Blade of Time

LĂDUNCĂ IOAN-DARIAN
1212B

<u>Proiectare context</u>

Gen joc

Three Quarters View / RPG

Story

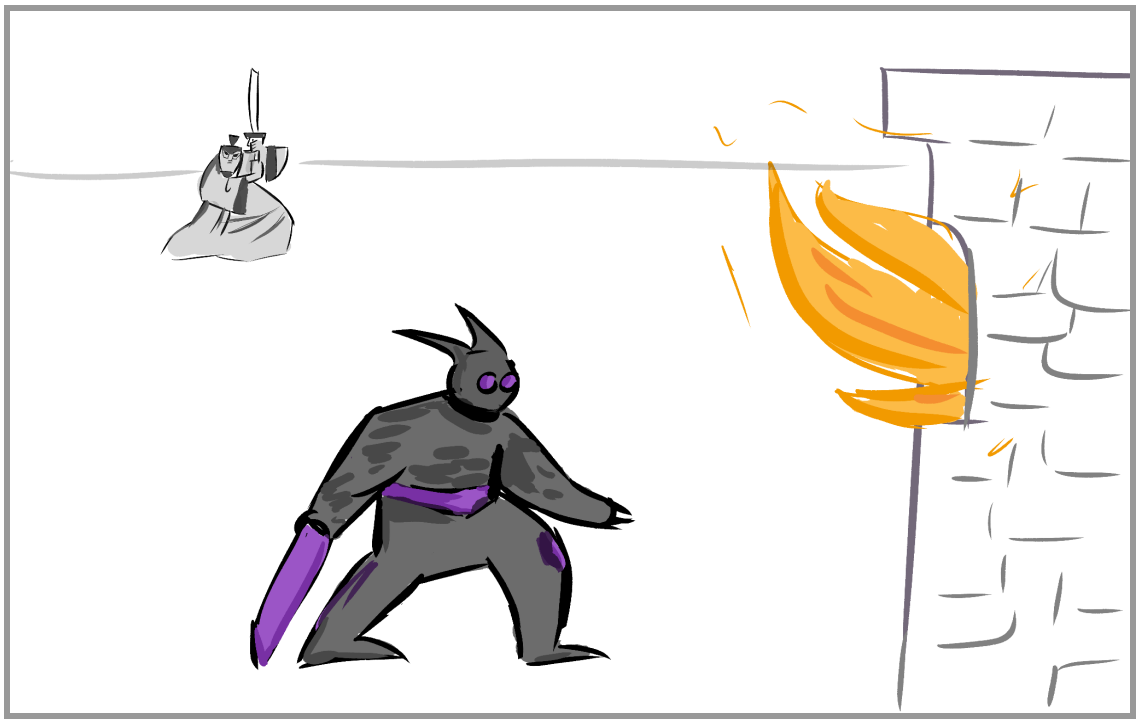
Într-un tărâm al legendelor antice și al războinicilor legați de onoare, a apărut un mare rău. Numele său era Akaza și nu lăsa în urma sa decât distrugere și moarte. Oamenii au strigat după un erou, dar nici împăratul însuși nu a putut învinge acest demon.

Disperat, împăratul s-a întors către zei, cerându-le o armă care să poată nimici demonul. Dar chiar și divinii nu puteau crea o sabie suficient de puternică pentru a-l ucide pe Akaza.

Atunci a venit rândul fiului împăratului, tânărul prinț samurai Rengoku. Cu sabia tatălui său în mână, el a înfruntat demonul cu curaj neclintit și cu o pricepere dincolo de anii săi. Pentru o clipă, părea că ar putea triumfa, dar Akaza era isteț și l-a prins pe prinț într-o capcană.

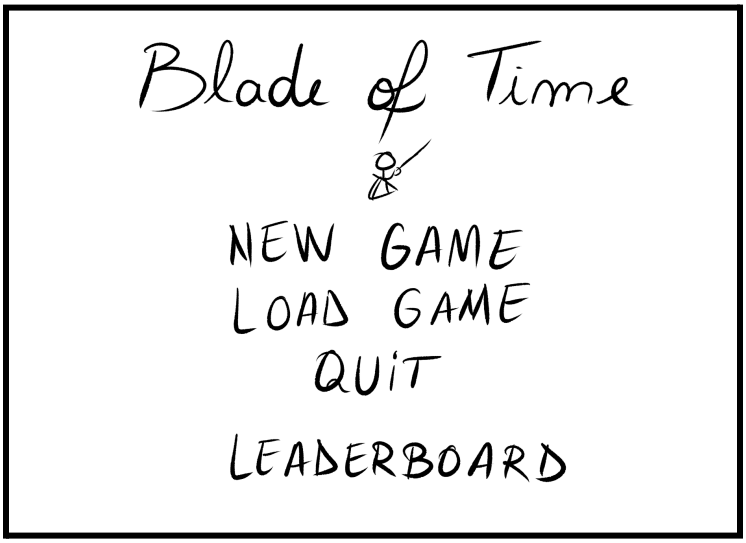
Și astfel, Rengoku s-a găsit transportat într-un viitor în care Akaza era un mare conducător, iar oamenii pe care îi iubea erau în pericol. Cu o inimă hotărâtă, el a plecat în căutarea elementelor lipsă ale sabiei și a unui mod de a călători înapoi în timp.

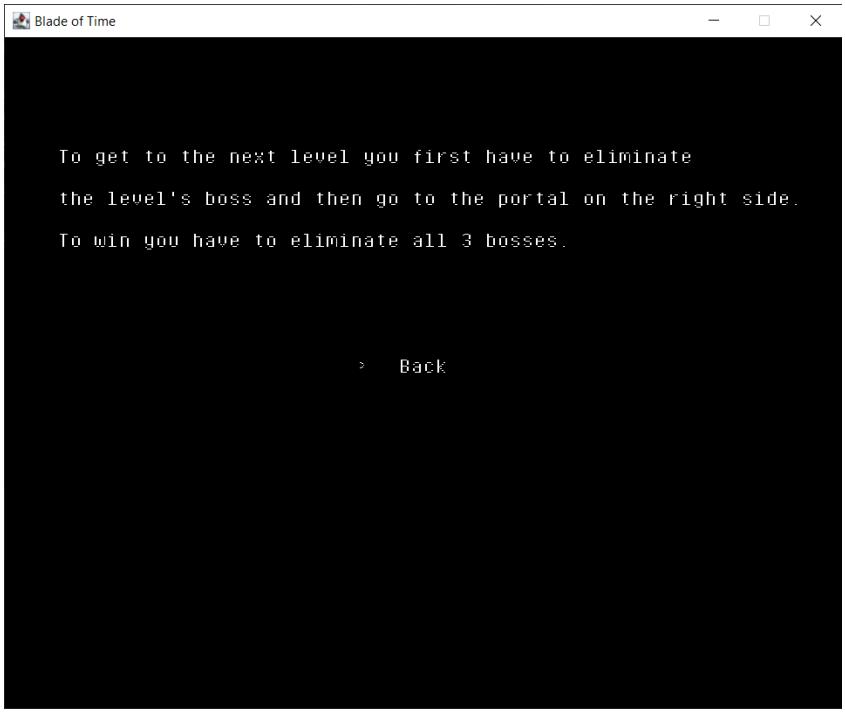
Doar prin înfrângerea lui Akaza, Rengoku va putea găsi o cale de întoarcere în trecut și să-și salveze oamenii. Dar călătoria va fi una lungă și primejdioasă, iar prințul va avea nevoie de toată priceperea și curajul său pentru a reuși.



Proiectarea Sistemului

Jocul va avea un meniu principal, de unde se poate alege Start, Load(Neimplementat), Exit, (Leaderboard) -> Help.





Jocul are 3 niveluri prin care trebuie sa treacă jucătorul, iar trecerea lor se face printr-un portal care se activează doar la eliminarea demonului final de nivel. Pentru a câștiga jocul, jucătorul trebuie să parcurgă toate nivelurile și să învingă toți demonii finali. Dacă jucătorul nostru îl va putea învinge și pe Akaza, samuraiul va reuși sa se întoarcă înapoi în timpul lui.

Jucătorul va începe în mijlocul hărții.

Jucătorul dispune de 7 inimioare, acestea vor scădea la fiecare atac al inamicului. Dacă inamicul este din fight zone, jucătorul va pierde o jumătate de inimă pentru fiecare atac primit, iar dacă se lupta cu demonii finali, va pierde o inima pentru demonul de nivel 1, o inima jumătate pentru demonul de nivel 2 și două inimi jumătate pentru demonul de nivel 3.

Modul de cameră va fi implementat deoarece viziunea jucătorului va fi mai mică decât harta nivelului respectiv.

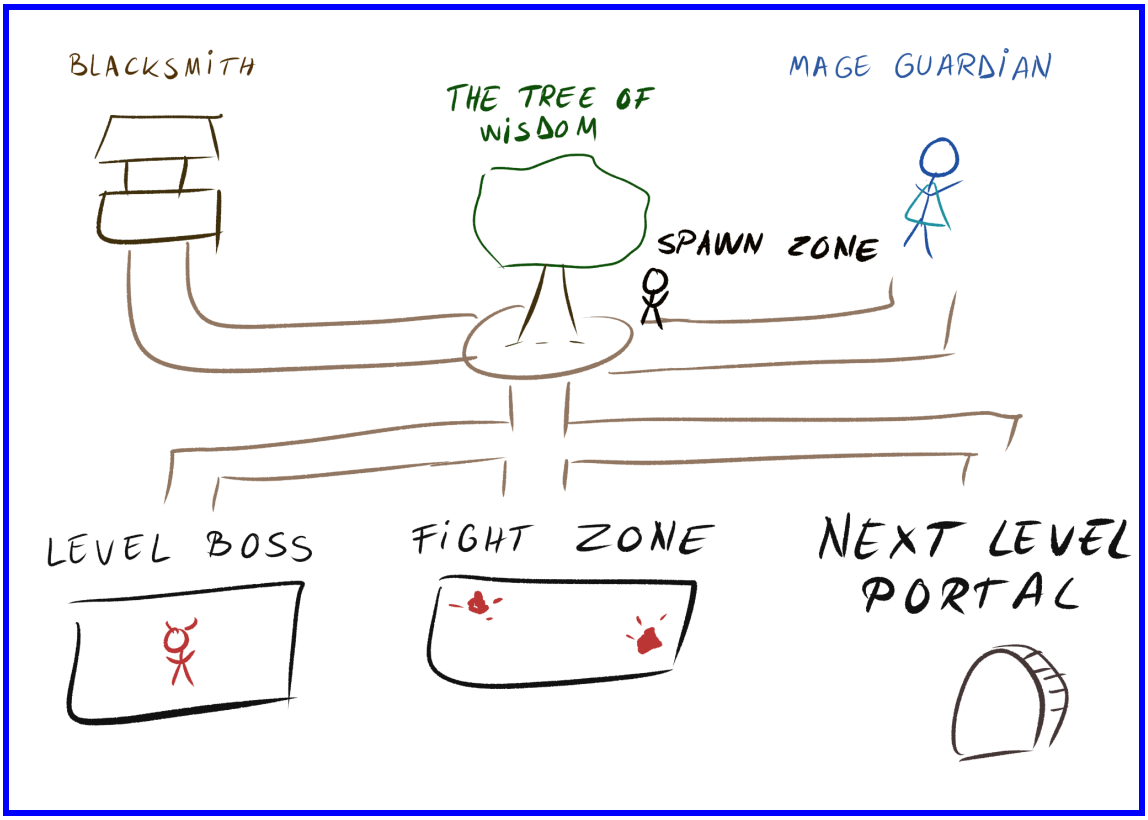
Jucătorul se poate mișca din tastele WASD.
Apasand Left-click caracterul va ataca.

Proiectarea Conținutului

Persoanaje / Zone importante

Blacksmith - fierarul care îi făurește Katana samuraiului dacă acesta deține materialele necesare. (neimplementat)
The Tree of Wisdom - Călăuzitorul samuraiului în lupta cu demonul de nivel. (Acesta nu a putut fi ucis de către Akaza, fiind protejat de o putere a zeilor) (neimplementat)
Mage Guardian - gardianul care îi oferă materiale pentru katana la schimbi pe coins. (neimplementat)
Fight Zone - zona de lupta cu demoni mai mici pentru a obține mai mult scor.
Level Boss - Demon mai puternic care trebuie ucis pentru a trece la nivelul următor. Acesta odată eliminat va oferi acces samuraiului la Next Level Portal. La nivelul 3 se află Rengoku.
Next Level Portal - Portalul către nivelul viitor. La nivelul 3, după ce samuraiul îl ucide pe Rengoku, acest portal îl va trimite înapoi în imperiul lui.

Zone



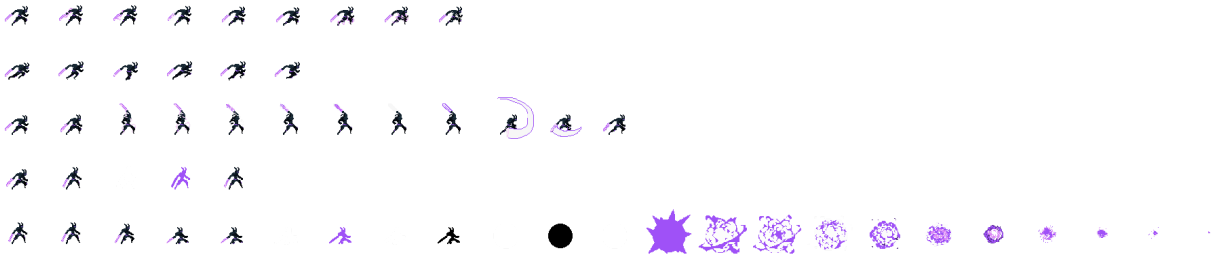
Fight zone :

Sprite-uri

Sprite WolfDemon

Sprite MagicDemon

Sprite Demon Akaza



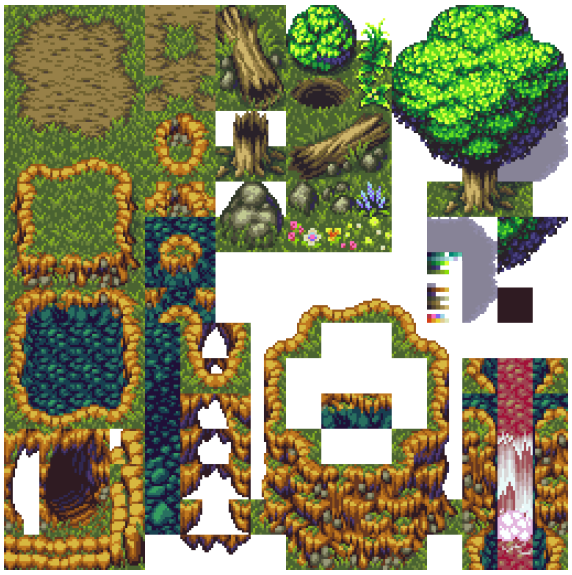
Sprite Samurai Rengoku (animatie de walk, attack, idle, defense)

Demoni din Fight Zone

Proiectarea Nivelelor

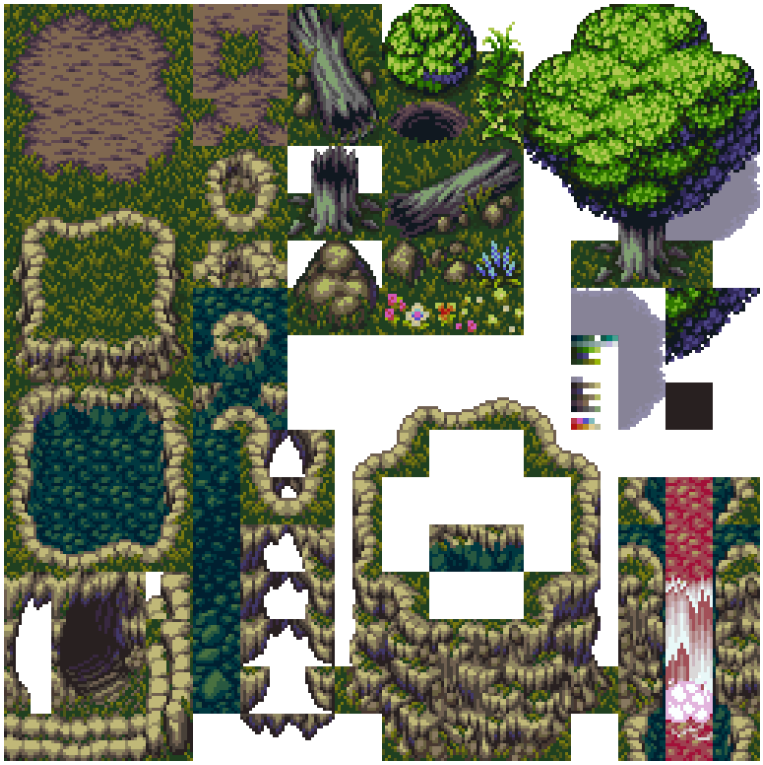
Vor fi 3 niveluri, fiecare nivel cu propria harta.

La nivelul 1, fight zone-ul va avea inamici care pot ataca doar direct. Demonul final va fi ușor de învins.

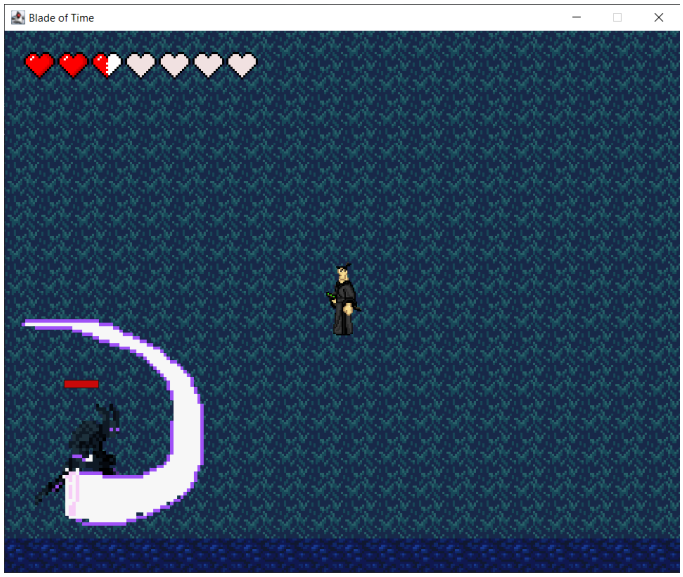


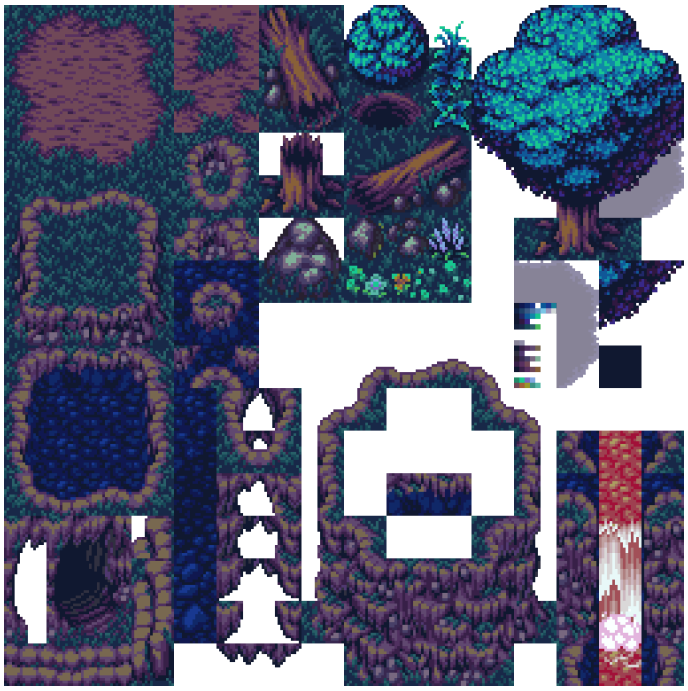
La nivelul 2, demonul final va avea mai multă viață și va ataca cu mai multă putere de atac.


Blade of Time

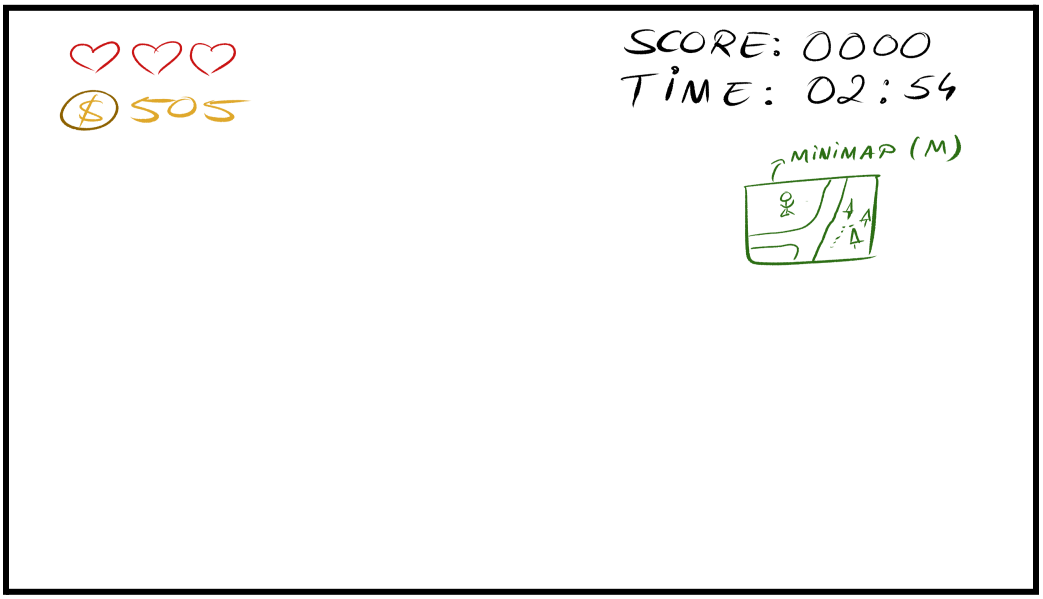


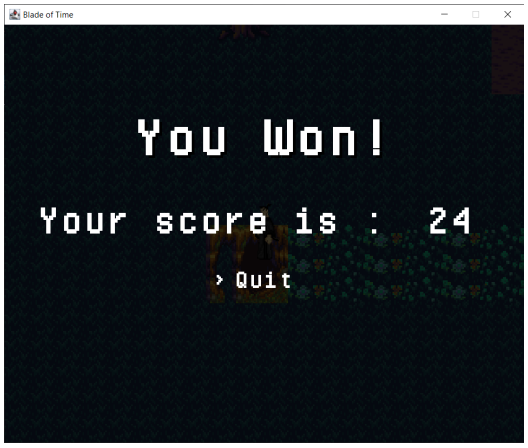
La nivelul 3, demonul final, Akaza va fi mult mai mare decat demonii inițiali și lungimea atacurilor lui vor fi de asemenea mai mari.





Proiectarea interfeței cu utilizatorul





Descrierea scheletului proiectului

În implementarea proiectului am folosit sablonul de proiectare singleton :
Clasa Game utilizează un pattern Singleton pentru a asigura că există doar o instanță a clasei Game în toată aplicația. În clasa Main, metoda main apelează Game.getInstance() pentru a obține referința la instanța unică a clasei Game. Prin această abordare, putem asigura că nu vor exista două instanțe ale jocului rulând în paralel în aceeași aplicație.

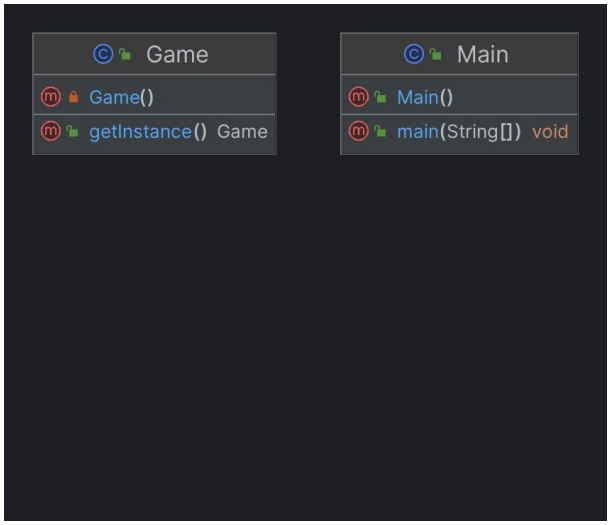
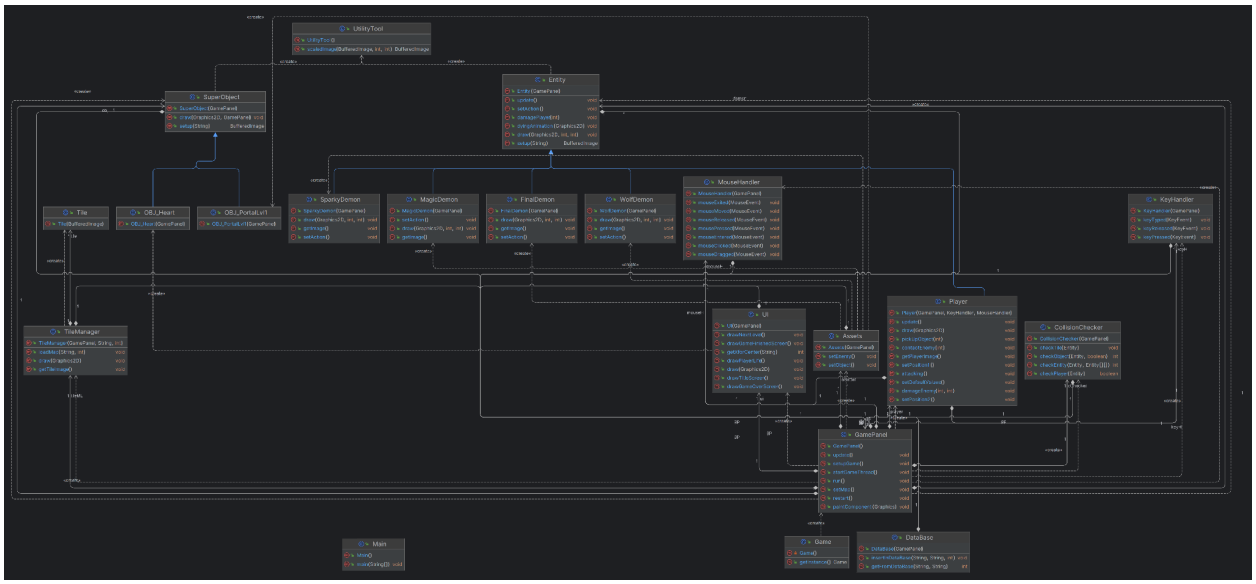
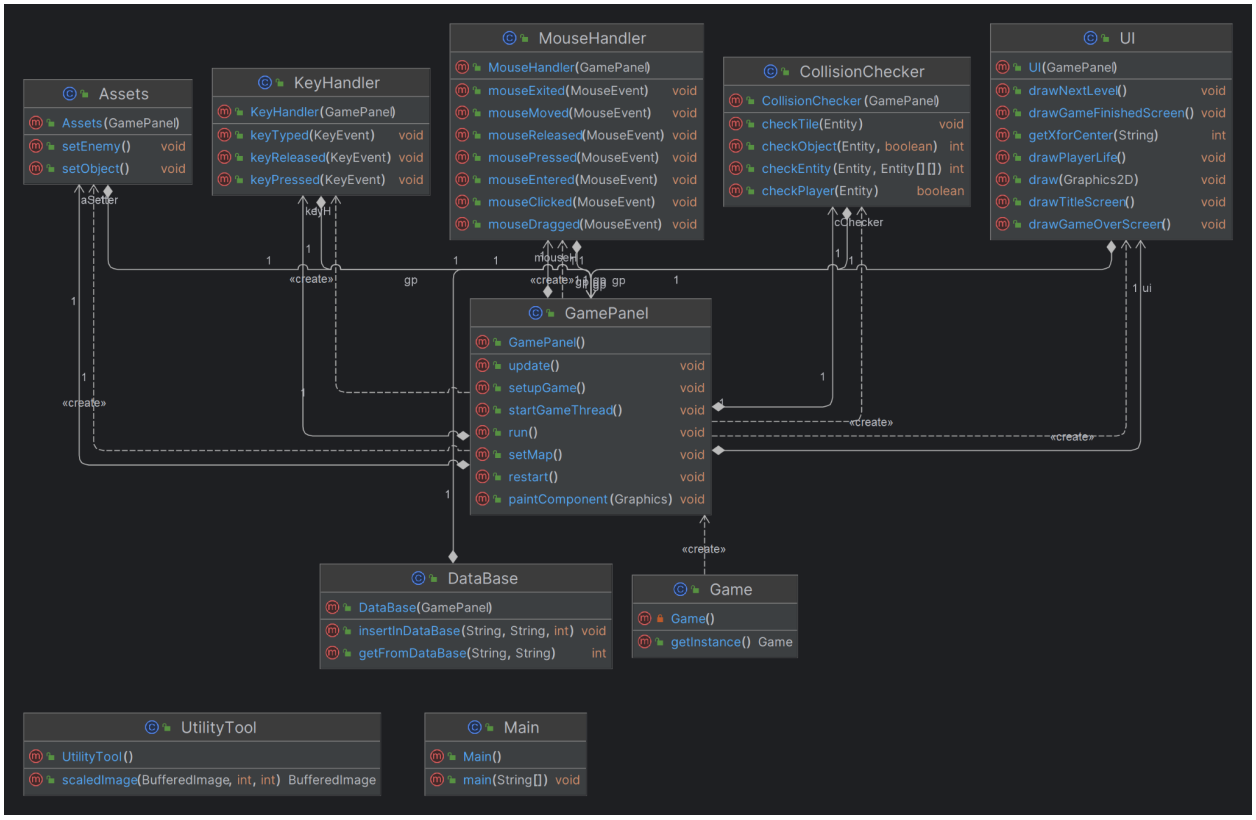


Diagrama UML a proiectului în starea sa actuală, cât și descrierea și modul de utilizare a claselor.

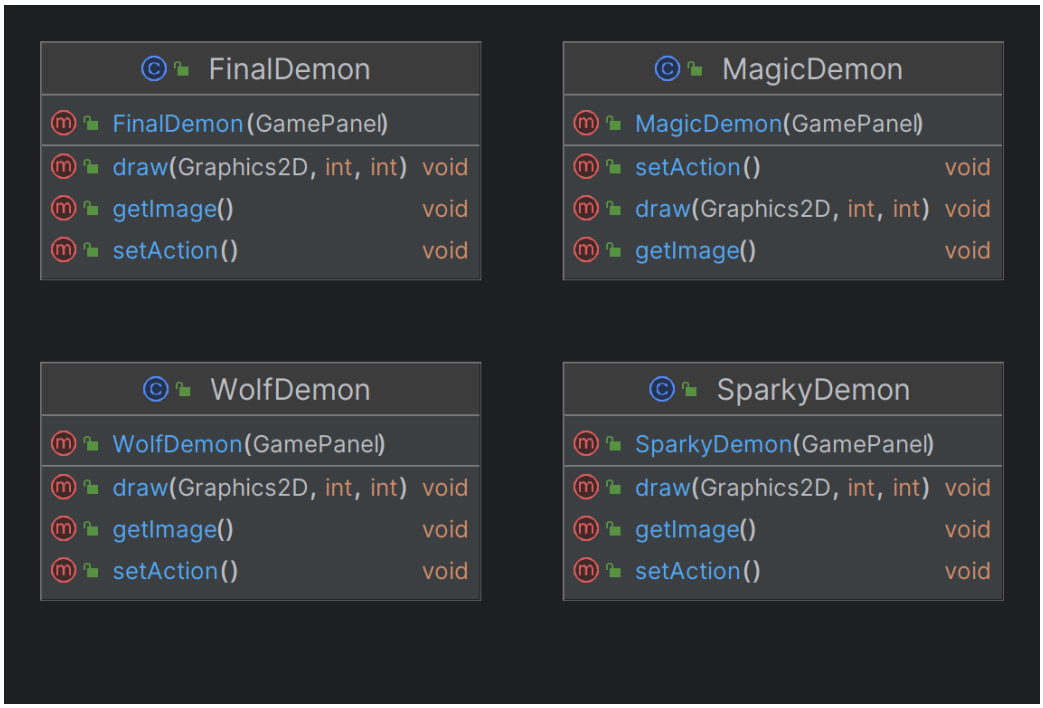
Diagrama UML
Pentru a simplifica vizualizarea, diagrama UML afișează doar metodele și constructorii claselor.



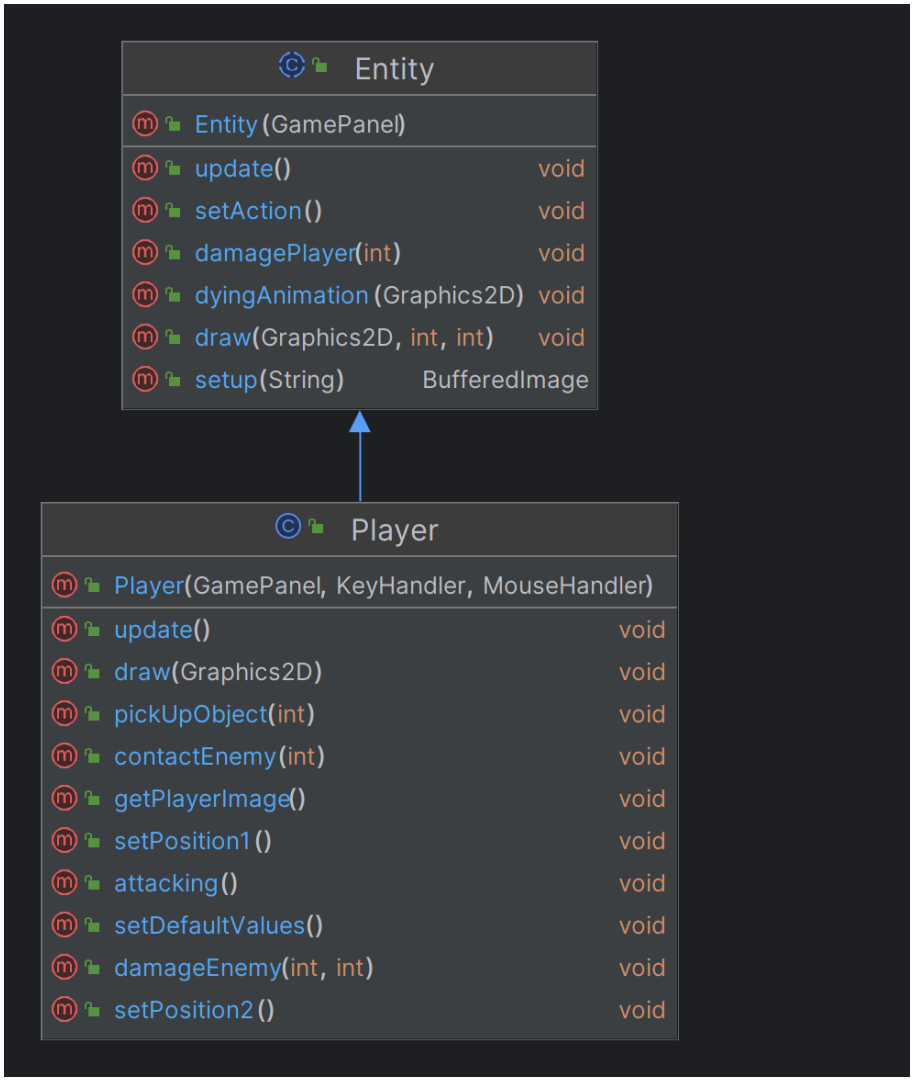
1.Main package:



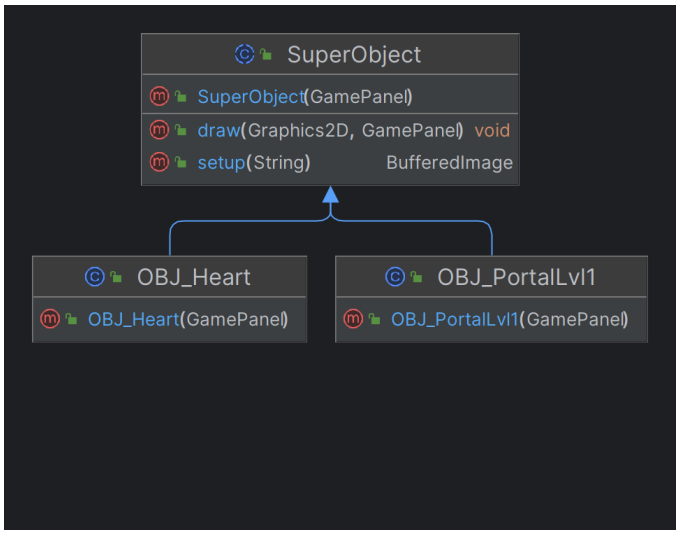
2.Enemy package



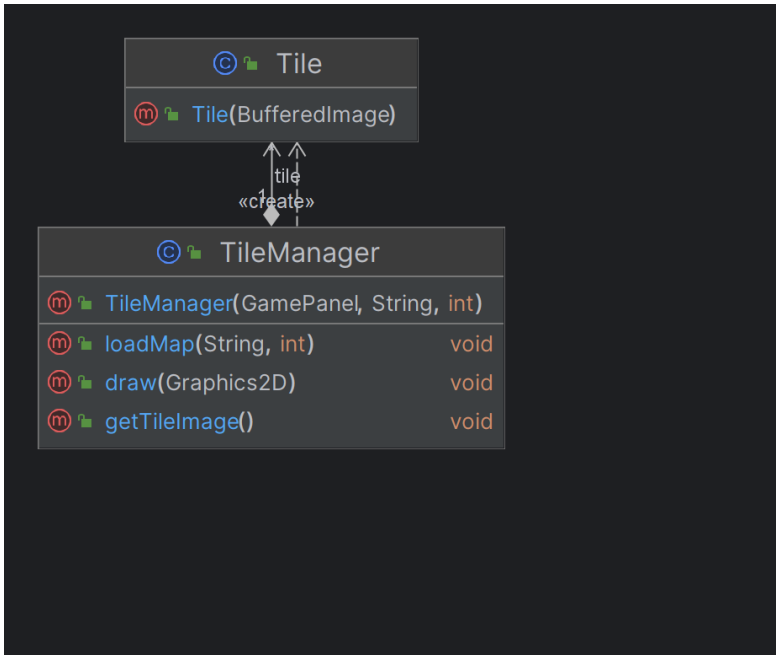
3.Entity package



4.Object package



5.Tile package



Descriere clase principale proiect:

GamePanel :

Clasa GamePanel reprezinta o componenta cheie a jocului, întrucât servește drept panou principal unde toate elementele jocului sunt desenate și interacțiunile cu utilizatorul sunt procesate.

Aceasta este o subclasă a clasei JPanel din biblioteca Swing a Java și implementează interfața Runnable.

Caracteristici principale ale clasei GamePanel:

1. Definirea mediului de joc (Sunt definite dimensiunile pentru o "caseta" (tile), ecranul și lumea jocului.)
2. Entități și obiecte de joc (Clasa include diverse instanțe ale obiectelor de joc, cum ar fi un obiect Player, o matrice de entități Entity pentru demoni și o matrice de SuperObject pentru alte obiecte în joc.)
3. Gestiunea hărților de joc (Există suport pentru mai multe hărți de joc, cu fiecare hartă având două straturi.)
4. Gestionarea stării jocului(Sunt definite mai multe stări ale jocului, cum ar fi titleState, playState, gameOverState, gameFinishedState și nextLevelState.)
5. Interacțiunea cu utilizatorul (Se folosesc KeyHandler și MouseHandler pentru a procesa intrările de la utilizator.)
6. Inițializare și resetare (Metodele setupGame() și restart() sunt utilizate pentru a iniția sau a reseta starea jocului.)
7. Actualizarea și desenarea (În funcția run(), starea jocului este actualizată și apoi desenată în mod regulat, în funcție de valoarea FPS (Frames Per Second). Mai mult, funcția update() se ocupă de actualizarea stării entităților (player și demoni), iar paintComponent() desenează diferitele componente ale jocului pe ecran.)
8. UI

Entity :

Clasa Entity este o clasă abstractă care reprezintă entitățile dintr-un joc, cum ar fi personajele și inamicii. Această clasă include diverse variabile membre pentru a gestiona poziția, viteza, viața, imagini reprezentative, direcția, coliziunile și alte caracteristici ale entității.

1. setAction(): Această metodă este utilizată în mod specific pentru entitățile care sunt inamici, pentru a seta direcția lor de mișcare. Direcția este aleasă în mod aleator.

2. update(): Această metodă actualizează starea entității, verificând posibilele coliziuni cu alte entități sau cu mediul de joc și actualizând poziția entității în funcție de direcția sa. Dacă entitatea este un inamic și intră în contact cu jucătorul, acesta din urmă va suferi daune. În plus, metoda gestionează perioada de invincibilitate a entității și animația sprite-ului în funcție de direcție și starea de coliziune.

3. setup(String imagePath): Metodă care încarcă o imagine de la o cale dată și returnează acea imagine ca un obiect BufferedImage.

4. draw(Graphics2D g2, int scale1, int scale2): Această metodă este folosită pentru a desena entitatea pe ecran, în funcție de direcția sa. În cazul în care entitatea este un inamic, este de asemenea afișată o bară de viață.

5. dyingAnimation(Graphics2D g2): Realizează o animație de "dispariție" a entității când aceasta moare.

6. damagePlayer(int attack): Această metodă este utilizată pentru a scădea viața jucătorului atunci când există o coliziune cu un inamic.

CollisionChecker :

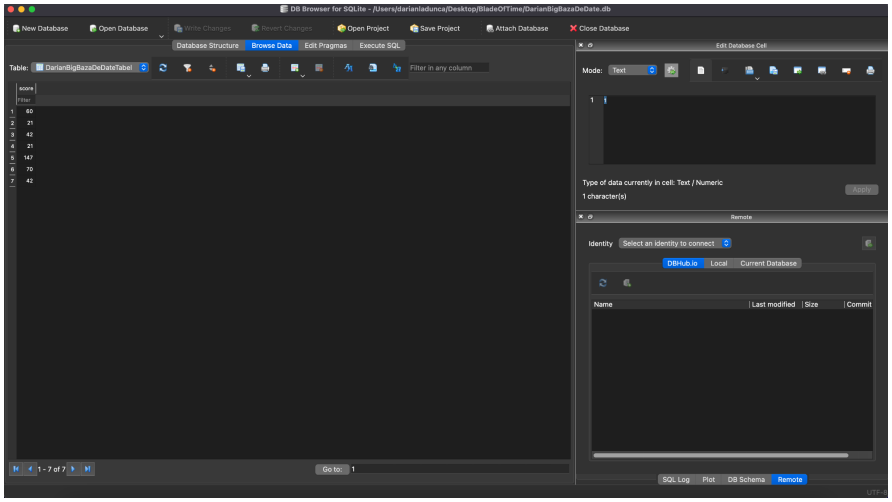
Clasa Collision Checker este utilizata pentru verificarea coliziunilor din joc.
Metodele clasei :

1. Metoda checkTile(Entity entity) verifică dacă o anumită entitate se ciocnește cu un tile solid. În primul rând, sunt calculate coordonatele marginilor entității în sistemul de coordonate al lumii de joc. Aceste coordonate sunt folosite apoi pentru a determina tile-urile cu care entitatea ar putea intra în coliziune. Dacă entitatea se îndreaptă spre un tile solid, metoda va seta entity.collisionOn pe true.
2. Metoda checkObject(Entity entity, boolean player) verifică dacă o anumită entitate se ciocnește cu un obiect din joc. Similar cu metoda checkTile, ea calculează poziția potențială a entității după mișcare și verifică dacă aceasta ar intersecta un obiect solid. În cazul în care se detectează o coliziune, entity.collisionOn este setat pe true. Dacă player este true, metoda returnează indexul obiectului cu care entitatea se ciocnește.
3. Metoda checkEntity(Entity entity, Entity[][] target) funcționează similar cu checkObject, dar în loc să verifice coliziunile cu obiecte, aceasta verifică coliziunile cu alte entități. Ea returnează indexul entității cu care s-a produs coliziunea.
4. În final, metoda checkPlayer(Entity entity) verifică dacă o anumită entitate a intrat în coliziune cu jucătorul. Dacă se produce o coliziune, entity.collisionOn este setat pe true și metoda returnează true.

DataBase :

Clasa DataBase din pachetul principal este o clasă utilitară folosită pentru manipularea datelor dintr-o bază de date SQLite.
Metodele clasei :

1. Metoda insertInDataBase(String nume_fisier, String nume_tabela, int a) este folosită pentru inserarea datelor în baza de date. Aceasta primește ca argumente numele fișierului de bază de date, numele tabelei și valoarea care trebuie inserată.
2. Metoda getFromDataBase(String nume_fisier, String nume_tabela) este utilizată pentru a extrage date din baza de date.(valoarea coloanei score din fiecare înregistrare).



TileManager :

Clasa TileManager este responsabilă pentru gestionarea 'tile'-urilor sau a celulelor pe care se desfășoară jocul.
Metodele clasei :

1. Metoda loadMap(String a, int mapNumber) este utilizată pentru a încărca o hartă dintr-un fișier text. Ea citește fișierul linie cu linie, înlătură spațiile și virgulele, și stochează numerele în matricea mapTileNum.
2. Metoda getTileImage() este folosită pentru a încărca imaginile pentru 'tile'-uri și pentru a stabili coliziunile. Ea citește o imagine în funcție de harta curentă și apoi o divide în mai multe 'tile'-uri, pe care le stochează în vectorul tile. În funcție de anumite condiții, fiecare 'tile' poate fi marcat ca având coliziune sau nu.
3. Metoda draw(Graphics2D g2) este utilizată pentru a desena 'tile'-urile pe ecran. Ea calculează poziția 'tile'-urilor pe ecran în funcție de poziția jucătorului și apoi le desenează folosind obiectul Graphics2D primit ca argument.

Resurse:

- **Informatii curs/laborator**
- **https://itch.io/**
- **https://www.pinterest.com/**
- **Samurai Jack / Cartoon Network**