarea jocului.
- Metoda getLevelValue() obține valoarea indexului nivelului curent din baza de date.
- Metoda getEnemiesData() extrage și returnează datele despre inamici din baza de date, sub forma unei liste de obiecte EnemyData, care conțin informații despre poziția și tipul inamicilor.
- Clasa EnemyData este o simplă clasă de date care reprezintă informațiile despre un inamic din joc, inclusiv coordonatele x și y și tipul acestuia. Această clasă este folosită pentru a organiza și transmite datele despre inamici între DatabaseManager și alte părți ale codului jocului.

Baza de date conține mai multe tabele, fiecare având rolul său specific în stocarea datelor despre joc și progresul jucătorului. Iată o scurtă descriere a fiecărei tabele:

> Tabela "Player": Această tabelă stochează informații despre jucător, cum ar fi poziția (coordonatele x și y), sănătatea și scorul acestuia.

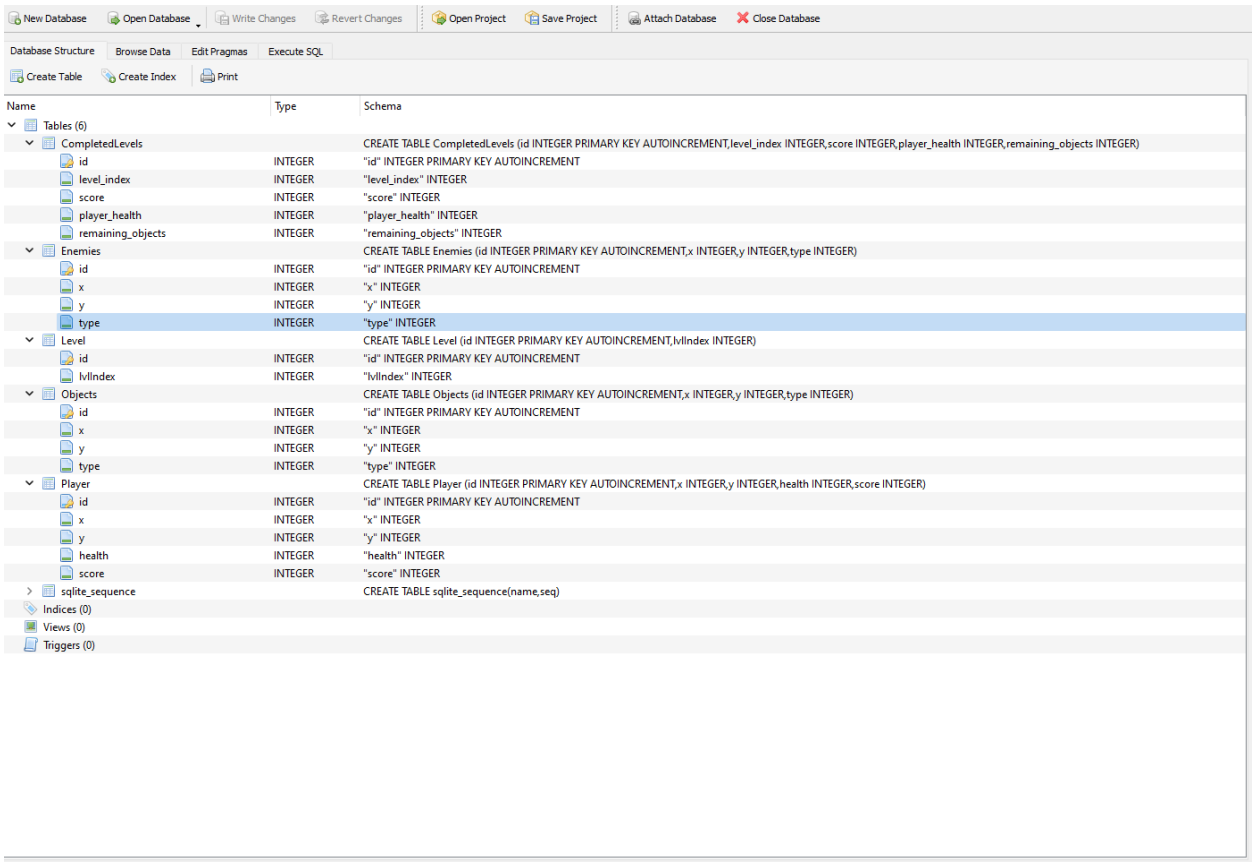
> Tabela "Enemies": Aici sunt stocate date despre inamicii din joc, inclusiv poziția lor (coordonatele x și y) și tipul lor.

> Tabela "Objects": Această tabelă păstrează informații despre obiectele din joc, precum poziția lor (coordonatele x și y) și tipul de obiect.

> Tabela "Level": Stochează indexul nivelului curent în care se află jucătorul în joc.

> Tabela "CompletedLevels": Aici sunt înregistrate date despre nivelurile completate, inclusiv indexul nivelului, scorul jucătorului, sănătatea rămasă a jucătorului și numărul de obiecte rămase neatinse.

Fiecare tabelă are câmpurile sale specifice, iar datele sunt organizate în tabele pentru a permite o gestionare eficientă și structurată a informațiilor jocului. Aceste tabele sunt esențiale pentru stocarea și recuperarea datelor necesare pentru funcționarea corectă și continuarea progresului jocului.



Bibliografie

Game Sprites inamici Swamp - https://craftpix.net/freebies/free-swamp-bosses-pixel-art-character-pack/

Game sprites
Inamici desert - https://free-game-assets.itch.io/free-enemy-sprite-sheets-pixel-art

Game sprites Skeleton - https://luizmelo.itch.io/monsters-creatures-fantasy

Game sprites Aiden - https://rvros.itch.io/animated-pixel-hero

Tiles
- https://ansimuz.itch.io/magic-cliffs-environment
- https://untiedgames.itch.io/kasakasa-desert-pixel-art-tileset
- https://theflavare.itch.io/forest-nature-fantasy-tileset

2D Game Development: From Zero to Hero:https://www.bookfusion.com/books/937506-2d-game-development-from-zero-to-hero\