

Anexa 1

A Ronin's Journey

Lupu Andra Maria

1209A

1. Povestea jocului

Jocul este situat în perioada Edo din Japonia (1603 – 1868) în care, personajul principal, Jin, este un ronin, adică un samurai ajuns în dizgrație.

Jin, personajul principal, este un samurai desăvârșit. Acesta s-a antrenat ani de zile sub aripa unuia dintre cei mai renumiți samurai din acea perioadă, dar pentru că era războinicilor samurai era spre sfârșit, sensei-ul lui Jin a fost obligat să transforme dojo-ul într-o școală pentru asasini. Jin care nu era de acord ajunge să își înfrunte maestrul. În urma luptei dintre cei doi, sensei-ul lui Jin a decedat, iar în ultimele clipe ale sale, acesta îi mărturisește lui Jin că este mândru de el.

După acest eveniment tragic, Jin, alungat, respins și izolat, începe o călătorie pentru a își ispăși păcatele.

În timpul lungilor sale călătorii pe teritoriul Japoniei, ajunge într-o ceainărie, unde o întâlnește pe Fuu. Această este o chelneriță, a cărei mama a murit de curând, și al cărei tată a abandonat-o la o vârstă fragedă.

În timpul petrecut de Jin în ceainărie, Fuu este jignită de un grup de clienți. Jin fiind martor la acest comportament nepotrivit, sare în ajutorul lui Fuu, luându-i apărarea. După acest act de eroism din partea acestui samurai misterios Fuu îi propune să o ajute în găsirea unui anumit om. Acest om este tatăl lui Fuu, un fost samurai, care acum mulți ani a iscat o revoltă împotriva Shogun-ului. Deoarece revolta nu a fost de succes acesta a trebuit să se ascundă, lăsându-și familia în urmă pentru nu a o pune în pericol.

Acțiunea începe după ce Fuu îl angajează pe Jin pentru a își găsi tatăl.

În călătoria lui Jin de a îl găsi pe tatăl lui Fuu, acesta dă peste multe obstacole, capcane dar și inamici. Aceste piedici regăsite în misiunea lui Jin, sunt amplasate de Shogun (Titlu dat dictatorilor militari ai Japoniei între 1192 și 1867), deoarece acestuia îi este frică de o nouă revoluție.

Prezentare joc:

Campanii pentru un singur jucător în care jucătorul trebuie să treacă de obstacole/capcane colectând indicii, să învingă inamicii și să ajungă la o ieșire pentru a trece la nivelul următor.

Strategia de joc - SINGLEPLAYER

Reguli joc:

Jocul implică parcurgerea unui traseu cu obstacole și lupta dintre protagonist și inamici. **Jucătorul este ucis dacă este atins de 5 ori de inamic.** Arma protagonistului este o sabie/katana. Până să ajungă jucătorul la inamici el trebuie să mai treacă și de niște obstacole/capcane puse în calea sa de către Shogun. **Dacă jucătorul este prins într-o capcană, v-a pierde o viață, la fel dacă cadem în gol.**

Dacă colectează toate indiciile, primește 5 puncte, plus primești o viață în plus.

Putem ataca în timp ce sarim, dar nu în timp ce mergem.

Personajele jocului:

Jin este protagonistul și jucătorul-personaj. El este de obicei descris ca fiind calm, putând să pară la o primă vedere indiferent și stoic, dar eroic, cu un simț al dreptății foarte puternic. Acesta nu se ferește de a îi ajuta pe cei în nevoie.



Fuu este personajul de la care pornește intriga jocului. Aceasta este o fire veselă, a cărei dorință este de a își găsi tatăl, cu scopul de a se răzbuna pe acesta.



Mugen - inamicul final din călătoria protagonistului. Acesta este un samurai angajat de Shogun-ul din acea perioadă.



Mukuro - inamicul întâlnit în nivelul doi. Acesta este un asasin/ninja trimis de către Shogun pentru a îl împiedică pe Jin din a își duce la capăt misiunea.



Tabla de joc: (side view, platformer)

- **Componente pasive:** iarba, sol, piatra, copac
- **Componente active :** obstacolele/capcane, indicii
- Structura tablei de joc (elemente minime, dispunere etc.) și modul în care este construită

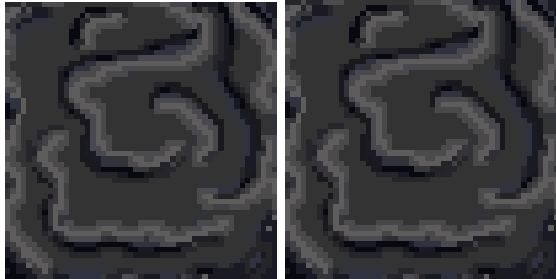
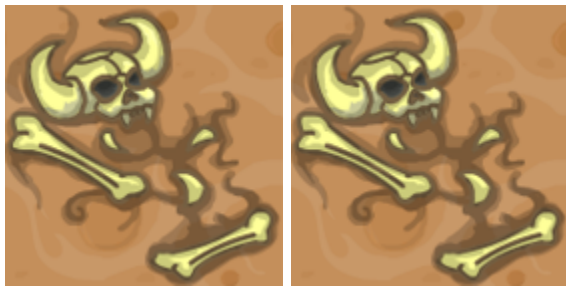
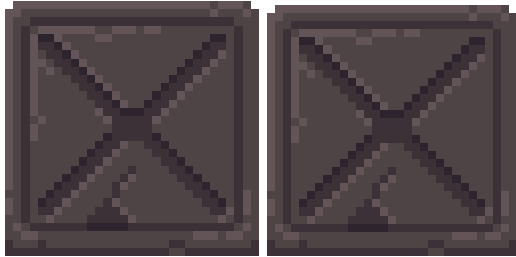




Indicii



Capcane - daca o atingi, pierzi o viata



Mecanica jocului:

Jucătorul se poate mișcă cu butoanele:

- deplasare în față : ->
- deplasare în spate: <-
- poate sări de pe **space**
- poate lovi cu sabia de pe **R**

Jucătorul v-a trebui să se ferească de capcane, totodată colectând și indiciile. Dacă te prinde o capcană pierzi o viață.

Game Sprite

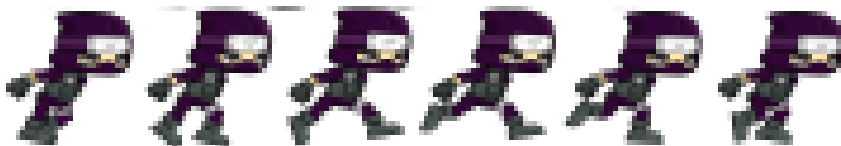
Jin - atac



Mugen - mers

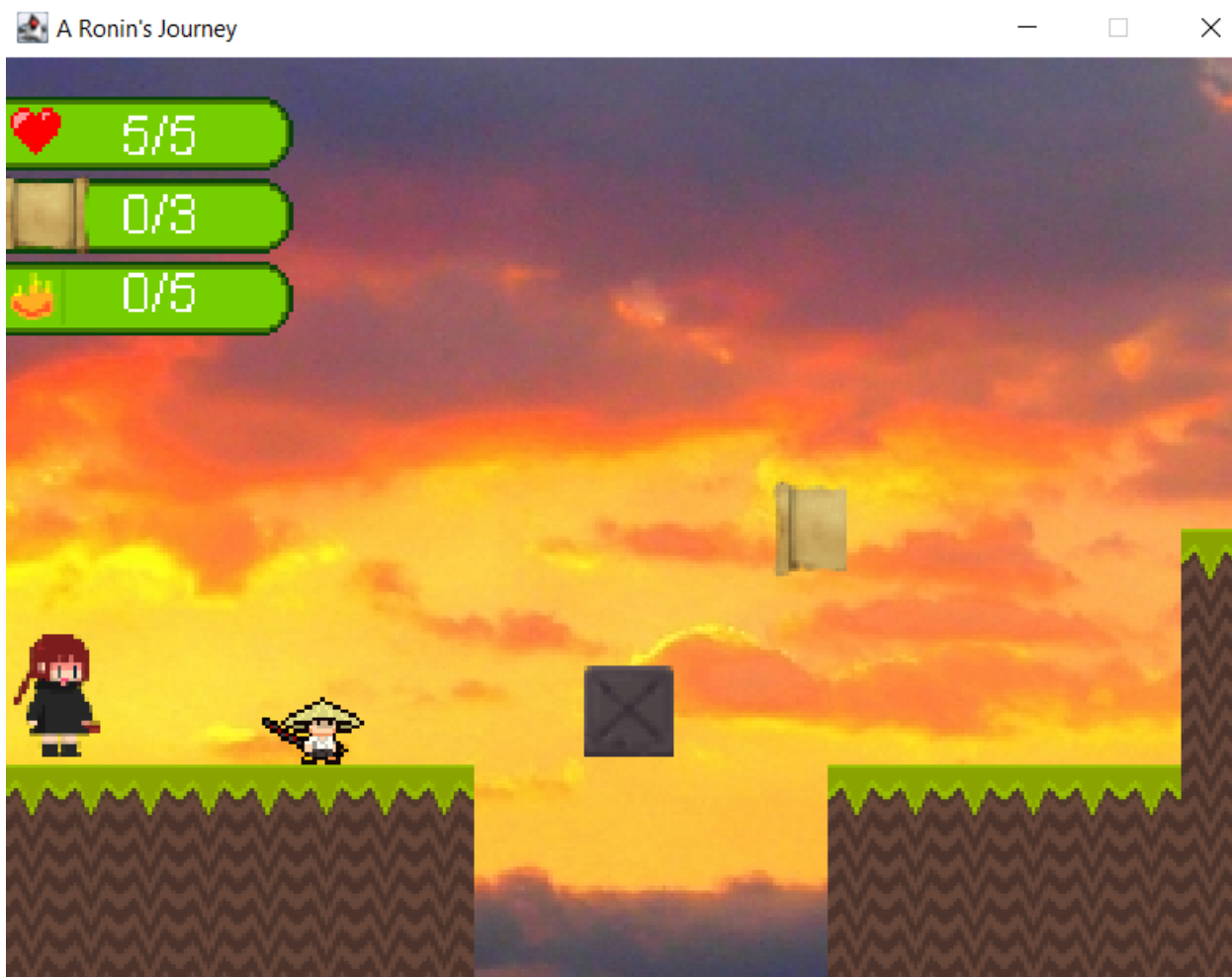


Mukuro - salt



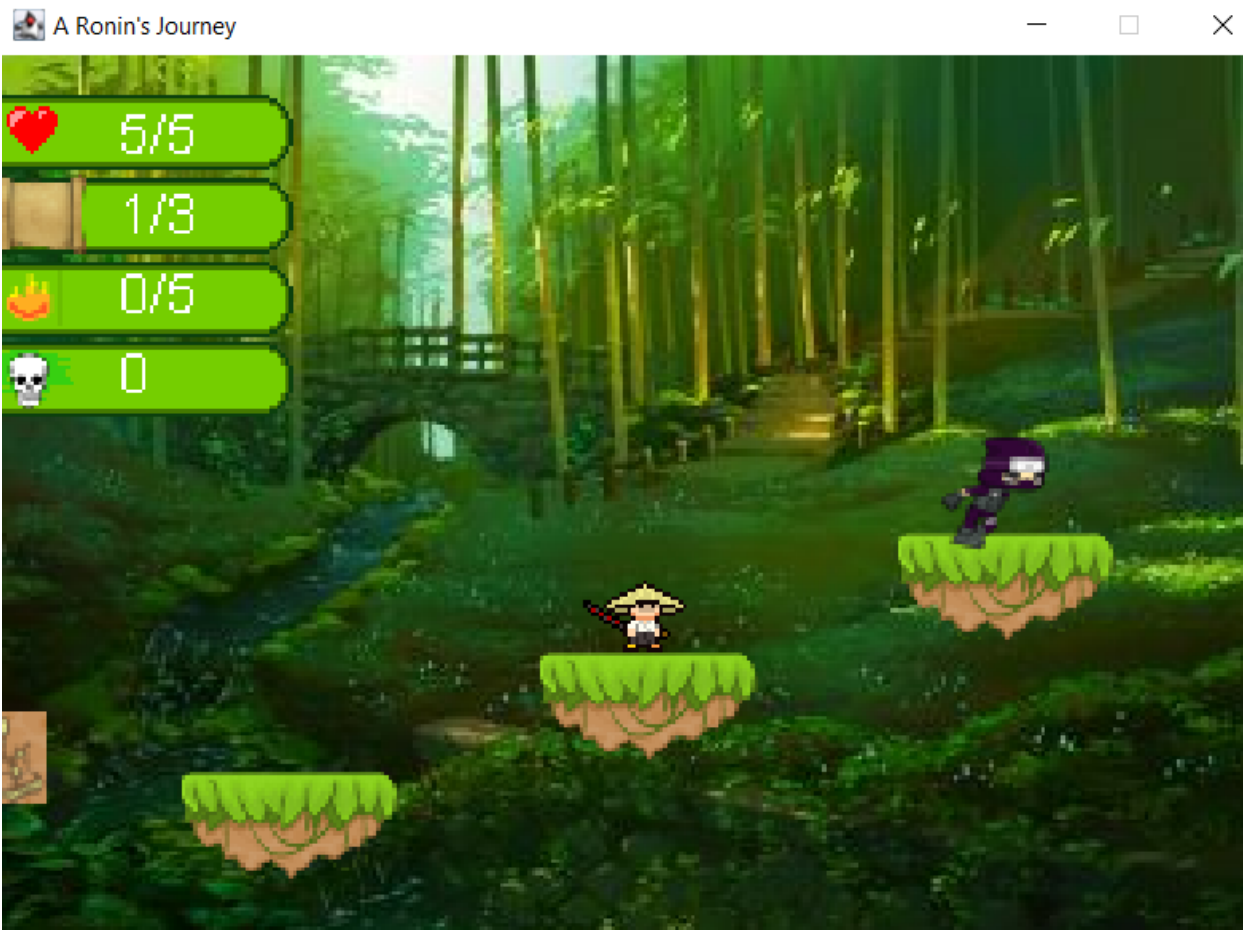
Descriere fiecare nivel

Primul nivel reprezintă prima parte din călătoria personajului nostru, Jin. În acest nivel, jucătorul trebuie să adune cât mai multe indicii cu scopul de al găsi în cel mai scurt timp pe tatăl lui Fuu. Acest nivel presupune și trecerea de obstacolele/capcanele plasate de către Shogun în drumul lui Jin.



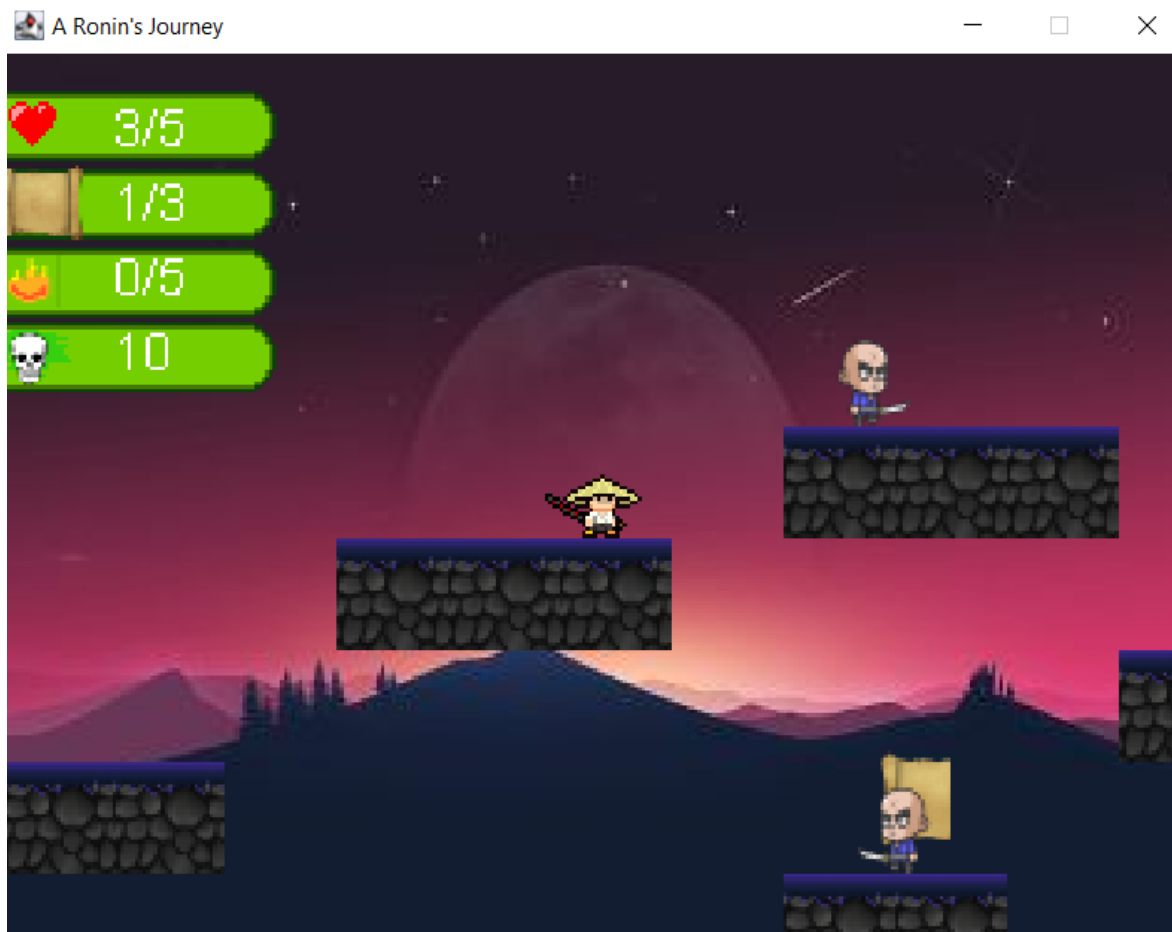
Al doilea nivel este a doua etapă din călătoria lui Jin. Această etapă devine mult mai grea pentru Jin, deoarece Shogun-ul a realizat că l-a subestimat pe samurai-ul nostru. Al doilea nivel include un traseu asemănător cu cel din primul nivel, totodată aici îl vom întâlni pe primul nostru inamic, Mukuro, un asasin însărcinat de către Shogun să scape de Jin.

Inamicii din acest nivel, de tip Mukuro, sunt inamici slabi, cu o singura viață, din o lovitură vor muri.



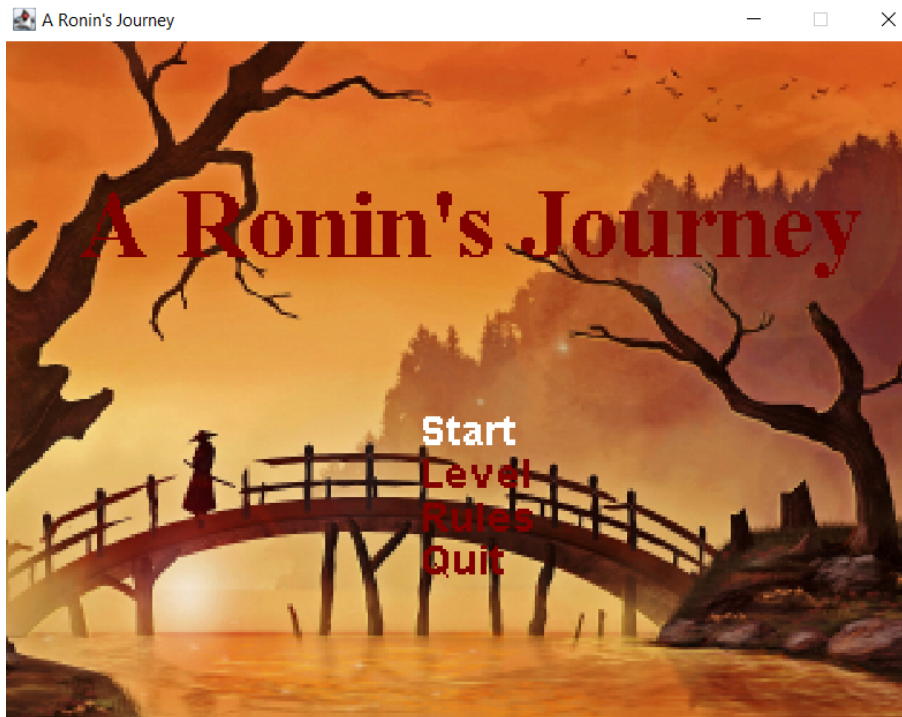
Al treilea nivel constituie ultima parte din călătoria lui Jin. După multe atacuri eșuate, Shogun-ul trimite cel mai bun războinic al său, Mugen, un samurai excentric, violent și foarte puternic. Ultimul nivel îl prezintă pe Jin obosit, la capătul puterilor, dar perseverând împotriva dificultăților întâlnite pe parcurs. După un drum anevoios, cu multe obstacole menite să îl obosească psihic și fizic, acesta este confruntat de Mugen.

Inamicii din acest nivel, de tip Mugen, vor fi mai greu de invins, deoarece vor muri din 2 lovituri.

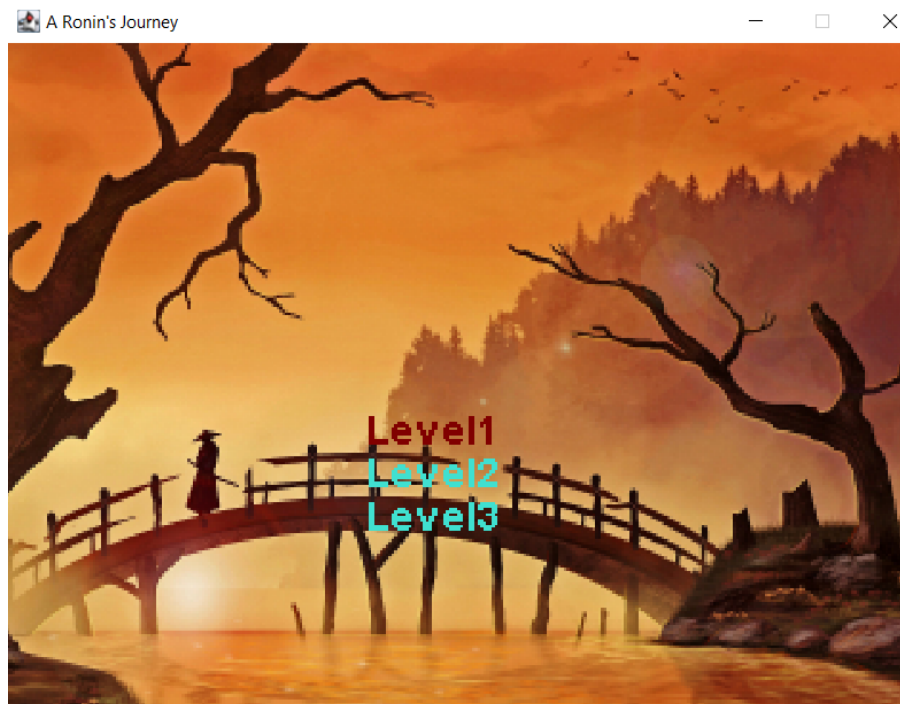


Descriere meniu

Meniul v-a contine Start (de unde intram in nivelul 1), Level (de unde alegem ce nivel dorim), Rules (cum ne deplasam), Quit (iesire din joc)



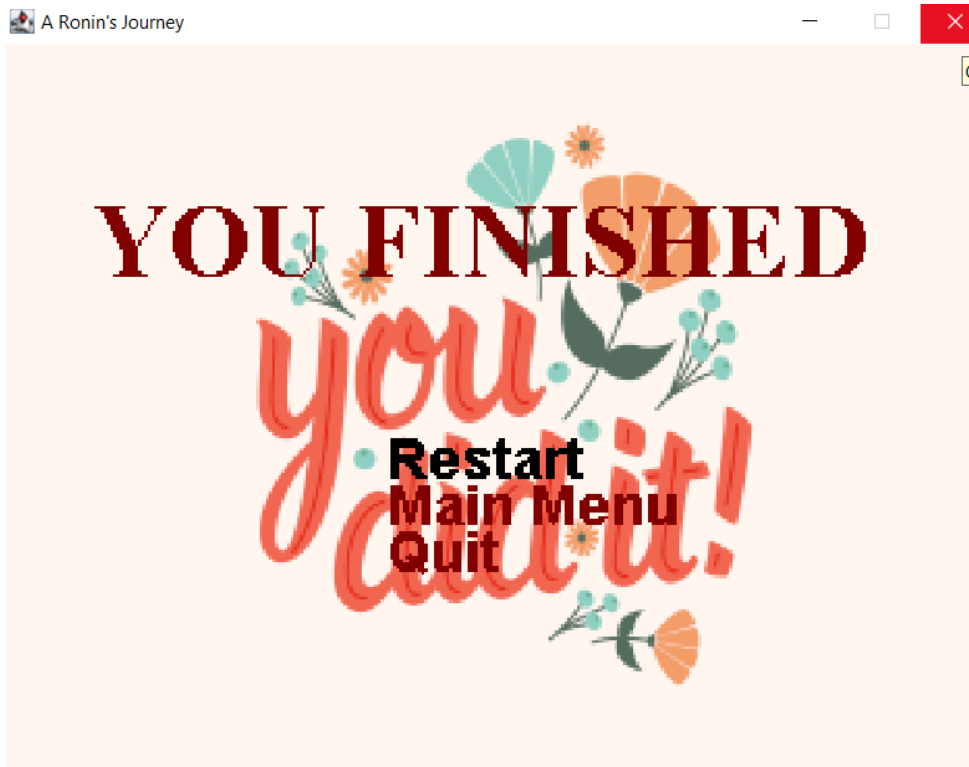
Level State



Game Over



Finish Game



Rules State (cum ne deplasam)

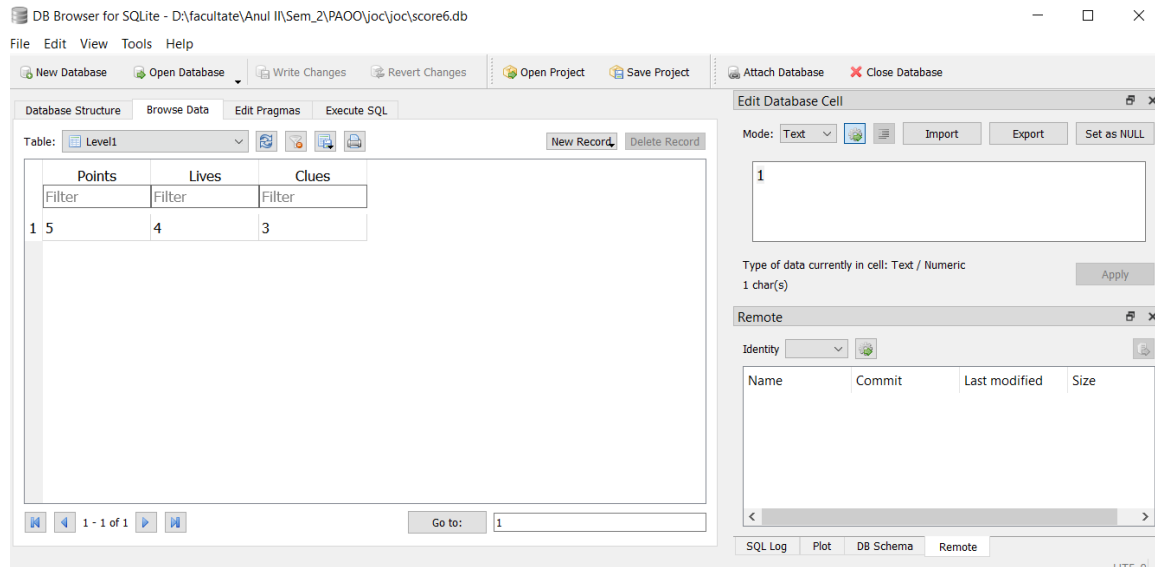
A Ronin's Journey



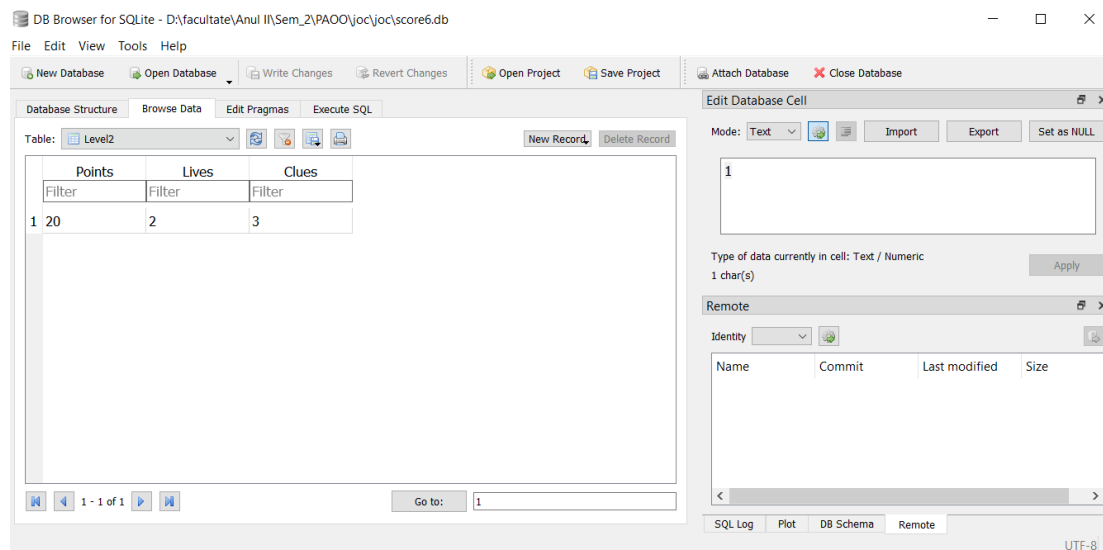
ANEXA 2

Structura Bazei de Date

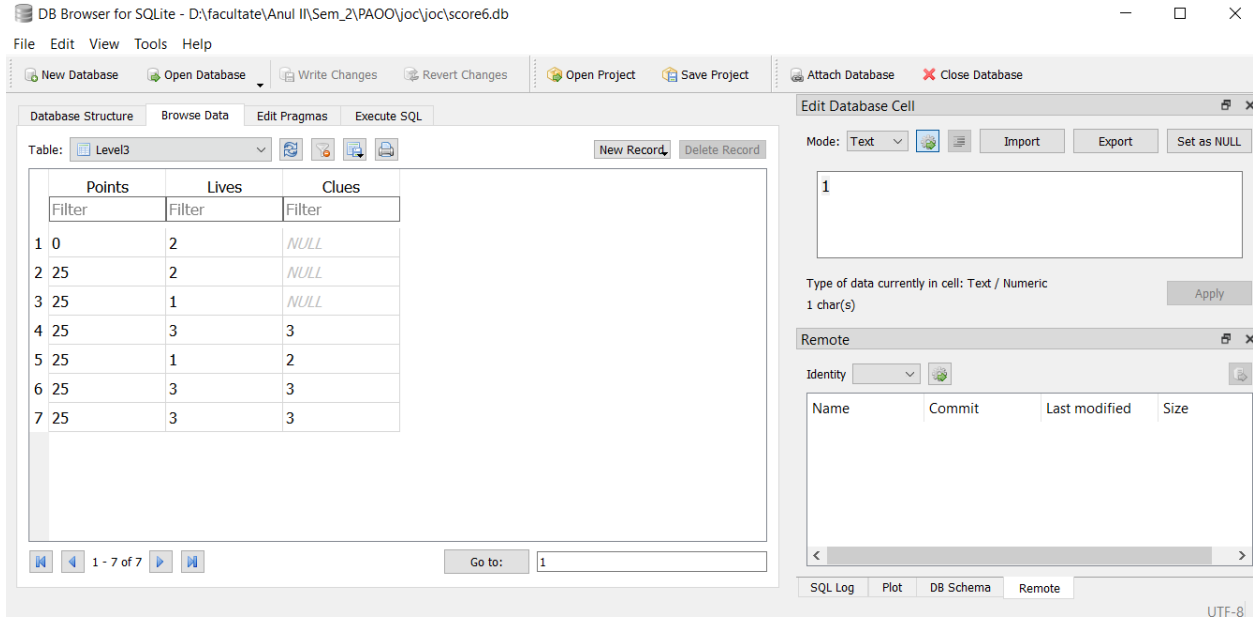
Am creat o Tabela pentru **nivelul 1**, unde am inserat Scorul (5 puncte daca am colectat toate indiciile), Lives (cate vieti mai aveam la finalul nivelului), Clues (cate indicii am reusit sa colectez)



Am creat o Tabela pentru **nivelul 2**, unde am inserat Scorul (5 puncte pentru fiecare inamic invins), Lives (cate vieti mai aveam la finalul nivelului), Clues (cate indicii am reusit sa colectez)



Am creat o Tabela pentru **nivelul 3**, unde am inserat Scorul (5 puncte pentru fiecare inamic invins), Lives (cate vieti mai aveam la finalul nivelului), Clues (cate indicii am reusit sa colectez)



DB Browser for SQLite - D:\facultate\Anul II\Sem_2\PAOO\joc\joc\score6.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragas Execute SQL

Table: Level3

| | Points | Lives | Clues |
|---|--------|--------|--------|
| | Filter | Filter | Filter |
| 1 | 0 | 2 | NULL |
| 2 | 25 | 2 | NULL |
| 3 | 25 | 1 | NULL |
| 4 | 25 | 3 | 3 |
| 5 | 25 | 1 | 2 |
| 6 | 25 | 3 | 3 |
| 7 | 25 | 3 | 3 |

1 - 7 of 7

Go to: 1

Edit Database Cell

Mode: Text

1

Type of data currently in cell: Text / Numeric

1 char(s)

Apply

Remote

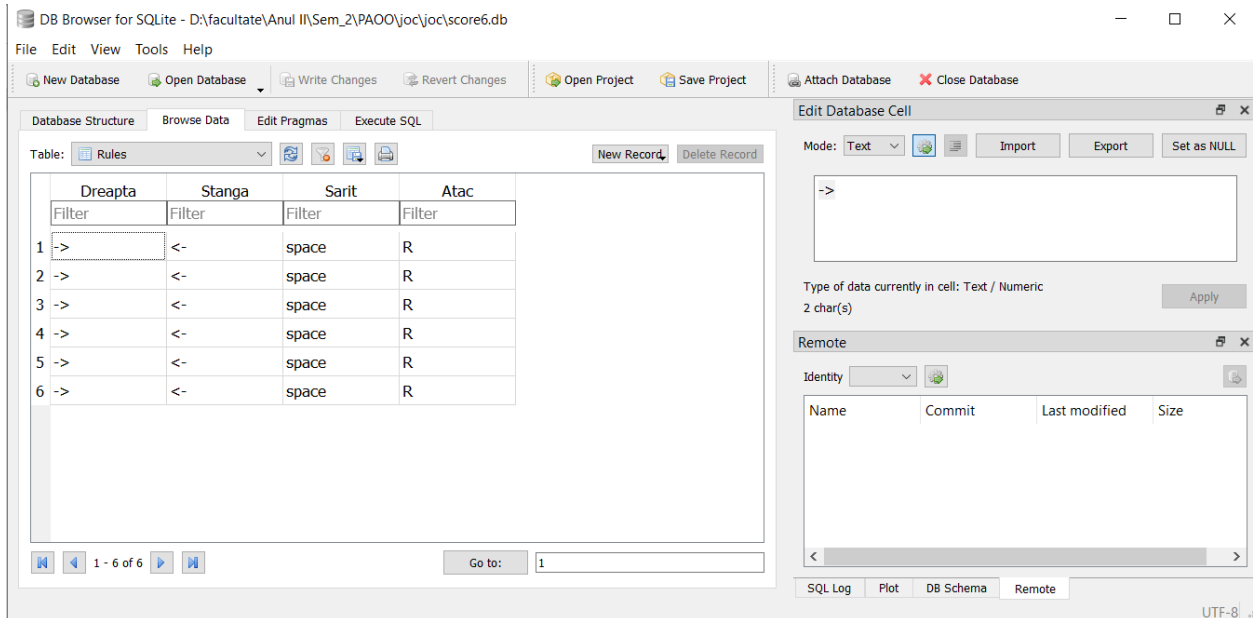
Identity

| Name | Commit | Last modified | Size |
|------|--------|---------------|------|
| | | | |

SQL Log Plot DB Schema Remote

UTF-8

Am creat o tabela pentru **Reguli**, unde am inserat de pe ce butoane ne jucam



DB Browser for SQLite - D:\facultate\Anul II\Sem_2\PAOO\joc\joc\score6.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragas Execute SQL

Table: Rules

| | Dreapta | Stanga | Sarit | Atac |
|---|---------|--------|--------|--------|
| | Filter | Filter | Filter | Filter |
| 1 | -> | <- | space | R |
| 2 | -> | <- | space | R |
| 3 | -> | <- | space | R |
| 4 | -> | <- | space | R |
| 5 | -> | <- | space | R |
| 6 | -> | <- | space | R |

1 - 6 of 6

Go to: 1

Edit Database Cell

Mode: Text

->

Type of data currently in cell: Text / Numeric

2 char(s)

Apply

Remote

Identity

| Name | Commit | Last modified | Size |
|------|--------|---------------|------|
| | | | |

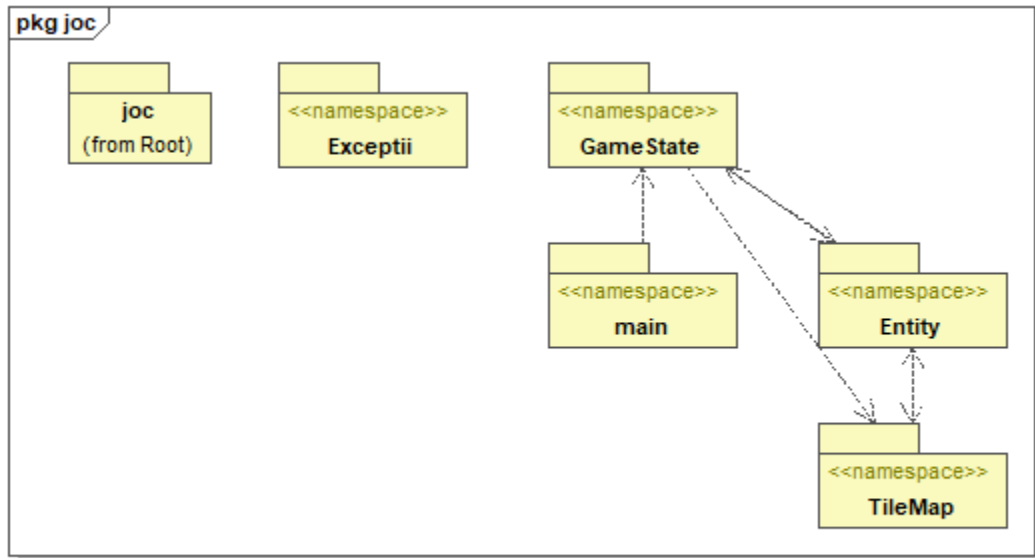
SQL Log Plot DB Schema Remote

UTF-8

Diagrama de Clase

Descriere clase proiect

Packages dependencies

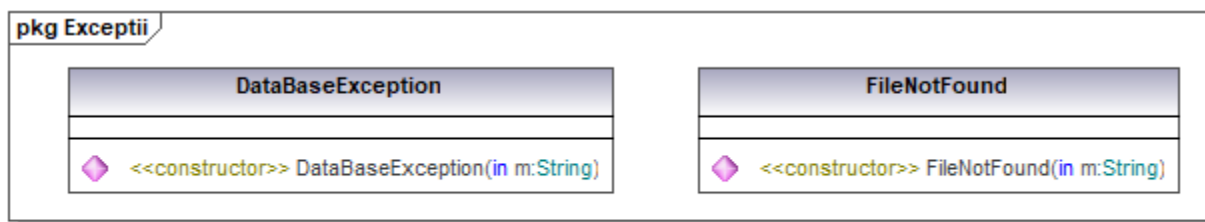


Generated by UModel

www.altova.com

Exceptii package

Clasele pentru tratarea exceptiilor.



Generated by UModel

www.altova.com

FileNotFound

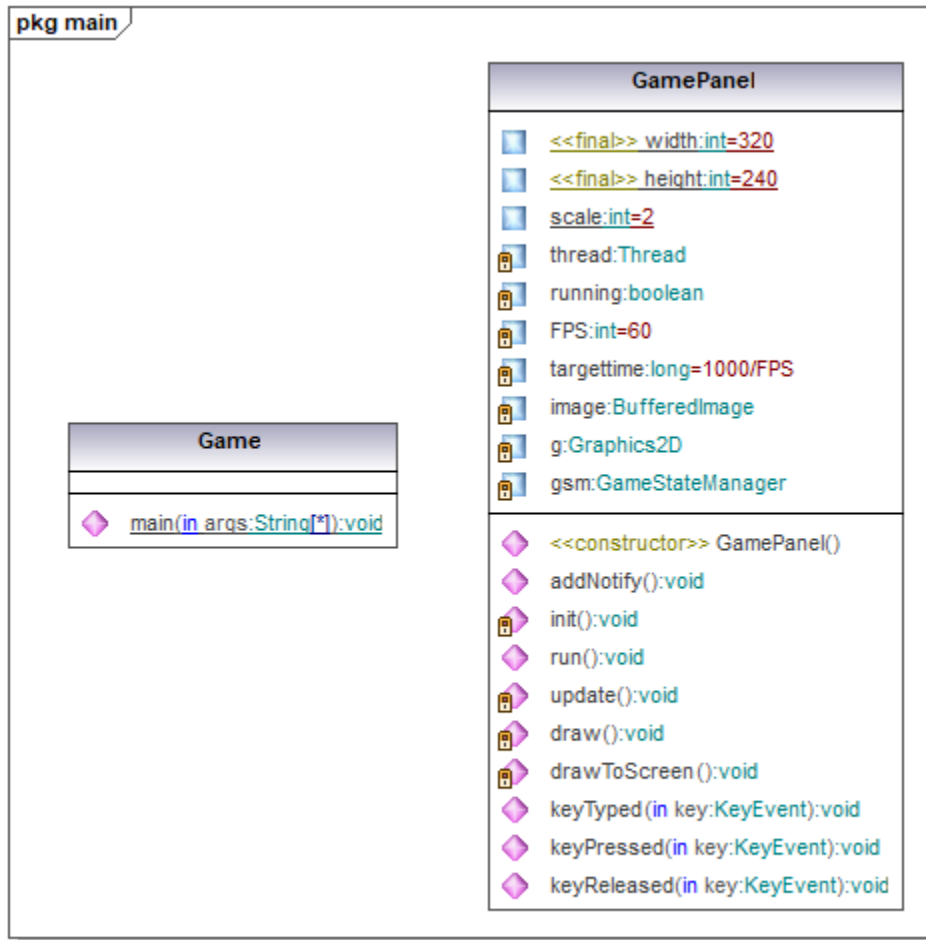
Aceasta clasa am folosit-o pentru aruncarea/prinderea exceptiilor in care citeam/incarcam o imagine care nu exista in fisierul cu Resurse.

DataBaseException

Aceasta clasa am folosit-o pentru aruncarea/prinderea exceptiilor in metoda draw din Clasa ScoreState, unde incarcam scorul jucatorului in baza de date. Aceasta

exceptie o aruncam in momentul in care dorim sa afisam scorul la finalul jocului, iar tabela este goala.

Main Package



Generated by UModel

www.altova.com

Game

Clasa implementeaza si afiseaza fereastra jocului, totodata creeaza o instantanta de tipul GamePanel.

GamePanel

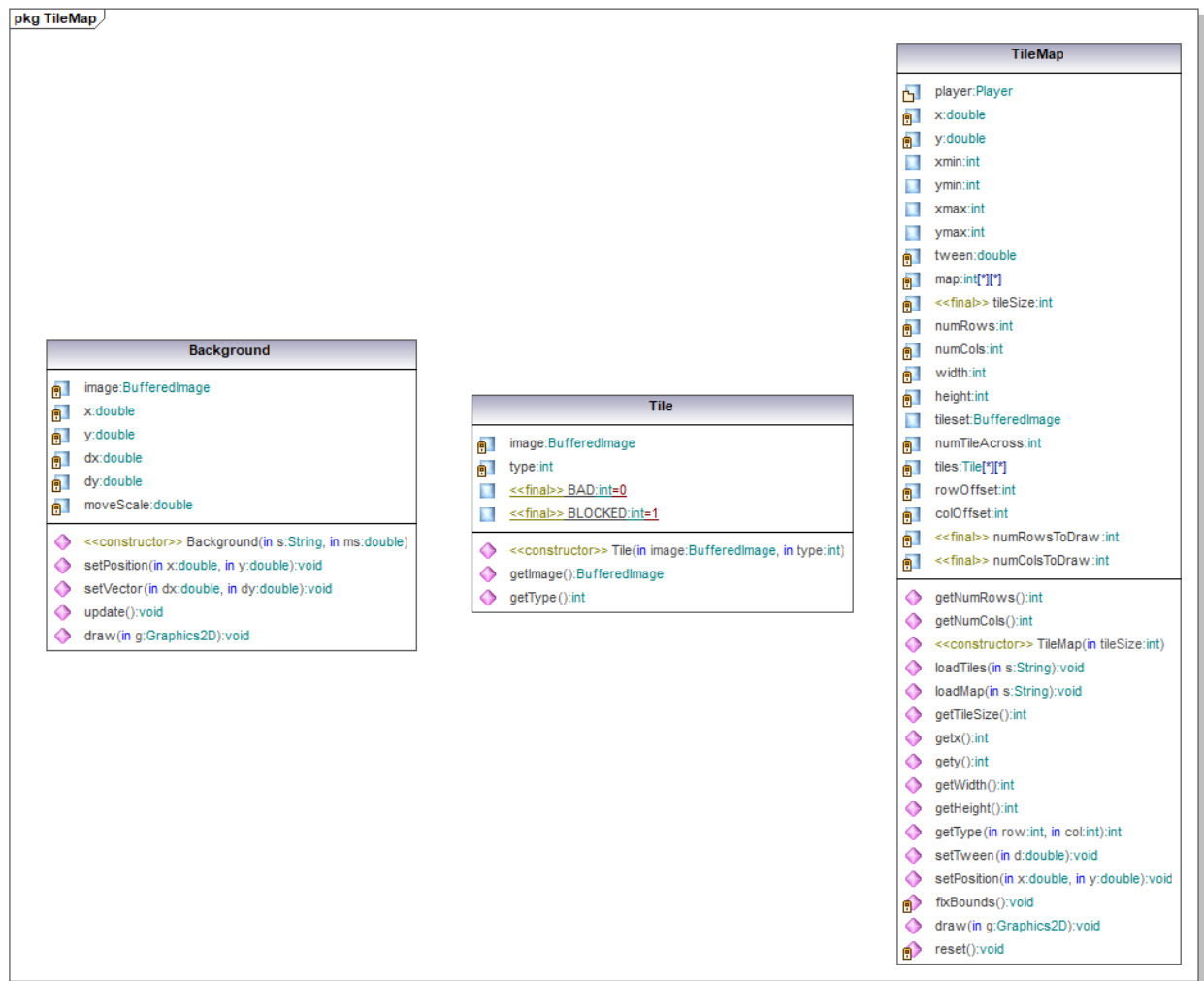
Clasa mosteneste Clasa **JPanel** si implementeaza **Runnable** si **KeyListener**. Această clasă implementează interfața Runnable pentru a avea comportamentul unui fir de execuție (thread). În momentul apelului metodei run() se instanțiază un obiect de tip Thread pe baza instanței curente a clasei GamePanel. Orice obiect de tip Thread trebuie să implementeze metoda run() care este apelată atunci când firul de execuție este pornit . Această metodă run() inițializează jocul prin crearea unei instanțe

GamePanel, iar mai apoi controlează numărul de cadre pe secundă printr-o buclă while și “pregătește” noua scenă pe care o va desena pe interfața grafică (draw()). Metoda update() actualizează starea jocului (de exemplu: modifica poziția jucătorilor pe baza tastelor apăsate, state-ul în care ne aflăm - meniu, level1).

Metodele draw() / drawToScreen() vor desena pe interfața grafică modificările făcute de metoda update().

Metodele keyTyped(), keyPressed(), keyReleased() ne ajută pentru deplasarea personajului și pentru a ne uita prin meniu.

TileMap Package



Generated by UModel

www.altova.com

Background

Clasa are rolul de a furniza background-ul/fundalul pentru joc.

Constructorul are rolul de a încărca o imagine.

Metodele **setPosition()**, **setVector()** au rolul de a asigura un smooth scrolling al camerei.

Tile

Clasa are rolul de a incarca/furniza o dala, care poate sa fie *BLOCKED*, prin care nu trecem, si *BAD*, prin care putem trece.

TileMap

Clasa are rolul de a citi/incarca harta si de a incarca tiles-urile.

Constructorul TileMap ne ajuta sa alegem cate tiles desenam.

Prin **metoda loadTiles** citim imagine tip png, tiles-urile jocului/nivelului, totodata ne ajuta sa le separam in tiles Blocked, prin care nu trecem, si tiles Bad, prin care trecem. Tiles-urile care se afla in partea de sus a imaginii png sunt Bad, iar cele de jos sunt Blocked.

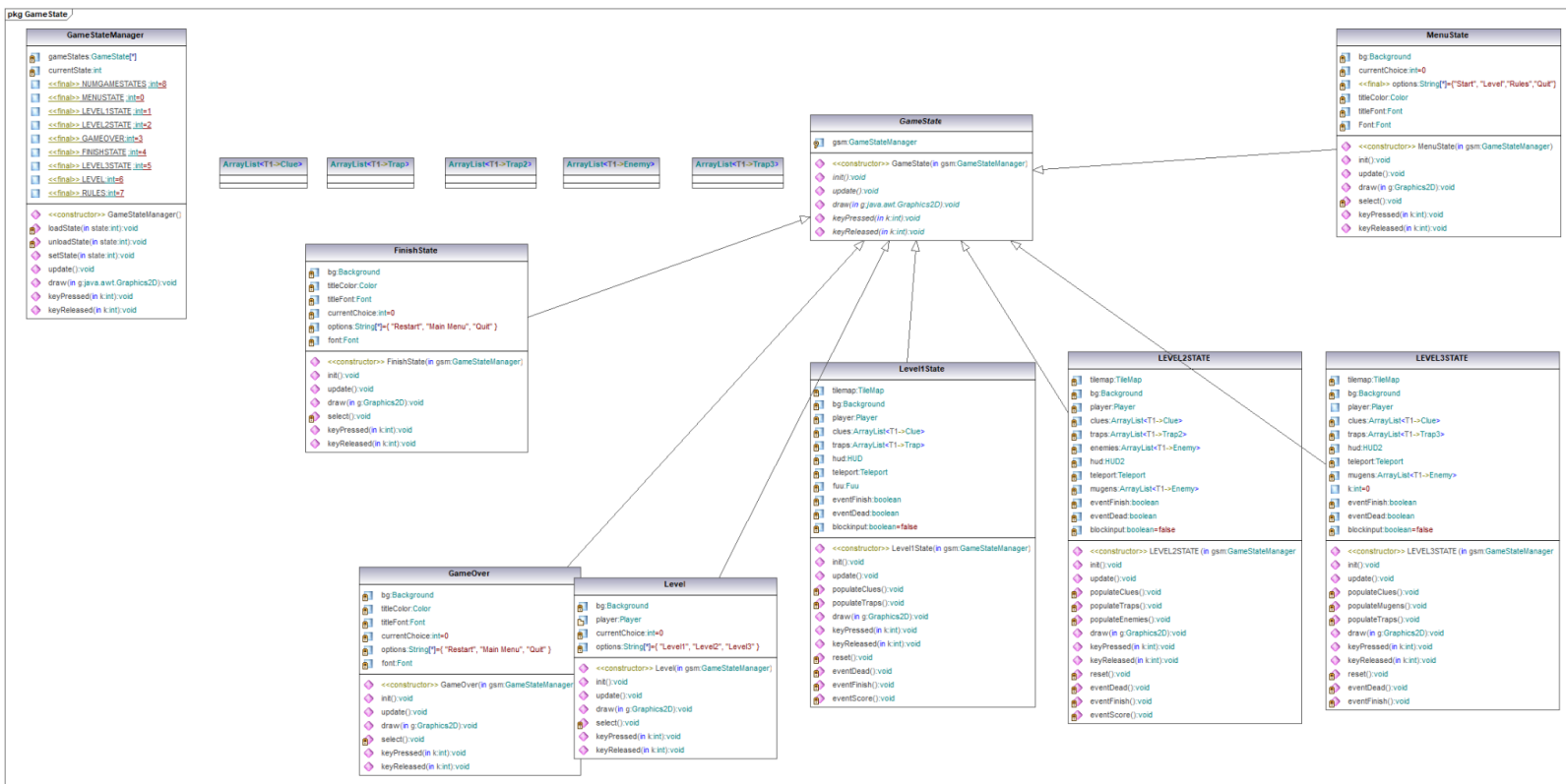
Prin **metoda loadMap** citim fisierul, harta, si o parcurgem.

Metoda getType ne ajuta sa vedem ce tip de tiles este o dala.

Metoda setPosition este pentru a urmari personajul cu camera, cat mai smooth, si pentru vedeam de unde incepem sa desenam harta si ce sa desenam din aceasta.

Metoda fixBounds este folosita pentru a ne asigura ca limitele hartii nu sunt incalcate/trecute.

GameState Package



Generated by UModel

www.altova.com

GameState

Clasa este o clasa abstracta care doar defineste metodele folosite in celelalte clase derivate din ea, inclusiv un constructor.

MenuState

Clasa are rolul de a ne furniza optiunile: de a intra in nivelul 1, de a ne uita la reguli, de a ne alege ce nivel vrem si de quit, in care iesim din joc, inchidem fereastra.

Constructorul MenuState are rolul de a incarca background-ul, de a scrie pe el si de a seta culorile textului.

Metoda select o folosim pentru a putea alege unde dorim sa mergem din meniu.

Metoda keyPressed este pentru a ne deplasa prin meniu/ pentru o selecta optiunea dorita.

FinishState

Clasa are rolul de a iti oferi diferite actiuni dupa ce termini ultimul nivel, de exemplu sa alegi alt nivel, sa te intorci la meniu, sau sa iesi din joc. Aceasta clasa implementeaza aproximativ aceleasi metode ca si clasa MenuState.

GameOver

Clasa iti ofera aceleasi optiuni ca si clasa FinishState, doar ca apare atunci cand mori.

Level

Clasa iti da optiunea de a alege ce nivel doresti sa joci, aceasta clasa fiind folosita la Menu, la GameOver si la FinishState.

RulesState

Clasa este folosita pentru a pune in baza de date modul de deplasare/atac al jucatorului. Totodata este folosita si ca o optiune in Menu.

ScoreState

Clasa este folosita pentru a incarca in baza de date diferite scoruri/colectibile/vieti. Aceasta clasa este folosita da toate niveluri.

Level1State

Clasa are rolul de a crea nivelul 1.

Metoda init incarca tiles, harta, background-ul, creaza jucatorul alegandu-i pozitia, totodata aici folosim functiile populateClues si populateTraps pentru a pune pe harta indiciile si capcanele.

Metoda **update** verifica starea jucatorului, ii calculeaza pozitia, verifica da o luat vreun indiciu sau a atins vreo capcana, caz in care colecteaza sau respectiv, isi pierde o viata.

Metodele **populateClues** si **populateTraps** creaza un vector de obiectele respective si le seteaza pozitii pentru fiecare in parte.

Metoda **draw** are ca scop desenarea/afisarea in fereastra a player-ului, obiectelor, hartii, background-ului, hud-ului (imagine in care ne arata cate viata mai avem, cate indicii am colectat) si a portalului prin care trecem la nivelul urmator.

Metoda **reset** reseteaza pozitia player-ului atunci cand cade in gol.

Metoda **eventDead** te duce la GameOver.

Metoda **eventFinish** face trecerea de la nivelul tocmai terminat la urmatorul, sau in caz de finalizarea jocului te duce la FinishState.

Metoda **eventScore** incarca in baza de date cate vieti i-au mai ramas jucatorului si cate indicii a colectat.

LEVEL2STATE

Clasa are rolul de a crea nivelul 2. Aceasta clasa este aproape la fel fata de Level1State, singura diferenta fiind inamicii.

Functia pentru Inamici se numeste **populateEnemies** in care cream un vector de inamici **Mukuro**, care au doar o viata. In aceasta functie setam pozitia fiecarui inamic de tip **Mukuro**.

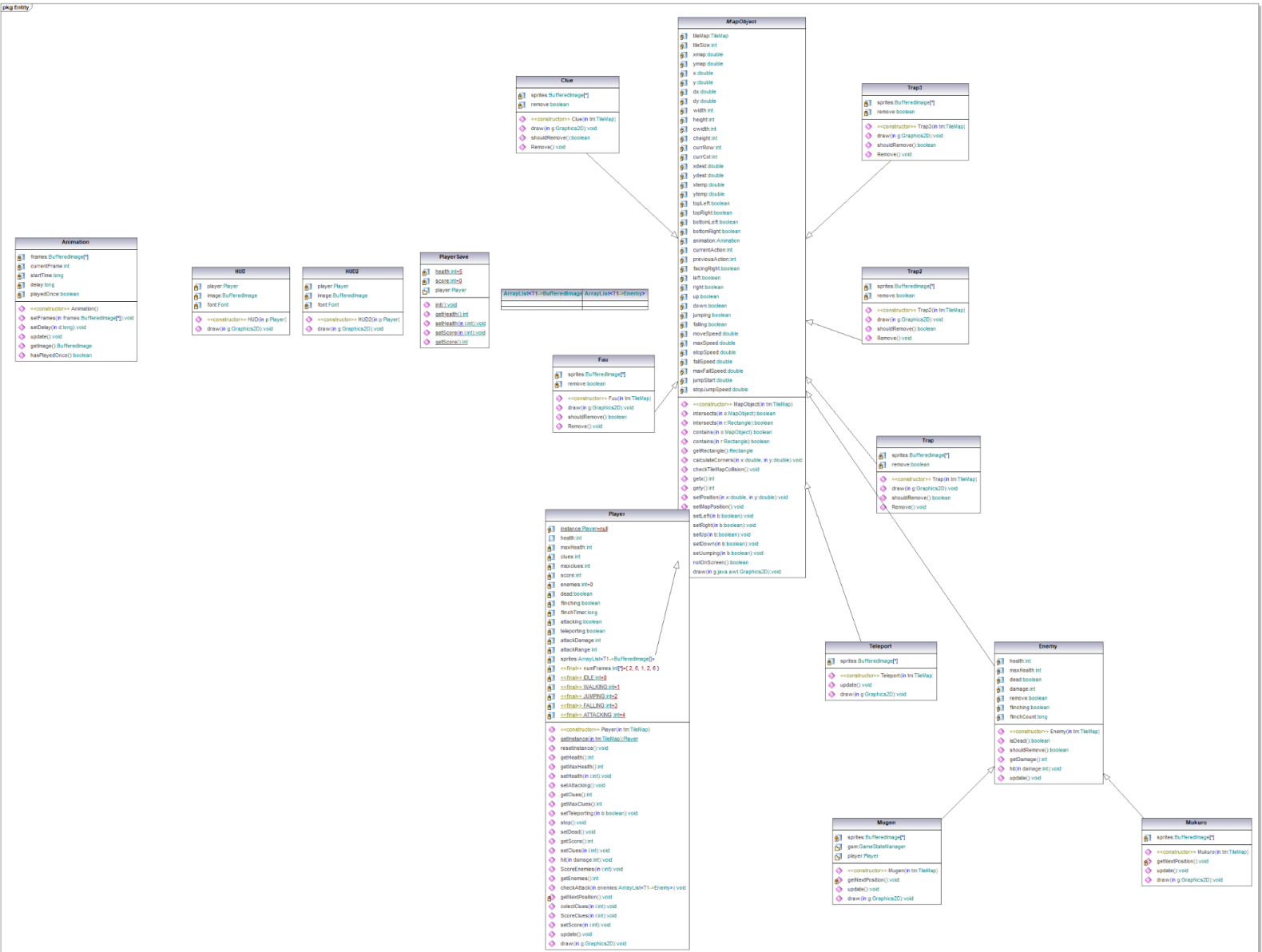
LEVEL3STATE

Clasa are rolul de a crea nivelul 3. Aceasta este aproape la fel ca si LEVEL2STATE, singura diferenta este ca inamicii sunt de tip Mugen, putin mai puternici/rezistenti.

GameStateManager

Clasa are rolul de a ne ajuta sa ne deplasam prin toate state-urile jocului (nivele, Menu, GameOver, FinishState).

Entity Package



Generated by UModel www.altova.com

MapObject

Clasa are rolul de a calcula/rezolva coliziunile cu harta.

Metodele **intersects** sunt folosite la verificarea intersectiei jucatorului cu un inamic.

Metodele **contains** sunt folosite pentru a verifica daca player-ul este intr-un portal (pentru trecerea la urmatorul nivel).

Metoda **calculateCorners** are rolul de a determina coliziunea cu tiles.

Metoda **checkTileMapCollision** are rolul de a ne ajuta sa vedem pozitia/deplasarea obiectului, determinand interactiunea/coliziunea acestuia cu tiles-urile.

Metodele **setPosition** si **setMapPosition** ne ajuta sa setam pozitia obiectului si, respectiv, sa setam unde trebuie desenat obiectul.

Metoda **onScreen** ne spune daca jucatorul este sau un vizibil in fereastra de joc.

Animation

Clasa ne ajuta sa furnizam animatiile realizate cu ajutorul sprites-urilor.

Enemy

Clasa mosteneste clasa MapObject, si este folosita pentru a implementa metodele de verificare daca mai este in viata si daca a fost lovit.

Fuu

Clasa este folosita doar pentru a incarca sprite-urile acesti personaj si pentru a desena.

Clues, Trap, Trap2, Trap3

Sunt clase care mostenesc MapObject si tot ce fac este sa citeasca si sa deseneze imaginile cu indiciile/capcanele specifice fiecarui nivel.

HUD, HUD2

Aceste clase citesc si deseneaza imagini, care se deplaseaza cu player-ul, in care desenam/afisam cate vieti are jucatorul, numarul de indicii colectate, cati inamici a invins eroul.

Teleport

Aceasta clasa citeste si deseneaza o imagine de care ne folosim ca un portal pentru a trece la urmatorul nivel.

Mukuro, Mugen

Ambele clase au rolul de a furniza inamici jucatorului si sunt aproape la fel, singura diferenta fiind viata, adica Mukuro este mai slab, avand o viata, iar Mugen are 2 vieti.

Constructorul, in ambele clase, are rolul de a citi/incarca sprite-urile, dar si pentru a il seta pe o animatie.

Metoda **getNextPosition** o folosim pentru a vedea cum/unde se deplaseaza inamicul.

Metoda **update** o folosim pentru a calcula pozitia, coliziunea cu harta si pentru a isi cunoaste imprejurimile astfel incat sa nu se blocheze sau sa cada in gol.

Player

Aceasta clasa are rolul de a citi/incarca sprite-urile personajului principal, pentru a seta cateva attribute despre jucator, de exemplu attackRange, health sau damage.

In aceasta clasa avem mai multe metode care ne ajuta sa numaram indiciile, sa primim scor daca luam indicii, si sa primim puncte pentru fiecare inamic invins.

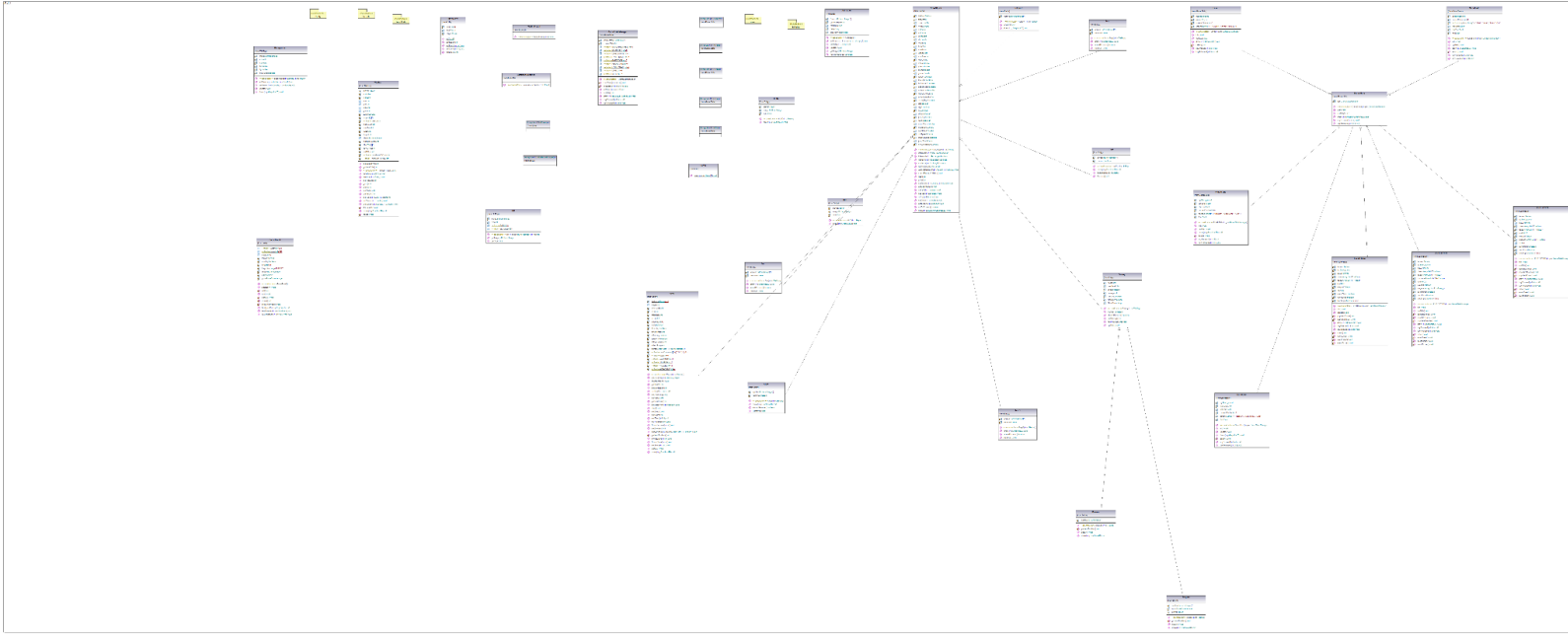
Metoda **checkAttack** este folosita pentru a verifica daca sunte in raza de atac a unui inamic sau daca pur si simplu ne intersectam cu unul, caz in care tot pierdem o viata.

Metoda **getNextPosition** ne ajuta sa aflam urmatoare pozitie a player-ului si sa il facem sa se miste.

Metoda **update** o folosim pentru a actualiza pozitia, verificare coliziuni cu harta/tiles, dar si pentru a seta animatiile in functie de actiuni si de pozitie.

PlayerSave

Aceasta clasa este folosita pentru reinitializarea scorului si vietii jucatorului la trecerea dintre nivele.



Sabloane de proiectare

1.Singleton

Am implementat acest sablon pentru clasa Player, declarand o variabila statica de tip Player care este null.

```
protected static Player instance = null;
```

Am creat o functie **getInstance** in care verificam daca instanta este null, iar in caz ca da instantiam un Player, apelandu-i constructorul.

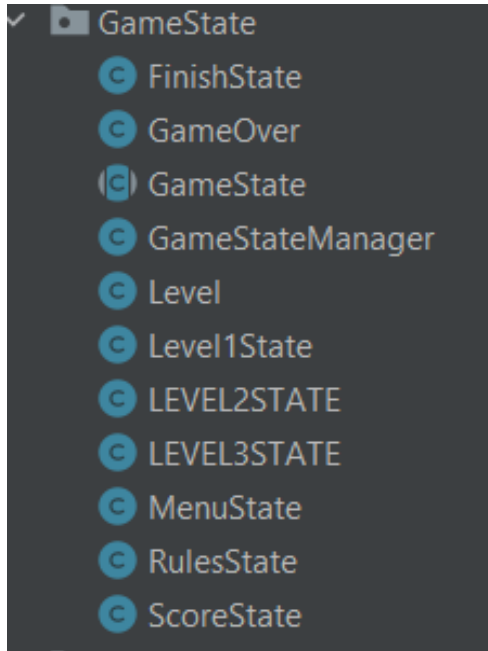
```
public static Player getInstance(TileMap tm){  
    if(instance==null){  
        instance=new Player(tm);  
    }  
    return instance;  
}
```

Dupa, in clasele pentru nivele, la instantierea Player-ului am apelat metoda getInstance.


```
player = player.getInstance(tilemap);  
player.setPosition(x: 80, y: 330);
```

2.State

Am implementat acest sablon prin clasele de tip State.



Sablonul State sugereaza ca programul nostru are un anumit numar de stari in care se poate afla si din care se poate muta repede.

De exemplu din Menu putem intra in Rules, Level1, sau la Level, unde, din nou, puteam intra in diferite stari : Level1 sau Level2 sau Level3. Din GameOver ne reintoarcem la Level sau Menu, la fel si in FinishState.

Bibliografie

<https://opengameart.org/>

<https://letsmakegames.org/resources/art-assets-for-game-developers/>

<https://graphicsgale.com/us/>