Knight’s Legacy: Reign of Justice

Realizat de Bîrleanu Andreea

Grupa 1207B, 2023-2024

Proiectarea contextului

Povestea jocului

Într-un regat îndepărtat, trona un rege drept și iubit de poporul său. Cu toate acestea, fericirea regatului era umbrită de un personaj malefic, Lordul Malachar, care își pusese ochii pe tron. Pentru a-și îndeplini planul diabolic, Malachar îl răsturnase pe rege și își instaurase tirania în regat.
În miez de noapte, într-o tavernă dintr-un sat îndepărtat, un tânăr cavaler, pe nume Sir Arthur, auzea vorbele disperate ale oamenilor loviți de nedreptatea lordului tiran. Sir Arthur, cu inima plină de curaj și dreptate, a jurat să aducă pacea și libertatea înapoi în regat. Cu inima aprinsă și cu sabia strălucitoare, Sir Arthur a părăsit taverna în liniștea nopții, hotărât să pornească într-o călătorie primejdioasă pentru a aduce schimbarea. În drumul său, a traversat sate pustii, vârfurile muntoase ale regatului, a cunoscut oameni nevinovați al căror sânge fusese vărsat nedrept. Fiecare poveste a întărit hotărârea sa și a alimentat flacăra pasiunii sale pentru dreptate.
Cu fiecare pas înainte, Sir Arthur se apropria mai mult de confruntarea cu Lordul Malachar și de eliberarea regatului de sub tirania sa întunecată. Oare va reuși să-și ducă misiunea până la capăt?

Genul jocului
Acțiune, Aventură, Fantasy

Tematica jocului
Aventură

Proiectarea sistemului

Controls

- Deplasare: A-stânga, D-dreapta, SPACE-săritură,
- Folosire powerup: click dreapta mouse
- Atac: Enter
- Pauză: backspace
- S: salvare în baza de date
- L: încărcare din baza de date

Gameplay

- Jucătorul poate manipula direct doar personajul principal, cavalerul Sir Arthur.
- Jucătorul trebuie să completeze cele trei niveluri pentru a câștiga.

- Completarea unui nivel presupune omorârea tuturor inamicilor întâlniți pe traseu cu viața curentă mai mare decât zero.
- Sir Arthur își poate pierde viețile prin diverse interacțiuni cu inamicii, deși îi poate ataca, inamicii răspund la interacțiune sau atacă primii sau cu obiectele din jur.
- Datorită poțiunilor roșii de pe traseu, jucătorul își poate mări viața dacă le colectează.
- De asemenea, poțiunile albastre îi oferă o mai mare putere de atac împotriva inamicilor.

Reguli principale:

1. Supraviețuire: monitorizarea vieții curente, pierderea acestora duce la reluarea nivelului;
2. Combaterea inamicilor: folosirea sabiei și a abilităților de luptă pentru a înfrunta inamicii, evitarea atacurilor lor pentru a rămâne în viață;
3. Colectarea de resurse: adunarea de poțiuni pentru a extinde timpul de supraviețuire și pentru a căpăta o putere de atac mai mare;

Cum progresezi în joc:

- Depășirea provocărilor apărute în nivele cu viață curentă mai mare decât zero la trecerea dintre nivele
- Trecerea prin cele trei niveluri și înfrângerea inamicilor din fiecare nivel

Proiectarea conținutului

Caractere

Sir Arthur, un cavaler curajos în căutarea dreptății și libertății într-un regat întunecat de tiranie. Acesta poate interacționa cu diverse obiecte, cu inamici și se poate mișca pe dalele din care este constituită mapa. Inamicii sunt diferiți de la un nivel la altul. În primul nivel, se confruntă cu urși polari înfometați din zona muntoasă înzăpezită. Acești inamici îi pun la încercare abilitățile de luptă și de navigare într-un mediu periculos și ostil. În următorul nivel, Sir Arthur se luptă cu fantomele locuitorilor omorâți în urma tiraniei lordului, care pândesc în umbrele întunecate ale așezării părăsite. Acești inamici necunoscuți și înspăimântători îi pun la încercare curajul și rezistența în fața răului din trecut. În ultimul nivel, cavalerul se confruntă cu garda întunecată a lordului într-o luptă finală pentru destinul regatului.

Obiecte

Sir Arthur poate interacționa cu următoarele elemente:

- Poțiuni care sunt de două tipuri: roșii pentru creșterea vieții curente și albastre pentru adăugarea unei puteri suplimentare de atac.
- Un singur cufăr pentru a evidenția singletonul, odată ce e distrus va fi dezvăluit conținutul acestuia.
- Dale pe care se poate deplasa

Animații:

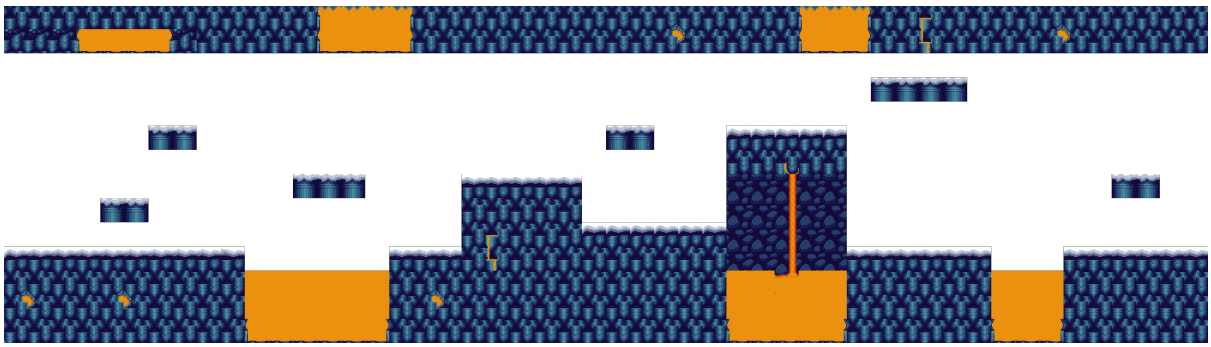
- Mers normal: Cavalerul se mișcă cu mândrie și hotărâre, cu pași siguri și grei. Își ține scutul ridicat și sabia pregătită pentru orice amenințare ce ar putea apărea în calea sa.
- Atac cu sabie: Cavalerul lovește în față cu sabia sa, efectuând o tăietură puternică și precisă.
- Săritură: Cavalerul sări cu forță și agilitate, pregătit să evite pericolele.
- Rănit: Acesta poate fi atacat de inamici și împins în urma atacului, dar își revine imediat pentru a-și continua călătoria.
- Moarte: În momentul în care cavalerul este învins, el cade la pământ.

Proiectarea nivelurilor

Jocul constă în trei niveluri, cu grad diferit de dificultate, alți inamici și un mediu diferit în care se desfășoară acțiunea.

Nivelul 1: Muntele Înzăpezit

În călătoria sa prin peisajele reci și impunătoare ale munților înzăpeziți, cavalerul Sir Arthur se confruntă cu noi provocări și pericole. Adâncurile munților ascund amenințări înspăimântătoare, iar printre acestea se numără urșii polari înfometați, gata să atace la cea mai mică mișcare greșită. Când urșii polari își simt teritoriul amenințat sau sunt în căutare de hrană, ei devin agresivi și atacă orice li se ivește în cale, inclusive pe cavaler. Cu forța și agilitatea lor impresionante, acești urși reprezintă o amenințare constantă pentru căletorul nostrum. Pe lângă urșii polari, Sir Arthur trebuie să fie vigilent în fața altor primesjdii întâlnite în munții înzăpeziți. Lava fierbinte îi stă în cale, punându-l la încercare abilitățile de navigare și reflexele.
Obiectivul în acest nivel este să navigheze cu success prin muntele înzăpezit, să înfrunte și să învingă toți inamicii care îi ies în cale și să depășească toate obstacolele pentru a progresa către următoarea parte a regatului. Cu curajul și determinarea sa neclintită, el va face față tuturor provocărilor și va continua să lupte pentru dreptate ș pace în regatul său amenințat.



Nivelul 2: Satul Părăsit

Cavalerul Sir Arthur ajunge într-un sat abandonat, odată plin de viață și activitate, acum este un loc bântuit de fantomele trecutului. În adâncurile satului, fantomele se ridică din morminte și își pândesc prada, urmărind-o și atacând-o. O singură apropriere de acestea, jucătorul își pierde viața. Cavalerul Sir Arthur se află în mijlocul acestei atmosfere înfricoșătoare, cu o misiune clară: să înfrunte forțele întunericului și să își continue călătoria. În părțile mai întunecate ale satului, capcanele și obstacolele ascunse îl pun pe Sir Arthur în pericol. Cu fiecare inamic învins și fiecare obstacol depășit, Sir Arthur se apropie tot mai mult de finalul jocului.

Nivelul 3: Cetatea Lordului

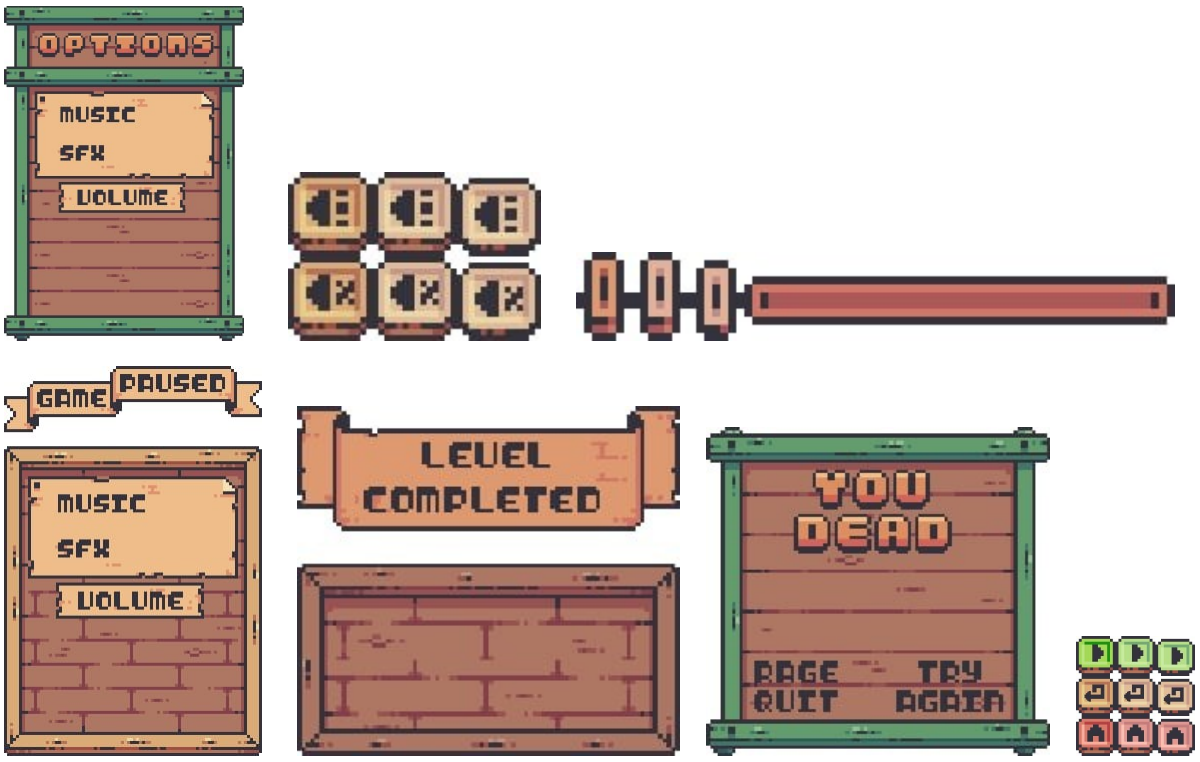
În inima întunecată a cetății, cavalerul se confruntă cu garda întunecată a maleficului lord. Coridoarele sinister și sălile întunecate sunt pline de amenințări, iar garda lui Malachar este gata să apere cu ferocitate tărâmul stăpânului său. Garda întunecată este compusă din războinici puternici și vrăjmași neîndurători, instruiți să-și apere stăpânul cu orice preț. Când îl zăresc pe Sir Arthur, acești războinici se aruncă asupra lui cu săbiile lor în mâini, emanând o forță malefică care îl îngenunchează pe cavaler. Cu toate acestea, spiritual lui este neînfricat, iar determinarea sa este mai

puternică decât orice lovitură primită. Luptele sunt intense și pline de suspans, iar jucătorul trebuie să fie pregătit pentru orice. Fiecare mișcare și fiecare lovitură sunt cruciale în această bătălie împotriva gardienilor întunecați ai lordului. Obiectivul jucătorului în acest nivel este clar: să înfrunte garda întunecată pentru a pune capăt tiraniei și a restabili pacea în regatul său.

Proiectarea interfeței cu utilizatorul

Jocul prezintă mai multe interfețe cu jucătorul, acesta având mai multe stări: meniu, opțiuni, joc pierdut, pauză, nivel complet și jocul în desfășurare.

Meniul principal oferă opțiuni pentru începerea jocului, accesarea opțiunilor sau ieșirea din joc. Opțiunile permit jucătorului să regleze sunetul melodiei de fundal sau efectele animațiilor jucătorului. Joc pierdut apare când cavalerul este învins sau cade în capcană. De aic jucătorul poate reveni la meniul principal sau să reia nivelul. Pauza se activează la apăsarea tastei backspace și permite reglarea sunetului, întoarcerea la meniu, reluarea jocului sau resetarea nivelului curent. Nivel complet permit la fel întoarcerea la meniu și trecerea la următorul nivel. Interfața principală a jocului, în care jucătorul își controlează personajul și interacționează cu lumea jocului. Afișează informații precum viața jucătorului și bara de power attack.

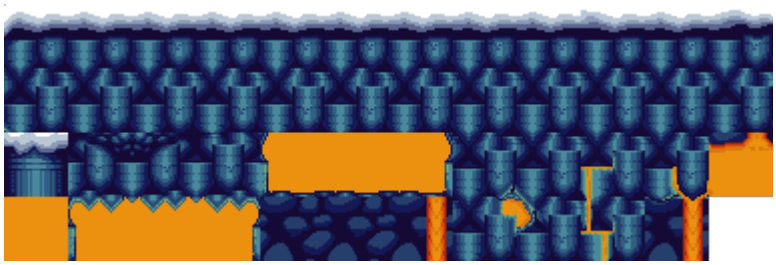


Game sprites

- Sir Arthur

- Chest

- Tile sets
  1. Nivel 1

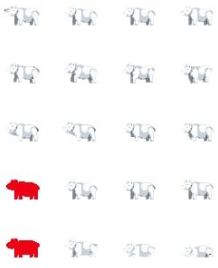


  2. Nivel 2

  3. Nivel 3

- Poțiuni

- Urs polar: MaxHealth = 10, EnemyDmg = 10



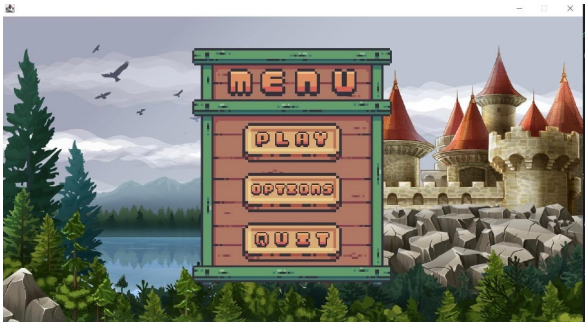
- Fantomă: MaxHealth = 18, EnemyDmg = 20

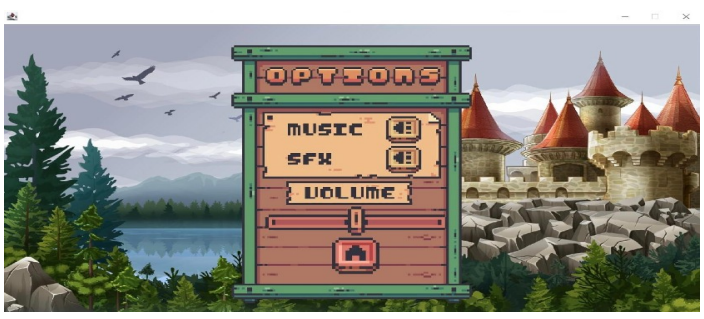
- Luptător: MaxHealth = 32, EnemyDmg = 30

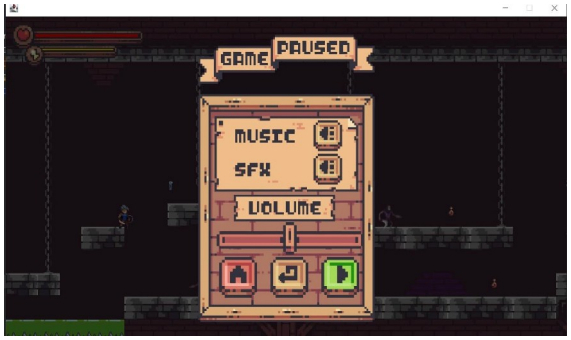
Metoda de victorie și de înfrângere

Jucătorul trebuie să parcurgă trei niveluri diferite și să omoare toți inamicii din fiecare nivel pentru a avansa la următorul. În timpul acestui proces, trebuie să evite capcanele și să-și păstreze viața, deoarece pierderea tuturor punctelor de viață duce la moartea cavalerului. Reluarea jocului se face de la nivelul la care a pierdut, resetându-se viața jucătorului la valoarea maximă inițială.

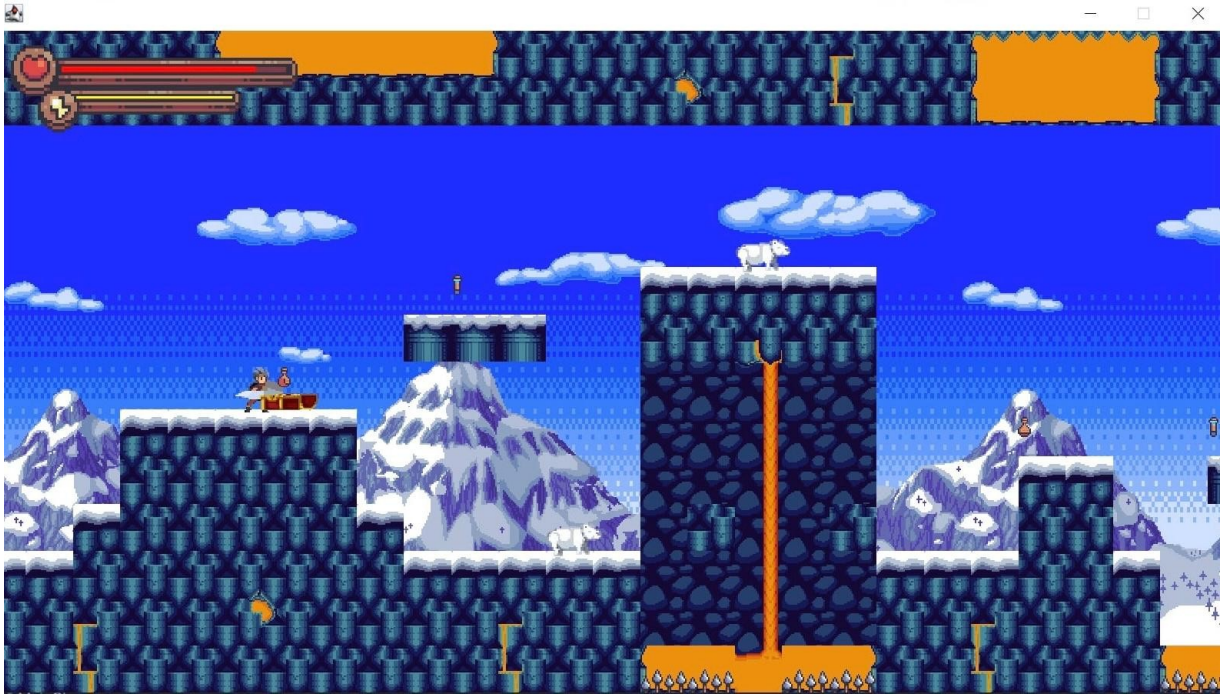
Screenshot-uri cu diferite ipostaze din joc

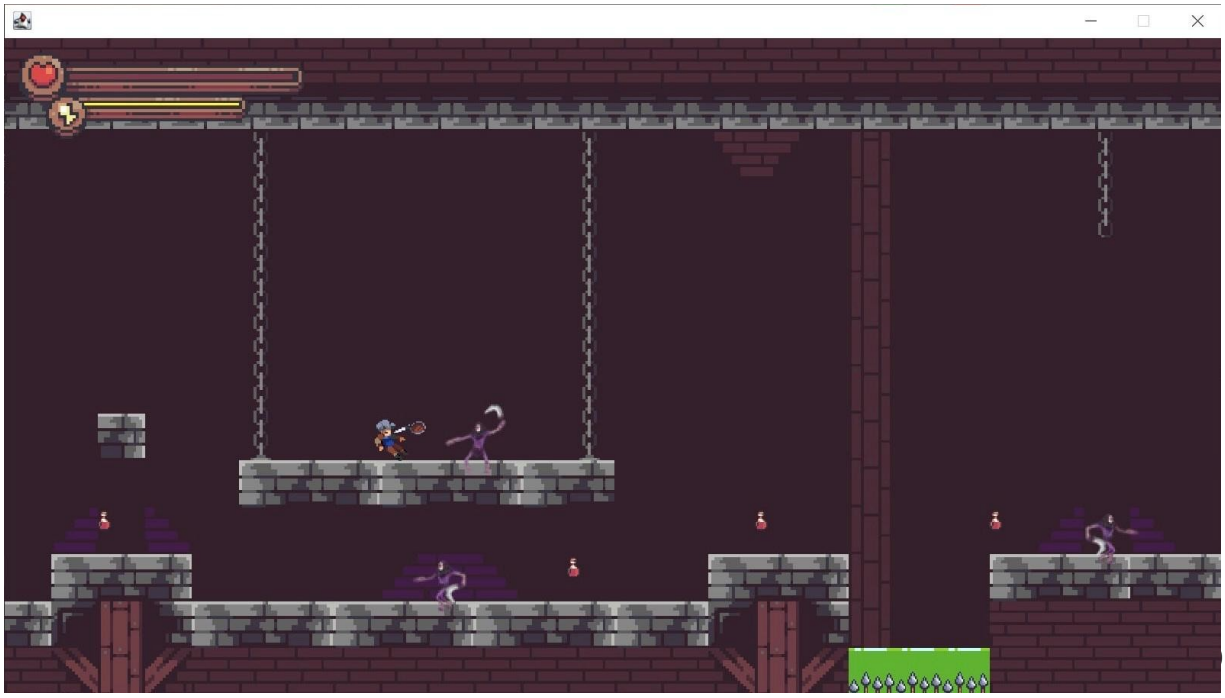





GAME PAUSED
MUSIC
SFX
VOLUME


LEVEL COMPLETED





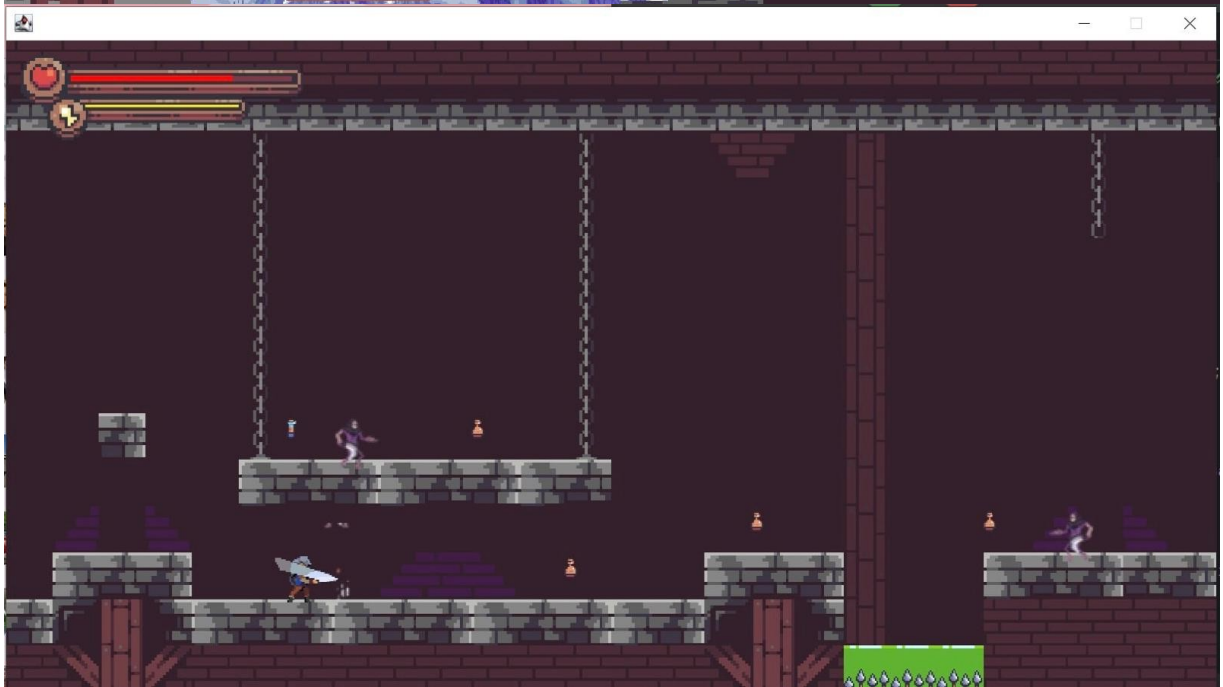
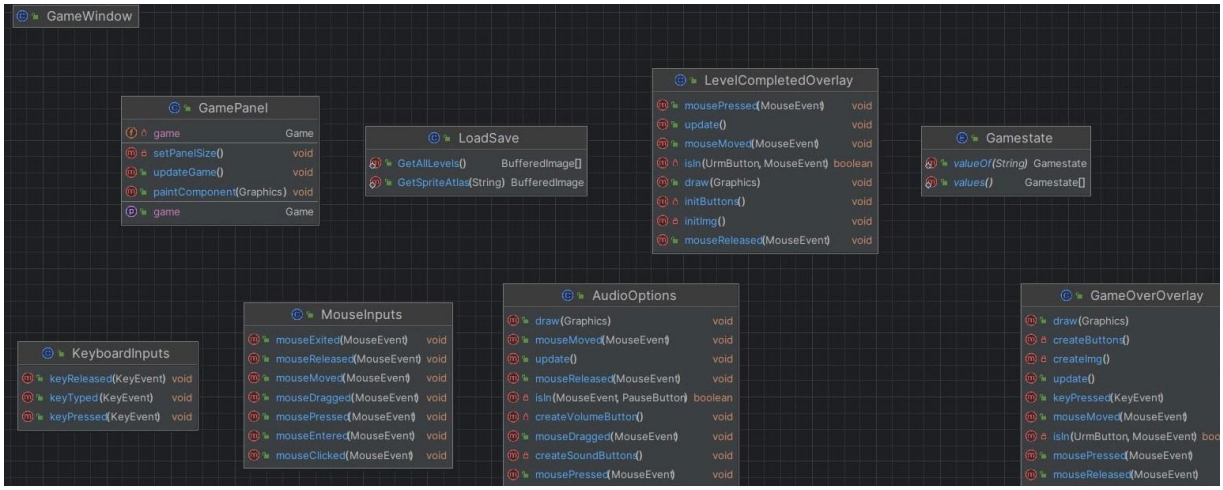
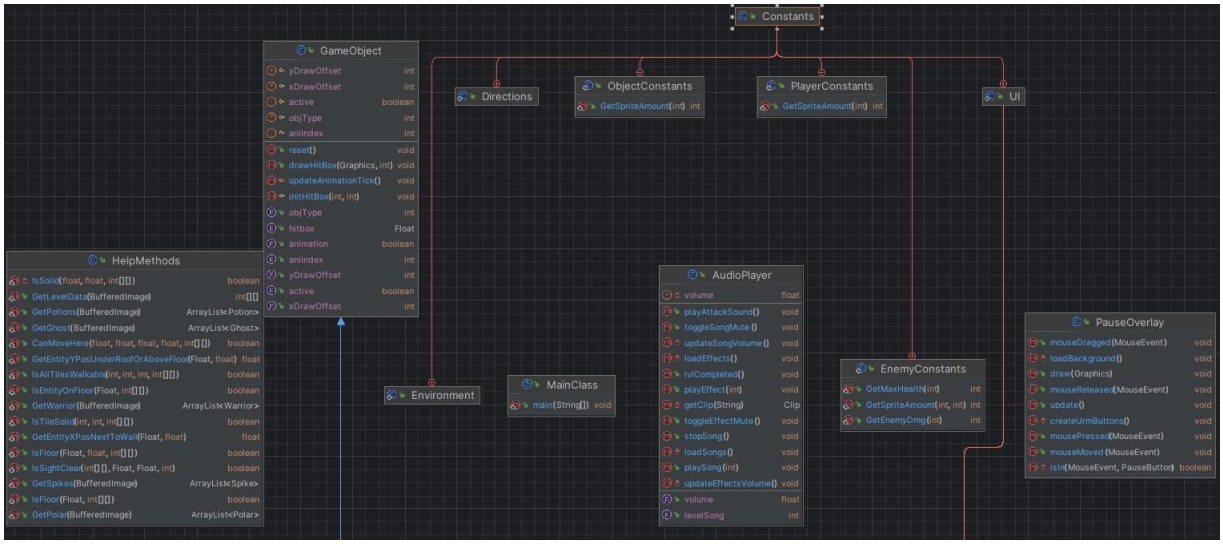
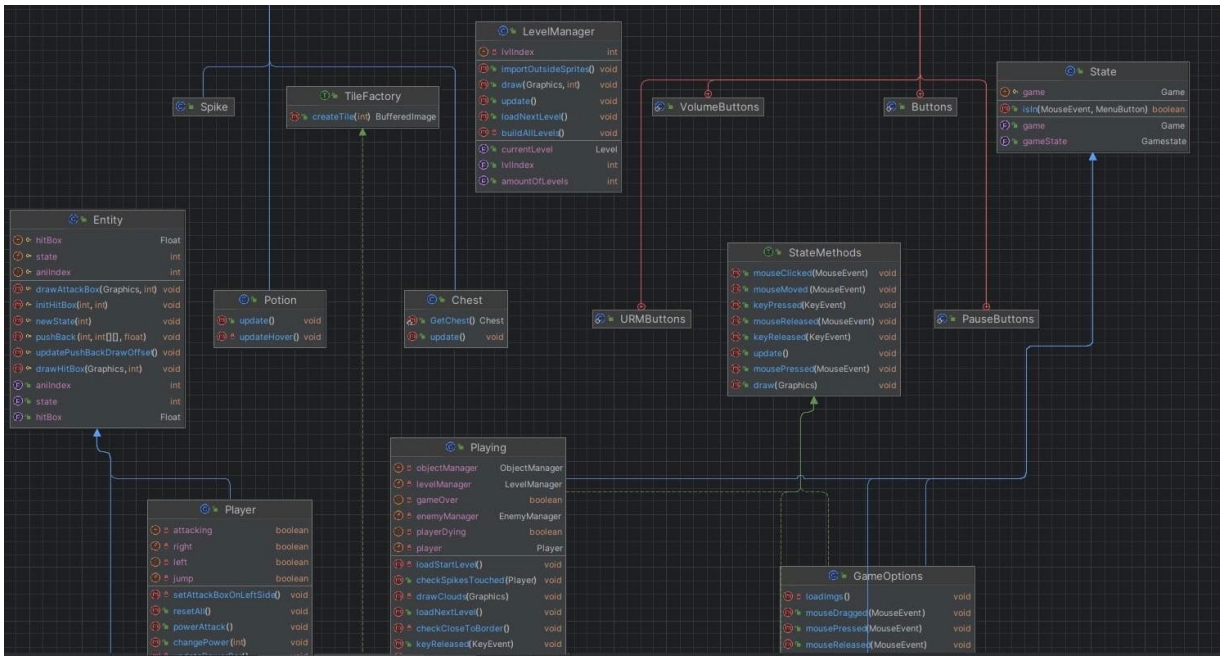
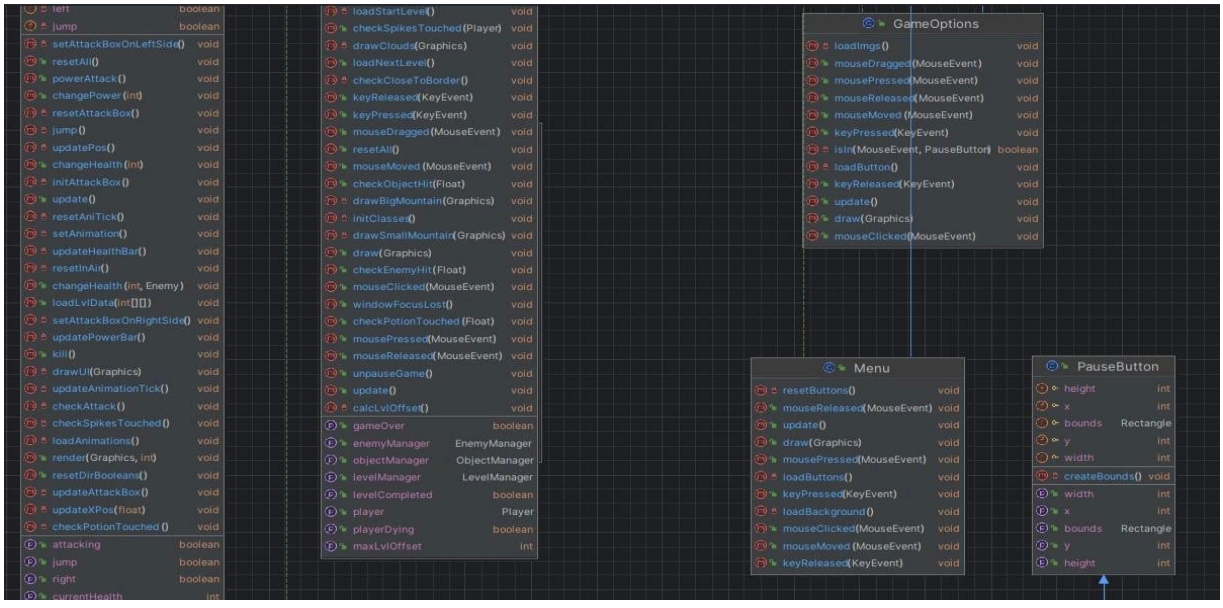


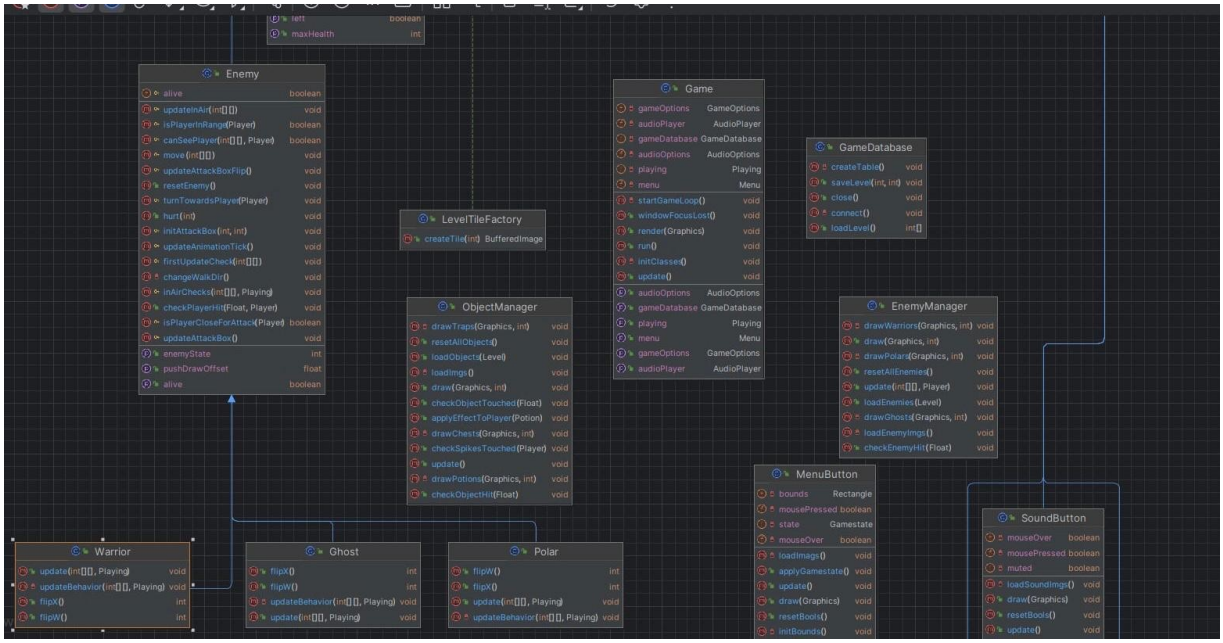
Diagrama UML

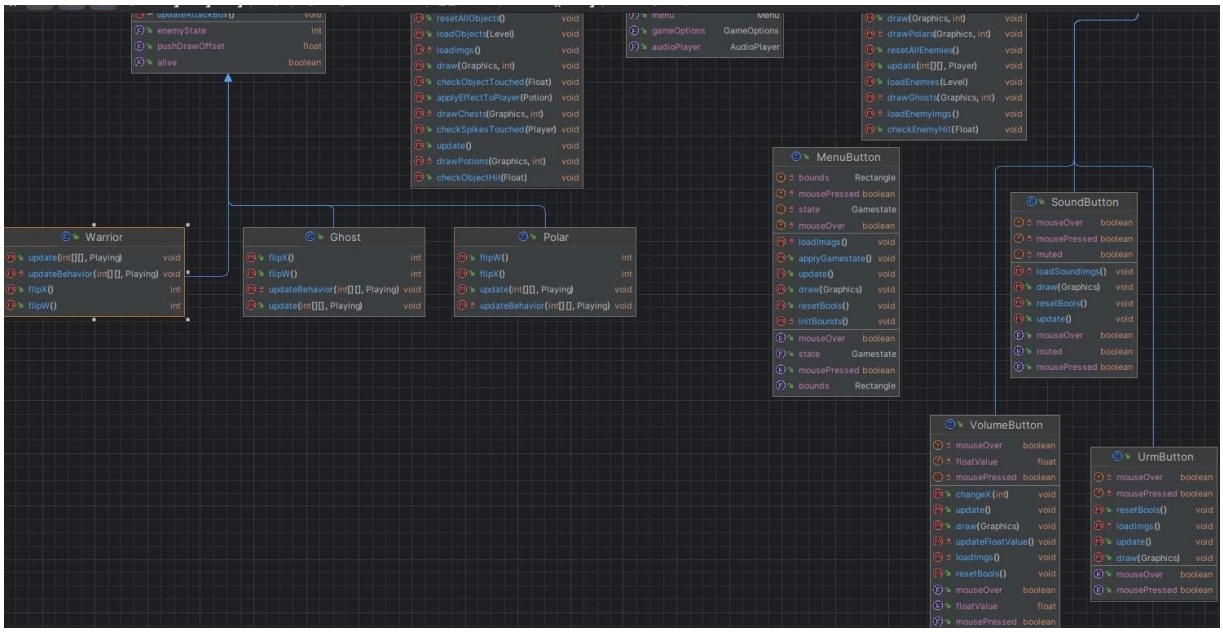


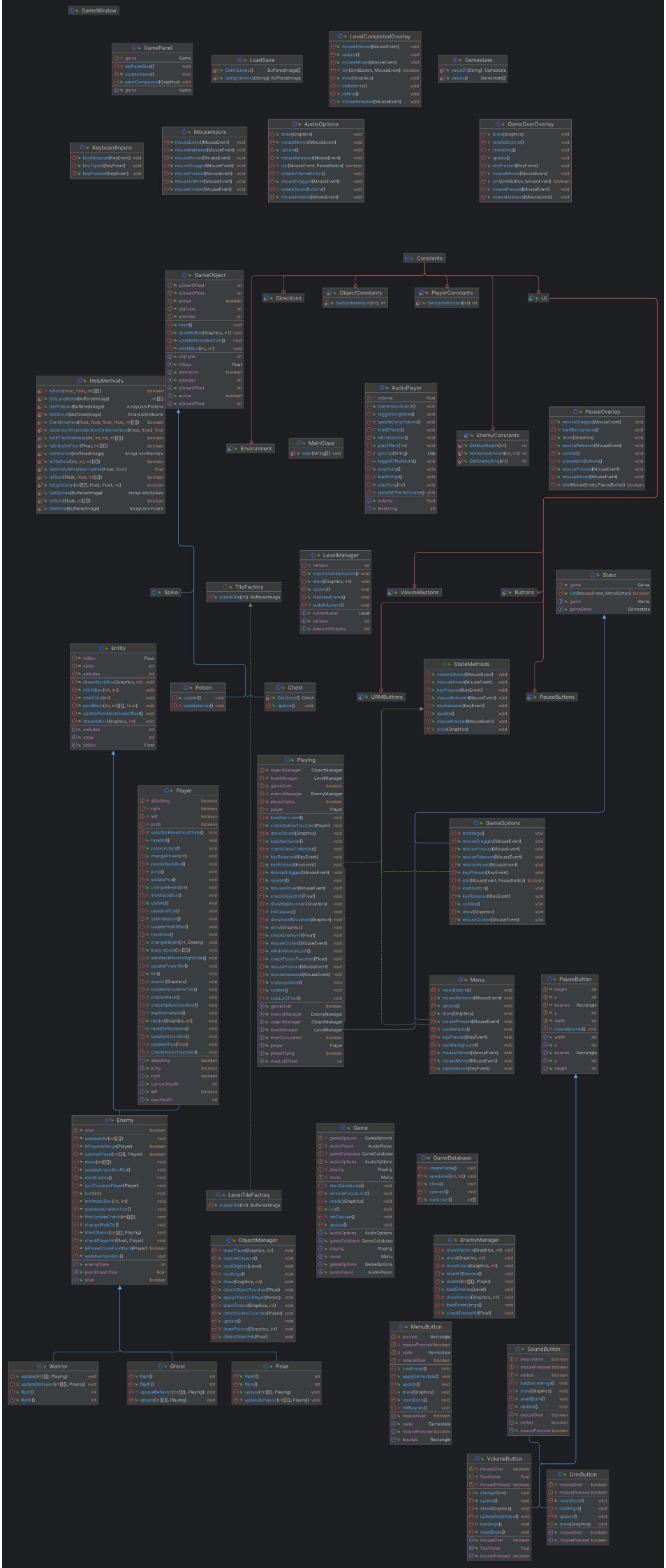

Constants
GameObject
yDrawOffset int
xDrawOffset int
active boolean
objType int
aniIndex int
reset() void
drawHitBox(Graphics, int) void
updateAnimationTick() void
initHitBox(int, int) void
objType int
hitbox Float
animation boolean
aniIndex int
yDrawOffset int
active boolean
xDrawOffset int
Directions
ObjectConstants
GetSpriteAmount(int) int
PlayerConstants
GetSpriteAmount(int) int
UI
HelpMethods
IsSolid(float, float, int[][]) boolean
GetLevelData(BufferedImage) int[][]
GetPotions(BufferedImage) ArrayList<Potion>
GetGhost(BufferedImage) ArrayList<Ghost>
CanMoveHere(float, float, float, float, int[][]) boolean
GetEntityYPosUnderRoofOrAboveFloor(Float, float) float
IsAllTilesWalkable(int, int, int, int[][]) boolean
IsEntityOnFloor(Float, int[][]) boolean
GetWarrior(BufferedImage) ArrayList<Warrior>
IsTileSolid(int, int, int[][]) boolean
GetEntityXPosNextToWall(Float, float) float
IsFloor(Float, float, int[][]) boolean
IsSightClear(int[][], Float, Float, int) boolean
GetSpikes(BufferedImage) ArrayList<Spike>
IsFloor(Float, int[][]) boolean
GetPolar(BufferedImage) ArrayList<Polar>
Environment
MainClass
main(String[]) void
AudioPlayer
volume float
playAttackSound() void
toggleSongMute() void
updateSongVolume() void
loadEffects() void
lvlCompleted() void
playEffect(int) void
getClip(String) Clip
toggleEffectMute() void
stopSong() void
loadSongs() void
playSong(int) void
updateEffectsVolume() void
volume float
levelSong int
EnemyConstants
GetMaxHealth(int) int
GetSpriteAmount(int, int) int
GetEnemyDmg(int) int
PauseOverlay
mouseDragged(MouseEvent) void
loadBackground() void
draw(Graphics) void
mouseReleased(MouseEvent) void
update() void
createUrmButtons() void
mousePressed(MouseEvent) void
mouseMoved(MouseEvent) void
isIn(MouseEvent, PauseButton) boolean


LevelManager
lvlIndex int
importOutsideSprites() void
draw(Graphics, int) void
update() void
loadNextLevel() void
buildAllLevels() void
currentLevel Level
lvlIndex int
amountOfLevels int
Spike
TileFactory
createTile(int) BufferedImage
VolumeButtons
Buttons
State
game Game
isIn(MouseEvent, MenuButton) boolean
game Game
gameState Gamestate
Entity
hitBox Float
state int
aniIndex int
drawAttackBox(Graphics, int) void
initHitBox(int, int) void
newState(int) void
pushBack(int, int[][], float) void
updatePushBackDrawOffset() void
drawHitBox(Graphics, int) void
aniIndex int
state int
hitBox Float
Potion
update() void
updateHover() void
Chest
GetChest() Chest
update() void
URMButtons
StateMethods
mouseClicked(MouseEvent) void
mouseMoved(MouseEvent) void
keyPressed(KeyEvent) void
mouseReleased(MouseEvent) void
keyReleased(KeyEvent) void
update() void
mousePressed(MouseEvent) void
draw(Graphics) void
PauseButtons
Player
attacking boolean
right boolean
left boolean
jump boolean
setAttackBoxOnLeftSide() void
resetAll() void
powerAttack() void
changePower(int) void
Playing
objectManager ObjectManager
levelManager LevelManager
gameOver boolean
enemyManager EnemyManager
playerDying boolean
player Player
loadStartLevel() void
checkSpikesTouched(Player) void
drawClouds(Graphics) void
loadNextLevel() void
checkCloseToBorder() void
keyReleased(KeyEvent) void
GameOptions
loadImgs() void
mouseDragged(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void


left boolean
jump boolean
setAttackBoxOnLeftSide() void
resetAll() void
powerAttack() void
changePower(int) void
resetAttackBox() void
jump() void
updatePos() void
changeHealth(int) void
initAttackBox() void
update() void
resetAniTick() void
setAnimation() void
updateHealthBar() void
resetInAir() void
changeHealth(int, Enemy) void
loadLvlData(int[][]) void
setAttackBoxOnRightSide() void
updatePowerBar() void
kill() void
drawUI(Graphics) void
updateAnimationTick() void
checkAttack() void
checkSpikesTouched() void
loadAnimations() void
render(Graphics, int) void
resetDirBooleans() void
updateAttackBox() void
updateXPos(float) void
checkPotionTouched() void
attacking boolean
jump boolean
right boolean
currentHealth int
loadStartLevel() void
checkSpikesTouched(Player) void
drawClouds(Graphics) void
loadNextLevel() void
checkCloseToBorder() void
keyReleased(KeyEvent) void
keyPressed(KeyEvent) void
mouseDragged(MouseEvent) void
resetAll() void
mouseMoved(MouseEvent) void
checkObjectHit(Float) void
drawBigMountain(Graphics) void
initClasses() void
drawSmallMountain(Graphics) void
draw(Graphics) void
checkEnemyHit(Float) void
mouseClicked(MouseEvent) void
windowFocusLost() void
checkPotionTouched(Float) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
unpauseGame() void
update() void
calcLvlOffset() void
gameOver boolean
enemyManager EnemyManager
objectManager ObjectManager
levelManager LevelManager
levelCompleted boolean
player Player
playerDying boolean
maxLvlOffset int
GameOptions
loadImgs() void
mouseDragged(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
mouseMoved(MouseEvent) void
keyPressed(KeyEvent) void
isIn(MouseEvent, PauseButton) boolean
loadButton() void
keyReleased(KeyEvent) void
update() void
draw(Graphics) void
mouseClicked(MouseEvent) void
Menu
resetButtons() void
mouseReleased(MouseEvent) void
update() void
draw(Graphics) void
mousePressed(MouseEvent) void
loadButtons() void
keyPressed(KeyEvent) void
loadBackground() void
mouseClicked(MouseEvent) void
mouseMoved(MouseEvent) void
keyReleased(KeyEvent) void
PauseButton
height int
x int
bounds Rectangle
y int
width int
createBounds() void
width int
x int
bounds Rectangle
y int
height int


left boolean
maxHealth int
Enemy
alive boolean
updateInAir(int[][]) void
isPlayerInRange(Player) boolean
canSeePlayer(int[][], Player) boolean
move(int[][]) void
updateAttackBoxFlip() void
resetEnemy() void
turnTowardsPlayer(Player) void
hurt(int) void
initAttackBox(int, int) void
updateAnimationTick() void
firstUpdateCheck(int[][]) void
changeWalkDir() void
inAirChecks(int[][], Playing) void
checkPlayerHit(Float, Player) void
isPlayerCloseForAttack(Player) boolean
updateAttackBox() void
enemyState int
pushDrawOffset float
alive boolean
LevelTileFactory
createTile(int) BufferedImage
Game
gameOptions GameOptions
audioPlayer AudioPlayer
gameDatabase GameDatabase
audioOptions AudioOptions
playing Playing
menu Menu
startGameLoop() void
windowFocusLost() void
render(Graphics) void
run() void
initClasses() void
update() void
audioOptions AudioOptions
gameDatabase GameDatabase
playing Playing
menu Menu
gameOptions GameOptions
audioPlayer AudioPlayer
GameDatabase
createTable() void
saveLevel(int, int) void
close() void
connect() void
loadLevel() int[]
ObjectManager
drawTraps(Graphics, int) void
resetAllObjects() void
loadObjects(Level) void
loadImgs() void
draw(Graphics, int) void
checkObjectTouched(Float) void
applyEffectToPlayer(Potion) void
drawChests(Graphics, int) void
checkSpikesTouched(Player) void
update() void
drawPotions(Graphics, int) void
checkObjectHit(Float) void
EnemyManager
drawWarriors(Graphics, int) void
draw(Graphics, int) void
drawPolars(Graphics, int) void
resetAllEnemies() void
update(int[][], Player) void
loadEnemies(Level) void
drawGhosts(Graphics, int) void
loadEnemyImgs() void
checkEnemyHit(Float) void
MenuButton
bounds Rectangle
mousePressed boolean
state Gamestate
mouseOver boolean
loadImages() void
applyGamestate() void
update() void
draw(Graphics) void
resetBools() void
initBounds() void
SoundButton
mouseOver boolean
mousePressed boolean
muted boolean
loadSoundImgs() void
draw(Graphics) void
resetBools() void
update() void
Warrior
update(int[][], Playing) void
updateBehavior(int[][], Playing) void
flipX() int
flipW() int
Ghost
flipX() int
flipW() int
updateBehavior(int[][], Playing) void
update(int[][], Playing) void
Polar
flipW() int
flipX() int
update(int[][], Playing) void
updateBehavior(int[][], Playing) void


VolumeButton
mouseOver boolean
floatValue float
mousePressed boolean
changeX(int) void
update() void
draw(Graphics) void
updateFloatValue() void
loadImgs() void
resetBools() void
UrmButton
mouseOver boolean
mousePressed boolean
resetBools() void
loadImgs() void
update() void
draw(Graphics) void


GameWindow
GamePanel
LoadSave
LevelCompletedOverlay
Gamestate
MouseInputs
AudioOptions
GameOverOverlay
KeyboardInputs
Constants
GameObject
Directions
ObjectConstants
PlayerConstants
UI
HelpMethods
AudioPlayer
PauseOverlay
Environment
MainClass
EnemyConstants
LevelManager
Spike
TileFactory
VolumeButtons
Buttons
State
Entity
StateMethods
Potion
Chest
URMButtons
PauseButtons
Playing
Player
GameOptions
Menu
PauseButton
Enemy
Game
GameDatabase
LevelTileFactory
ObjectManager
EnemyManager
MenuButton
SoundButton
Warrior
Ghost
Polar
VolumeButton
UrmButton

Pachetele create în proiect sunt: audio, entities, gamestates, inputs, levels, main, objects, ui, utils. Fiecare pachet conține clase cu diferite utilități:

- **entities**: În acest pachet, clasa abstractă Entity este clasa de bază pentru toate entitățile din joc, inclusiv jucătorul și inamicii. Acesta conține proprietăți și metode comune pentru toate entitățile. Clasa Player extinde clasa Entity și implementează comportamente specifice pentru jucător. Clasa abstractă Enemy extinde Entity și servește drept bază pentru toate tipurile de inamici. Clasele Polar, Ghost și Warrior extind Enemy și implementează comportamente specifice inamicilor. EnemyManager gestionează toți inamicii din joc, inclusiv crearea, actualizarea și renderizarea acestora.

- **gamestates**: Interfața StateMethods definește metodele esențiale pentru gestionarea diferitelor stări ale jocului, incluzând actualizarea, desenarea și gestionarea evenimentelor de mouse și tastatură. Clasa State servește drept bază pentru toate stările specifice ale jocului și conține metode utile pentru gestionarea jocului și a stării acestuia. Clasa enum Gamestate definește stările posibile ale jocului: *PLAYING*, *MENU*, *OPTIONS*, *QUIT.* Clasa Menu extinde State și implementează StateMethods, gestionând logica meniului principal, inclusiv încărcarea butoanelor și a imaginii de fundal, precum și răspunsul la evenimentele de mouse. Clasa Playing gestionează stare jocului în timpul efectiv. Funcționalități principale: inițializarea obiectelor necesare (LevelManager, EnemyManager, ObjectManager) și a elementelor de interfață pentru suprapunerea ecranului(PauseOverlay, GameOverOverlay, LevelCompletedOverlay), desenarea elementelor grafice pe ecran, actualizarea continuă a stării jocului, gestionarea interacțiunilor cu mouse-ul și tastatură pentru controlul jucătorului și a interfeței utilizatorului, verificarea coliziunilor și a actiunilor jucătorului cu obiectele și inamicii din nivel. GameOptions este responsabilă de gestionarea opțiunilor de joc, inclusiv setările audio și navigarea către meniul principal.

- **inputs**: KeyboardInputs și MouseInputs asigură că evenimentele de la tastatură, respectiv de la mouse sunt corect direcționate și gestionate în funcție de starea curentă a jocului.

- **levels**: Interfața TileFactory definește o metodă pentru a crea imagini pentru diferite tipuri de dale într-un nivel. Clasa LevelTileFactory implementează interfața, este responsabilă pentru crearea de imagini pentru dalele dintr-un nivel și construiește un set de imagini pentru dalele dintr-un nivel pe baza unei imagini de atlas dată. Level este responsabilă de gestionarea și manipularea datelor asociate unui nivel din joc. LevelManager este esențială pentru gestionarea și schimbul nivelurilor în timpul jocului, precum și pentru desenarea și actualizarea lor pe ecran.

- **main**: Clasa Game inițializează toate componentele jocului și începe bucla principală a jocului într-un thread separat. Implementează Runnable pentru a putea rula bucla principală a jocului într-un thread separat. GamePanel reține și afișează jocul pe un Jpanel, inițializează și gestionează intrările de la tastatură și mouse. GameWindow creează fereastra jocului și o atașează la GamePanel. MainClass conține metoda main care creează o instanță a clasei Game, inițializând astfel întregul joc.

- **objects**: GameObject reprezintă obiectele generice din joc, precum poțiuni, cufăr și obiecte periculoase. Gestionează aspecte comune ale acestor obiecte, cum ar fi actualizarea animației și dezactivarea. ObjectManager se ocupă de gestionarea și actualizarea obiectelor din joc, verifică coliziunile dintre jucător și diverse obiecte, desenează obiectele pe ecran și le actualizează pozițiile și stările. Clasa Chest reprezintă cufărul din joc, care poate fi deschis de jucător pentru a elibera poțiuni. Gestionează animația deschiderii cufărului și actualizează starea obiectului în funcție de acțiunile jucătorului. Potion reprezintă poțiunile din joc, care pot fi colectate de către jucător pentru a-și îmbunătății viața și power attack. Realizează o animație de „plutire” pentru poțiuni și le actualizează pozițiile în timpul jocului. Clasa Spike reprezintă capcanele din joc, care pot răni jucătorul dacă intră în contact cu ele.

- **ui**: conține trei clase care reprezintă suprapunerile pentru interfețele utilizatorului care apar în momente specifice ale jocului. GameOverOverlay afișează un ecran care apare atunci când jucătorul pierde jocul, conține două butoane: unul pentru areveni la meniul principal și altul pentru a începe un nou joc. LevelCompletedOverlay afișează un ecran care apare atunci când jucătorul completează un nivel. De asemenea, conține două butoane: unul pentru areveni la meniul principal și altul pentru a trece la nivelul următor. PauseOverlay implementează un ecran de pauză, oferind jucătorului opțiuni precum revenirea la meniul principal, reluarea nivelului sau reluarea jocului. Toate trei inițializează imaginile de fundal și butoanele, gestionează butoane și opțiunile audio asociate, actualizează starea butoanelor și opțiunile audio în funcție de interacțiunea jucătorului, desenează fundalul și butoanele pe ecran, reflectând starea lor curentă(normal, pressed, hover). De asemenea, monitorizează evenimentele de mouse, cum ar fi apăsările, eliberările și mișcările.
  Clasa PauseButton este o clasă de bază pentru butoanele utilizate în meniul de pauză al jocului. Clasele UrmButton, SoundButton, VolumeButton extind PauseButton și sunt utilizate pentru a crea butoane speciale: revenire la meniu principal, reluarea nivelului, resetarea nivelului, butoane de sunet și un controler de volum. Clasa MenuButton este utilizată pentr a crea și gestiona butoanele din meniul principal al jocului. Aceasta clasează butoanele în funcție de starea jocului. Clasa AudioOptions este utilizată pentru gestionarea

opțiunilor adio din joc, cum ar fi controlul volumului și activarea/dezactivarea suntelor de fundal și a efectelor secundare.

- **utils**: HelpMethods oferă diverse metode de utilitate pentru manipularea datelor nivelului, verificarea coliziunilor și gestionarea entităților în joc. Constants conține clase inelare statice care conțin valori constante care vor fi utilizate în joc: ObjectConstants, EnemyConstants, Environment, UI, Directions și PlayerConstants. LoadSave este folosită pentru a încărca sprite-uri, background-uri, butoane și niveluri din fișiere, declarate ca constante de tip String. GameDatabase se ocupă de interacțiunea cu o bază de date SQLite pentru a salva și încărca date legate de nivelul jocului și starea de sănătate a jucătorului.

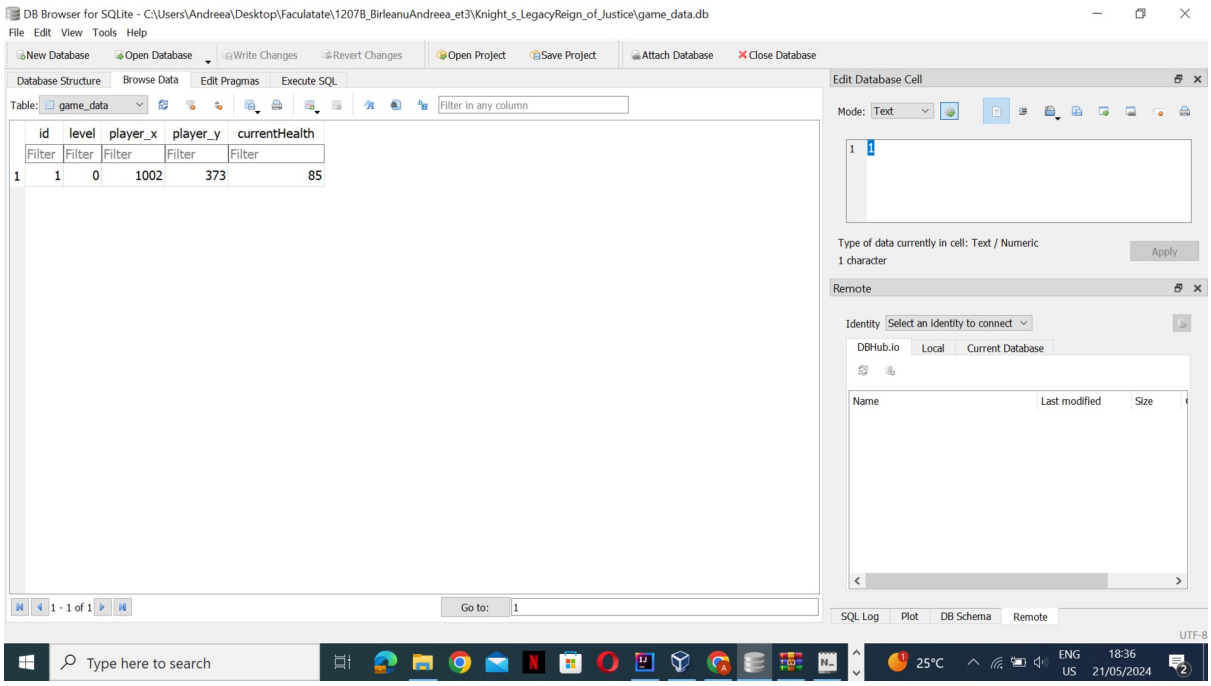
- **audio**: conține clasa AudioPlayer care se ocupă cu gestionarea și redarea fișierelor audio într-un joc. Aceasta include atât muzica de fundal pentru diferite nivele, cât și efectele sonore asociate cu diverse acțiuni ale jucătorului.

Structura bazei de date

Baza de date constă într-un singur tabel numit levels care are două coloane:

1. level: Un câmp de tip INTEGER care stochează nivelul curent al jocului.
2. player_x: Un câmp de tip INTEGER care reține coordonata hitbox-ului jucătorului pe axa OX.
3. player_y: Un câmp de tip INTEGER care reține coordonata hitbox-ului jucătorului pe axa OY.
4. current_health: Un câmp de tip INTEGER care reține starea de sănătate curentă a jucătorului.

Această structură de bază este concepută pentru a stoca și gestiona informațiile esențiale legate de progresul jocului și starea jucătorului.



Load and Save

Încărcarea din baza de date are loc în momentul în care utilizatorul apasă tasta L în timpul jocului. Se va încărca nivelul al cărui index se află în baza de date, viața curentă ia valoarea din bza de date, iar jucătorul va apărea pe ecran la coordonatele salvate în baza de date.

Salvarea în baza de date se face în momentul în care utilizatorul apasă tasta S în timpul jocului, pentru a asigura că se salvează corect progresul jocului. Se vor salva indexul nivelului curent, coordonatele x și y ale hitbox-ului jucătorului și viața curentă a acestuia.

Șabloane de proiectare

1. Singleton: Clasa Chest implementează un model Singleton, șablon creațional.
    - Constructorul clasei este privat, astfel încât să nu poată fi creată o instanță nouă din exteriorul clasei.
    - Există o metodă statică GetChest() care verifică dacă există deja o instanță a clasei Chest. Dacă există, acea instanță este returnată. Dacă nu există, este creată o instanță nouă și returnată.

- Variabila chest este declarată static și privată, astfel încât să nu poată fi accesată din afara clasei.
- Singleton-ul asigură că în întregul program există întotdeauna o singură instanță a clasei Chest, evitând astfel crearea accidentală a mai multor instanțe și asigurându-se că toate referințele la obiect sunt îndreptate către aceeași instanță.

2. Factory: Folosit la crearea dalelor pentru hartă, șablon creațional.Acesta permite obținerea unui obiect fără a fi necesară specificarea explicită a clasei exacte a obiectului care va fi creat. În schimb, este utilizată o interfață comună sau o clasă de bază pentru a declara metodele de creare a obiectelor, iar clasele derivate sau implementările concrete ale acestei interfețe sunt utilizate pentru a crea obiectele reale.

- Interfața TileFactory definește o metodă pentru a crea imagini pentru dalele din nivel. Această interfață specifică o abstracțiune comună pentru toate fabricile de dale.
- Clasa LevelTileFactory mplementează interfața și furnizează o implementare specifică pentru crearea de imagini pentru dalele din nivel. Această clasă conține logica pentru a extrage sub-imagini dintr-o imagine mai mare de atlas și pentru a le furniza sub formă de dale individuale.
- Clasa LevelManager este un client al fabricii. Ea folosește fabrica (LevelFactory) pentru a crea imaginile necesare pentru a desena nivelul pe ecran. Prin intermediul metodei createTile(int index) furnizată de fabrică, poate obține imaginile individuale pentru dale și le poate utiliza pentru desenare.

3. State: Acest șablonul de proiectare este un model comportamental care permite unui obiect să-și schimbe comportamentul în funcție de starea internă pe care o are. Acesta permite obiectului să varieze comportamentul său în funcție de starea sa internă, fără a schimba explicit clasa obiectului.

În joc, se utilizează șablonul State pentru a gestiona diferitele stări ale jocului și a coordona funcționalitățile acestuia în timpul rulări.

Interfața StateMethods: Această interfață definește metodele necesare pentru manipularea grafică și a evenimentelor de intrare în cadrul diferitelor stări ale jocului. Metodele includ update() pentru actualizarea stării, draw(Graphics g) pentru desenarea elementelor grafice, precum și metode pentru gestionarea evenimentelor de mouse și tastatură.

Clasa abstractă State: Această clasă servește drept bază pentru toate stările specifice ale jocului. Ea conține o referință la obiectul Game, care permite accesul la funcționalitățile jocului. De asemenea, conține metode utile pentru gestionarea

jocului și a stării acestuia, cum ar fi metoda setGameState() pentru schimbarea stării jocului.

Enumerarea Gamestate: Această enumerare definește stările posibile ale jocului, precum PLAYING, MENU, OPTIONS și QUIT.

Clasele specifice ale stărilor: Aceste clase implementează interfața și definesc comportamentul specific pentru fiecare stare a jocului.

Implementarea coliziunilor

- Dale

1. Detectarea coliziunilor:

   Pentru a detecta coliziunile, se utilizează mai multe metode, cum ar fi CanMoveHere, care verifică dacă entitatea poate să se miște într-o anumită poziție fără să intre în coliziune cu alte obiecte din joc. Aceste metode verifică fiecare colț al hitbox-ului entității pentru a detecta dacă ar intra în coliziune cu obiecte solide din mediu.

2. Răspunsul la coliziuni:

   După ce o coliziune este detectată, se iau diferite acțiuni în funcție de tipul coliziunii. De exemplu, dacă o entitate se ciocnește de un perete, poziția sa de pe axa X este ajustată astfel încât să se oprească lângă perete. Pentru coliziunile cu podeaua sau tavanul, poziția pe axa Y este ajustată pentru a asigura că entitatea se află pe podea sau sub un acoperiș.

3. Verificarea coliziunilor în timp real:

   Coliziunile sunt verificate în mod constant în timp real în timpul jocului. Acest lucru se realizează în metoda update() a entităților. În această metodă, se verifică și se răspunde la coliziuni pentru a asigura că entitățile se deplasează în mod corespunzător și nu trec prin obiectele din mediu.

- Obiecte

1. Capcane: Pentru fiecare obiect Spike, se verifică dacă hitbox-ul acestuia se intersectează cu hitbox-ul jucătorului prin metoda checkSpikesTouched(Player p). Dacă se detectează o coliziune, jucătorul este eliminat.
2. Poțiuni: Pentru fiecare obiect Potion activ, se verifică dacă hitbox-ul acestuia se intersectează cu hitbox-ul jucătorului prin metoda

checkObjectTouched(Rectangle2D.Float hitBox. Dacă se detectează o coliziune, se dezactivează potiunea și se aplică efectul acesteia asupra jucătorului prin apelul metodei applyEfectToPlayer(p).

3. Cufăr: Se verifică dacă cufărul este activ și nu este în timpul unei animații prin metoda checkObjectHit(Rectangle2D.Float attackBox). Se verifică dacă hitbox-ul cufărului se intersectează cu hitbox-ul atacului. Dacă se detectează o coliziune, se activează animația cufărului și se adaugă o nouă potiune în lista potions la o poziție calculată în funcție de hitbox-ul cufărului.

- Inamici

  Metoda checkPlayerHit(Rectangle2D.Float attackBox, Player player) verifică coliziunile între atacuri și jucător. Se verifică dacă hitbox-ul atacului se intersectează cu hitbox-ul jucătorului. Dacă se detectează o coliziune, se reduce sănătatea jucătorului cu o valoare prin apelul metodei changeHealth a jucătorului. Metoda checkEnemyHit(Rectangle2D.Float attackBox) verifică coliziunile între atacuri și inamici (warrior, ghost, polar) . Dacă se detectează o coliziune între hitbox-ul atacului și hitbox-ul unui războinic, se aplică o anumită cantitate de daune acestuia prin apelul metodei hurt a războinicului. Se efectuează același proces ca și în cazul războinicilor pentru fiecare tip de inamic (ghost și polar).

Deplasarea inamicilor

Mișcarea inamicilor este gestionată în următoarele etape:

1. Determinarea stării și a poziției: Inițial, se verifică dacă inamicul este pe sol sau în aer și care este starea sa curentă (IDLE, RUNNING, ATTACK, HIT). Aceste informații sunt esențiale pentru a decide cum trebuie să se miște inamicul în următoarea actualizare.

2. Detectarea jucătorului: Se verifică dacă inamicul poate vedea jucătorul și dacă acesta este la o distanță suficient de mică pentru a fi urmărit sau atacat.

3. Calcularea direcției de deplasare: În funcție de poziția jucătorului și de direcția în care se află acesta, inamicul calculează direcția în care trebuie să se deplaseze pentru a-l urmări sau a-l ataca.

4. Mișcarea în aer: Dacă inamicul este în aer, acesta actualizează poziția pe axa Y, luând în considerare gravitația și coliziunile cu obiectele din mediu, până când revine pe sol sau până când nu mai poate să se miște în sus.

5. Mișcarea pe sol: Atunci când inamicul este pe sol, acesta se deplasează pe axa X în direcția calculată anterior. Dacă întâlnește un obstacol, își schimbă direcția de mers.

În general, această logică asigură că inamicul urmărește și atacă jucătorul într-un mod credibil, evitând obstacolele și adaptându-se la mediul din jur.

Structurile de date importante

1. Jucător (Player):

   Hitbox: O zonă rectangulară care reprezintă coliziunea jucătorului cu obiectele din mediu.

   Stare (State): Starea actuală a jucătorului (IDLE, RUNNING, ATTACK, HIT, DEAD).

   Sănătate (Health): Cantitatea de sănătate a jucătorului.

   Putere (Power): Nivelul de putere sau energie al jucătorului.

2. Inamic (Enemy):

   Hitbox: Similar cu jucătorul, o zonă rectangulară care reprezintă coliziunea inamicului cu obiectele din mediu.

   Stare (State): Starea actuală a inamicului (IDLE, RUNNING, ATTACK, HIT, DEAD).

   Sănătate (Health): Cantitatea de sănătate a inamicului.

3. Nivel (Level):

   lvlData: Matricea care reprezintă harta nivelului, cu informații despre fiecare tile (solid sau trecător).

Formatul fișierelor de imagini pentru sprite-uri în joc este PNG. Sunt utilizate pentru personaje, obiecte și mediul de joc, interfețe grafice, hărți.

Arhitectura sistemului este structurată în jurul următoarelor componente principale:

1. Motorul jocului:

   Această componentă reprezintă nucleul jocului și gestionează toate aspectele sale fundamentale, cum ar fi logica jocului, grafica, sunetul, fizica și interacțiunea cu utilizatorul. Interacționează cu toate celelalte componente ale sistemului pentru a coordona funcționalitățile jocului.

2. Modul de procesare a coliziunilor:

Responsabil pentru gestionarea detecției și rezolvării coliziunilor dintre entități (cum ar fi jucătorul, inamicii și obiectele de mediu) în joc. nteracționează strâns cu motorul jocului pentru a integra corect logica coliziunilor în fluxul general al jocului.

3. Interfața Utilizator (User Interface):

   Această componentă gestionează interacțiunea dintre jucător și joc prin intermediul elementelor de interfață grafică, cum ar fi meniurile, butoanele. Interacționează cu motorul jocului pentru a reflecta corect starea și progresul jocului în interfața utilizator.

Resurse bibliografice

- Munte și fundal hartă: https://phi9009.itch.io/mountain-peak-background-pack
- Sat: https://anokolisa.itch.io/legacy-adventure-pack
- Castel: https://bakudas.itch.io/generic-dungeon-pack
- Cavaler: https://brullov.itch.io/generic-char-asset?download#google_vignette
- Cufăr: https://admurin.itch.io/free-chest-animations
- Poțiuni: https://github.com/KaarinGaming/PlatformerTutorial/blob/ep26/PlatformerTutorial/res/potions_sprites.png
- Capcane: https://github.com/KaarinGaming/PlatformerTutorial/blob/ep26/PlatformerTutorial/res/trap_atlas.png
- Urs polar: https://admurin.itch.io/top-down-mobs-bears
- Documentație: https://docs.oracle.com/javase/8/docs/api/
- Tutorial:https://www.youtube.com/watch?v=6_N8QZ47toY&list=PL4rzdwizLaxYmltJQRjq18a9gsSyEQQ-0
- Luptător:

https://clembod.itch.io/bringer-of-death-free?download#google_vignette

- Interfețe și butoane: https://github.com/KaarinGaming/PlatformerTutorial/tree/ep28_finale/PlatformerTutorial/res
- Fundal: https://www.artstation.com/artwork/xJv4WW