

***Chasing freedom***  
Hrițcu Petronela-Adelina  
Grupa 1212B

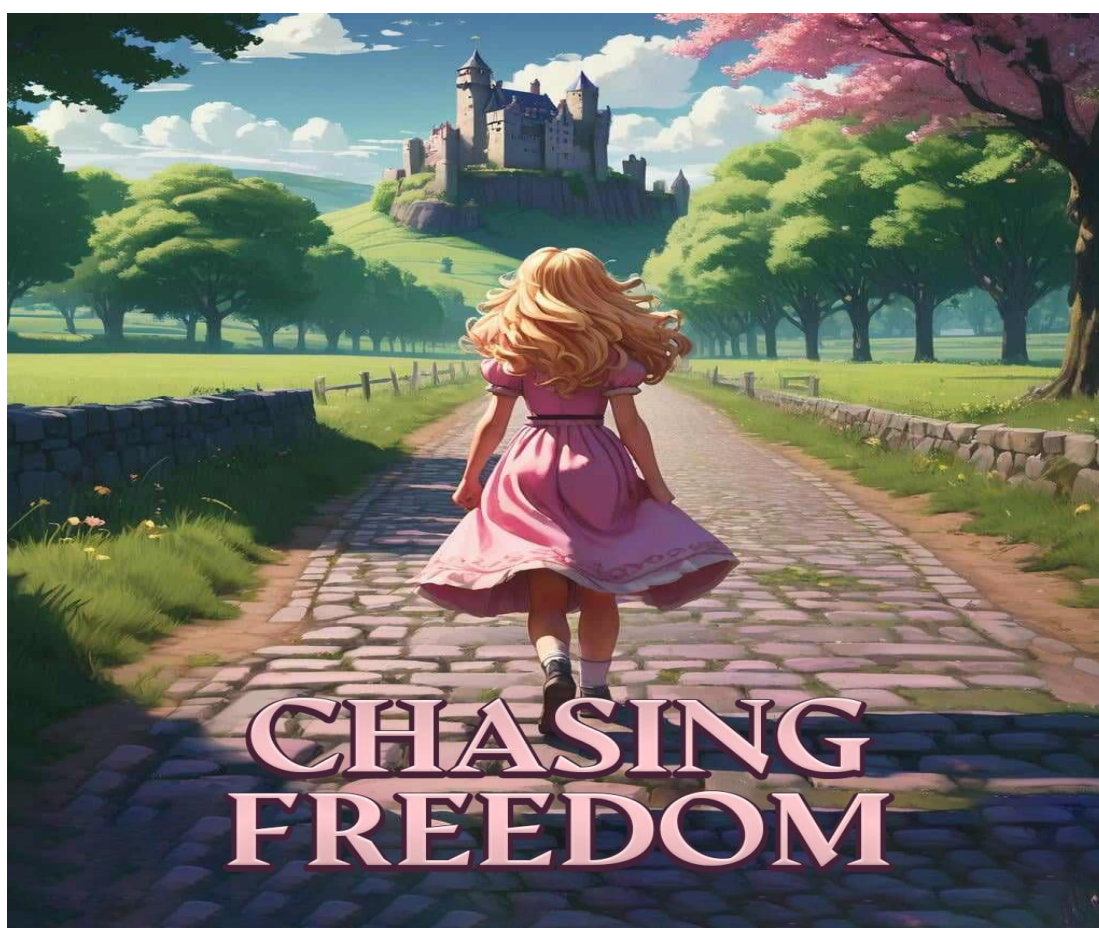
## **Proiectarea contextului**

“Chasing freedom” este un joc de tip Aventură și Acțiune, singleplayer, ce conține atât elemente de competiție ce adaugă un nivel de complexitate jocului, cât și elemente de tip puzzle.

Jocul este situat inițial într-un mic sat pitoresc, unde trăiește o tânără pe nume Ella, alături de mama și cele două surori vitrege. După moartea tatălui ei, familia a căzut în sărăcie și a trebuit să lupte pentru a supraviețui. Mama ei vitregă, Lady Tremaine, o trata pe Ella cu dispreț și o silea să facă toate treburile casnice, în timp ce își răsfăța propriile fiice.

Cu toate acestea, Ella rămâne o fată bună la suflet, plină de speranță și credință că va fi o zi mai bună. Când se anunță un mare bal la palatul prințului, ea visează la o viață mai bună și la posibilitatea de a scăpa de traiul deplorabil în care trăiește deja de ceva timp.

Hotărâtă să participe la bal, fata conștientizează că nu are o ținută adecvată unei asemenea ocazii, ea având o singură rochie ruptă și peticită în garderobă pe care o purta mereu la treburile casnice. În aceste momente, când disperarea i-a cuprins inima, o zână ursitoare a apărut în ajutorul său și i-a oferit o baghetă magică, promițându-i că nimic din cele rele ale lumii nu pot învinge bunătatea și generozitatea. Astfel, așa începe aventura sa în adunarea tuturor pieselor necesare pentru a-și crea o ținută de bal la care mereu a visat, dar și pentru a scăpa de sub robia mamei vitrege.



## **Proiectarea sistemului**

Jocul poate fi jucat folosind tastele W, A, S, D și click-ul stânga al mouse-ului. Pentru deplasarea personajului înainte se va folosi tasta W, pentru deplasarea înapoi tasta S, iar pentru deplasarea la stânga și la dreapta se vor utiliza tastele A, respectiv D. Click-ul din stânga al mouse-ului are rolul de a ataca/activa bagheta magică a Ellei cu care aceasta se va apăra de potențialii inamici pe care îi va întâlni în cadrul îndeplinirii provocărilor (acționarea cu bagheta magică va face inamicul să dispară din joc; 2 acționări vor conduce la terminarea vieții inamicului, implicit și la moartea acestuia).

“Chasing freedom” pornește prin amplasarea personajului în planul de joc (la fiecare nivel fiind amplasat într-un nou plan), acesta având misiunea de a colecta cât mai multe iteme (pentru obținerea unui scor cât mai mare), ele fiind necesare la îndeplinirea misiunii. El trebuie să se apere de inamicii ce îi apar în cale pentru a ajunge la finalul traseului. La finalul fiecărei misiuni (traseu parcurs) se va afla câte un obiect necesar Ellei pentru scopul ei final, iar trecera la următorul nivel va avea loc printr-un portal.

Câștigarea nivelelor 1 și 2, de asemenea fiind și condiția trecerii la următorul nivel, este reprezentată de colectarea a câte 5 obiecte necesare îndeplinirii misiunii Ellei, iar la ultimul nivel, aceasta trebuie să o înfrunte pe mama sa vitregă care, acum, este o vrăjitoare (final boss în joc).

## **Proiectarea conținutului**

### **Definirea personajelor**

Personajul principal.

Ella, o tânără cu un spirit neînfricat, care se ridică cu grație și curaj în fața adversităților vieții. Ea este descrisă ca având părul lung și blond, vălurit, o expresie plină de determinare și un suflet plin de dărnicie.

Personaje auxiliare.

Goblinii, creaturi malefice și înspăimântătoare reprezintă persoanele negative ale jocului, fiind slujitorii loiali ai mamei vitrege ale Ellei, instruiți să o împiedice pe aceasta la îndeplinirea misiunilor, dar mai ales la atingerea scopului ei final.

### **Definirea obiectelor**

Obiectele ce trebuiesc colectate de către jucător pot fi, în funcție de nivelul corespunzător, următoarele:

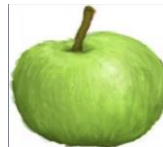
- obiecte necesare la îndeplinirea misiunilor:
  - provizii de mâncare
  - bomboane
  - unelte
- obiecte dobândite după îndeplinirea misiunilor:
  - rochie

- colier
- inel
- obiecte ce măresc sau reduc performanța personajului
  - măr roșu (la colectarea unui astfel de obiect, viteza personajului se mărește)



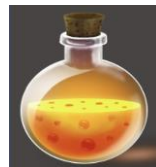
**+2 viteză**

- măr verde (la colectarea unui astfel de obiect i se reduce viteza personajului)



**-2 viteză**

- sticluta cu potiune (la colectarea unui astfel de obiect i se adaugă o viață personajului)



**+1 viață**

### **Definirea interacțiunilor și a parcursului pe care îl poate avea un jucător**

La începutul fiecărui nivel, personajul are 6 vieți. La atacul unui goblin, Ella își pierde câte o viață și dacă până la finalul atacului aceasta nu acționează cu bagheta magică împotriva inamicului, viața ei se termină (moare), iar jocul este încheiat.

În ceea ce privește goblinii, dispariția acestora are loc după acționarea Ellei cu bagheta magică asupra acestora de 2 ori.

## Proiectarea nivelurilor

Jocul are o perspectivă de sus (Top View), oferind jucătorilor o vedere panoramică și detaliată asupra fiecărui nivel. Această perspectivă permite explorarea și navigarea ușoară în lumea creată, oferindu-le jucătorilor un sentiment de control deplin în timpul aventurii lor.

Un element de mister și magie adăugat în joc este reprezentat de portalul magic care facilitează trecerea jucătorului de la un nivel la altul. Acest portal reprezintă o conexiune între lumi și oferă jucătorilor o senzație de călătorie și descoperire pe tot parcursul aventurii lor.

La primul nivel, Ella se găsește într-un labirint, oglindind perioada ei de viață plină de obstacole și greutăți. Misiunea ei este să vină în ajutorul unui croitor sărac, care se chinuie să își hrănească copiii. Tânăra trebuie să caute și să adune alimente pentru o masă hrănitoare pentru cei mici. După ce reușește să îndeplinească cu succes această misiune altruistă, dovedindu-și bunătatea sufletească, fata este răsplătită cu o rochie fermecată, care va completa cu eleganță ținuta ei la balul regal de la palatul prințului.





Pe măsură ce Ella avansează în aventura sa, în al doilea nivel al jocului, acțiunea este amplasată într-o padure labirintică, unde întâlnește o doamnă bătrână cu inima blândă și nepoți dornici de dulciuri. Pentru a obține inelul de care are nevoie pentru a-și completa ținuta de bal, Ella trebuie să colecteze diverse dulciuri pentru nepoții doamnei bătrâne. În timp ce se aventurează prin padure, Ella trebuie să fie atentă la capcanele și obstacolele care îi stau în cale, mai ales la goblinii care o vor ataca când nu se așteaptă.



La ultimul nivel, dificultatea jocului se mărește prin apariția unor inamici mai rapizi, dar și a vrăjitoarei care încearcă să o împidice pe Ella să colecteze ultimul item necesar pentru a merge la bal. Aceasta pășește prin portalul magic pe o plajă, iar destinul o conduce spre un pescar neajutorat, a cărui bărcuță s-a deteriorat din cauza unei furtuni neașteptate. Pescarul disperat îi cere ajutorul, iar Ella își asumă responsabilitatea de a aduna diverse obiecte necesare pentru repararea bărcii. Ea explorează nisipurile insulei, adunând diverse unelte care ar putea fi utile în reparația bărcii. După ce reușește să repare bărcuța pescarului, acesta, recunoscător, îi oferă Ellei un colier, ca semn de mulțumire și ca un ajutor magic în drumul ei spre bal.



## Proiectarea interfeței cu utilizatorul

În imaginea de mai jos, am evidențiat interfața cu utilizatorul în timpul jocului/în timpul unui nivel, având ca elemente principale scorul de joc, reprezentând numărul itemelor colectate, iar în colțul dreapta sunt amplasate numărul vieților personajului.





Interfața ce apare la deschiderea jocului:



Interfața ce apare la câștigarea jocului:



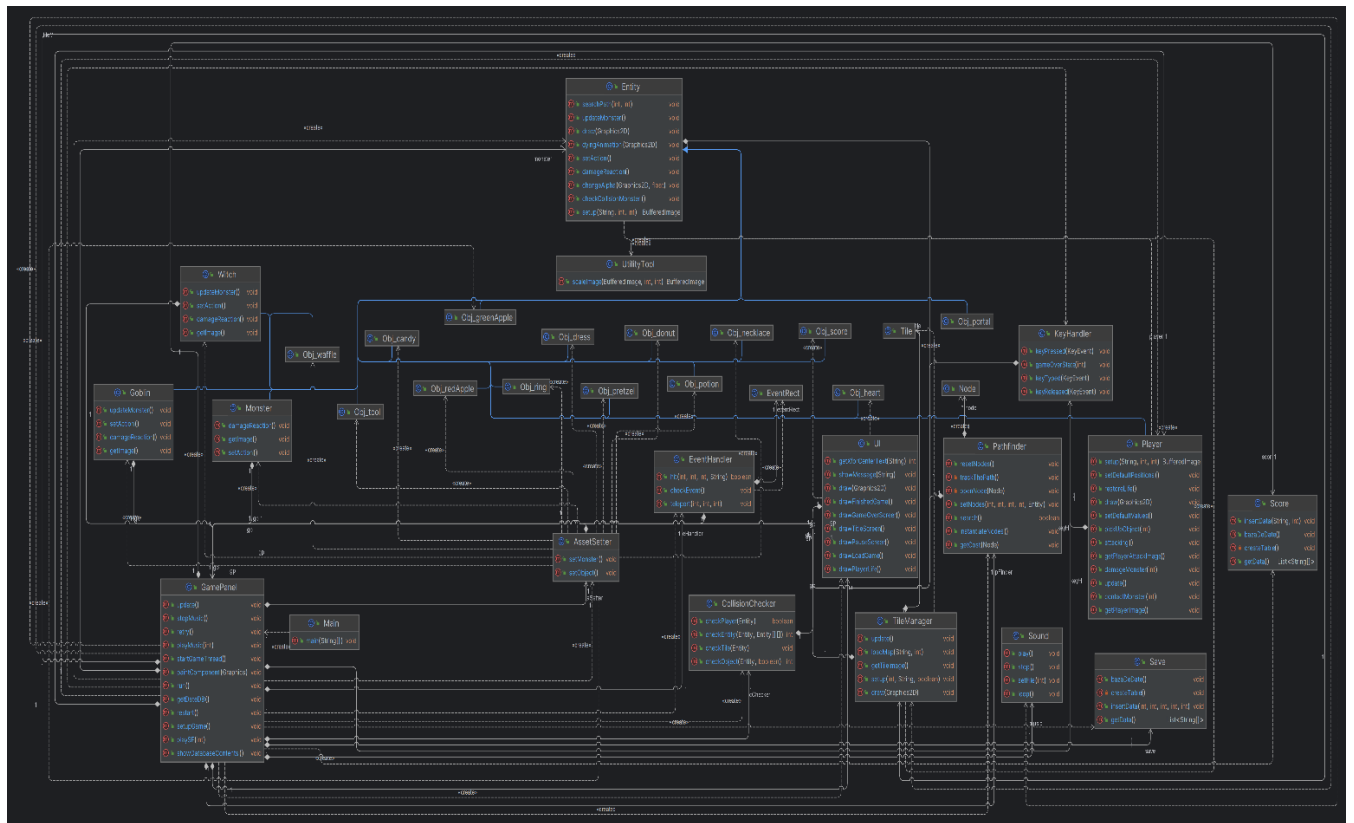
Interfața ce apare la pierderea unui nivel:



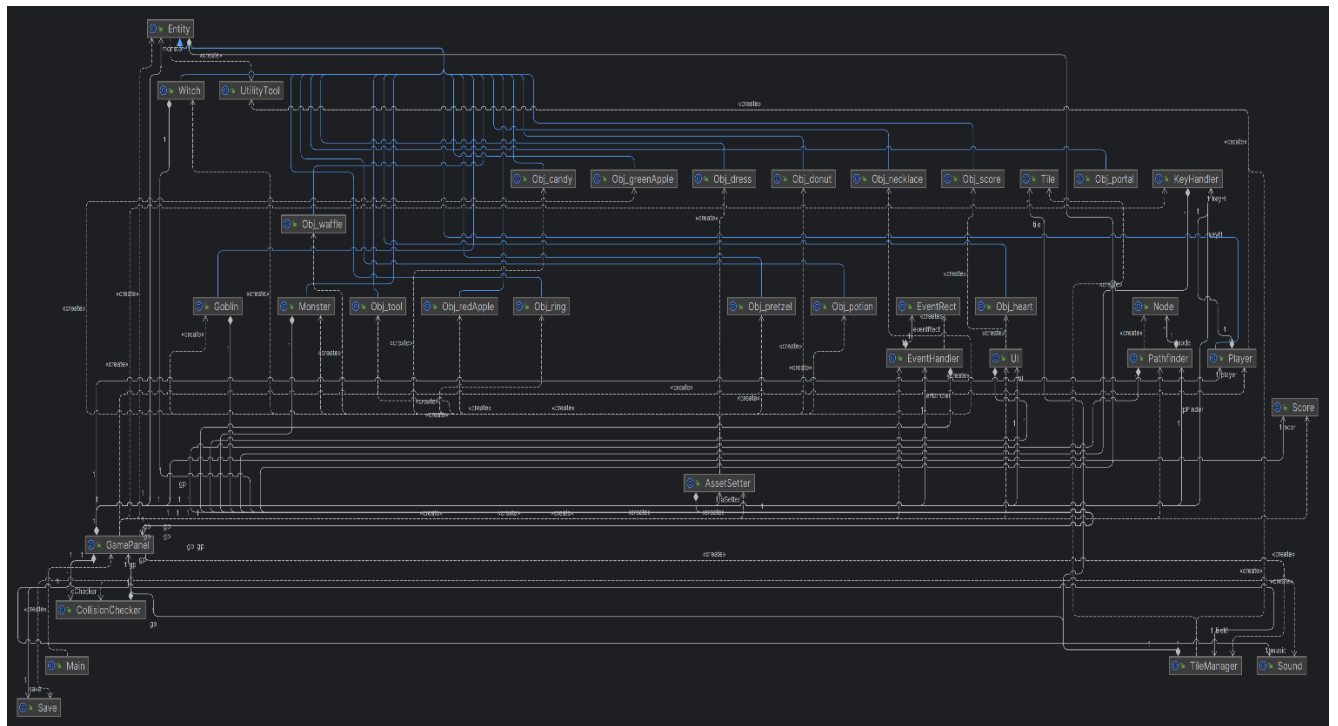


# Arhitectural Design Document

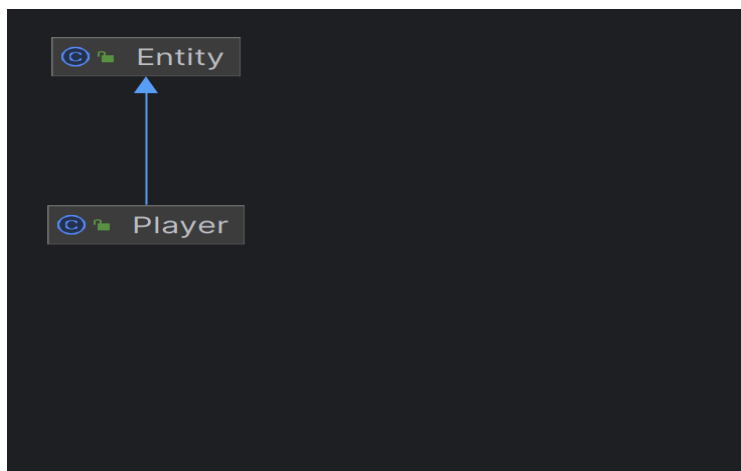
În figura de mai sus se poate observa diagrama de clase UML (sunt vizibile doar constructorii și metodele claselor pentru a nu aglomera diagrama). Fiecare clasa are rolul ei în cadrul proiectului și la rândul său instanțiază/moștenesc alte clase.



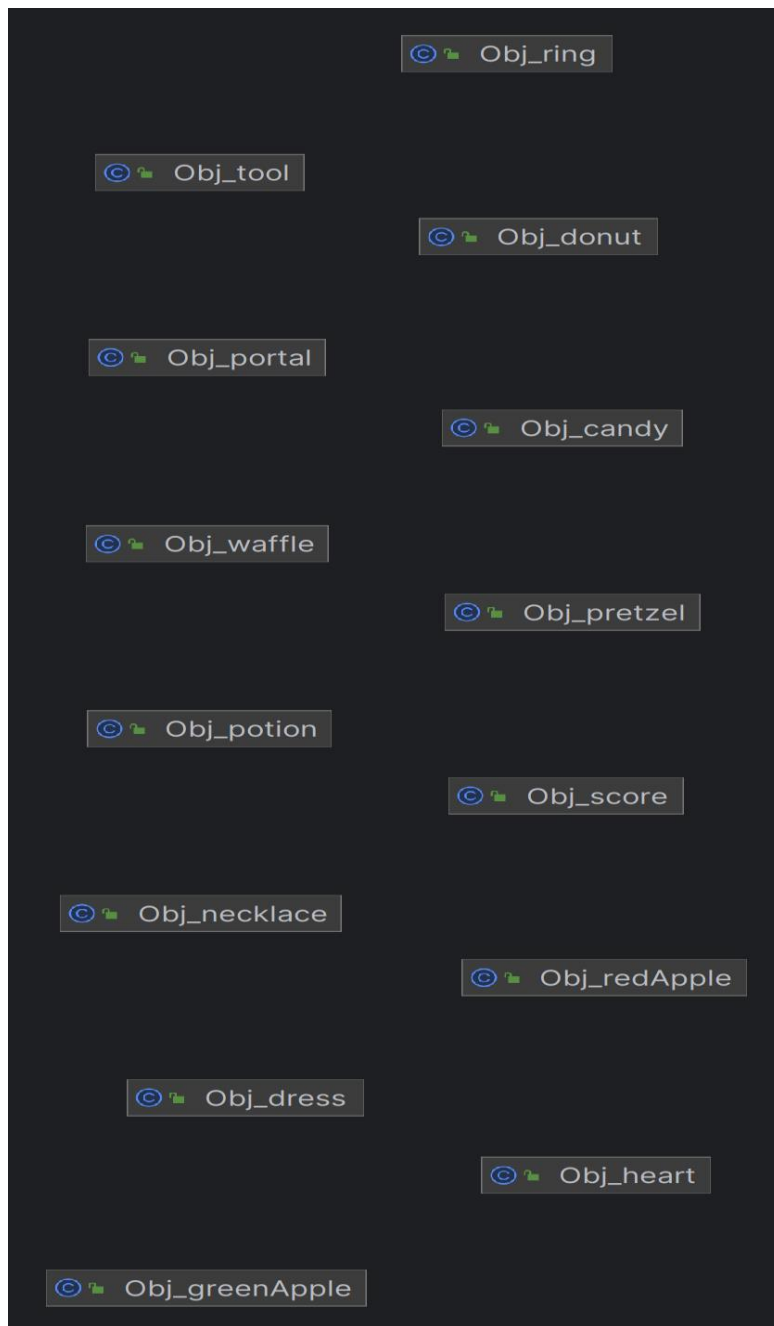
## Diagramă clase



## Diagramă pachet entity

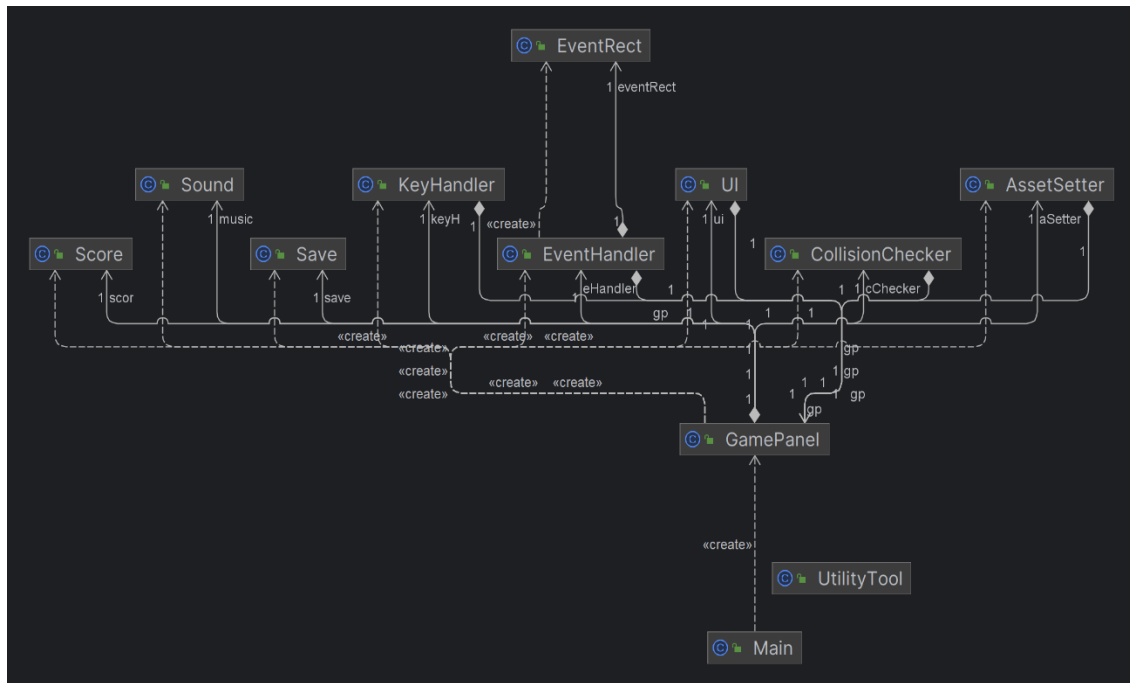


## Diagramă pachet objects

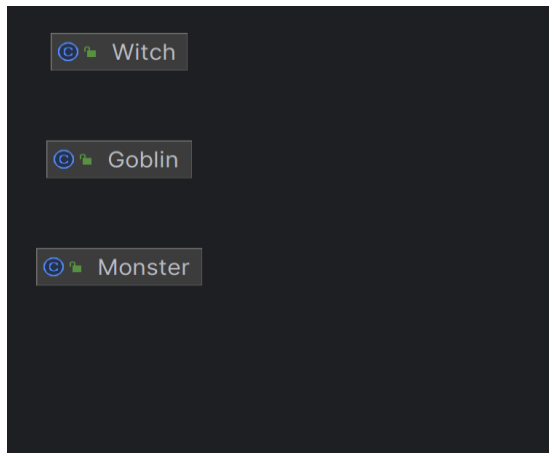




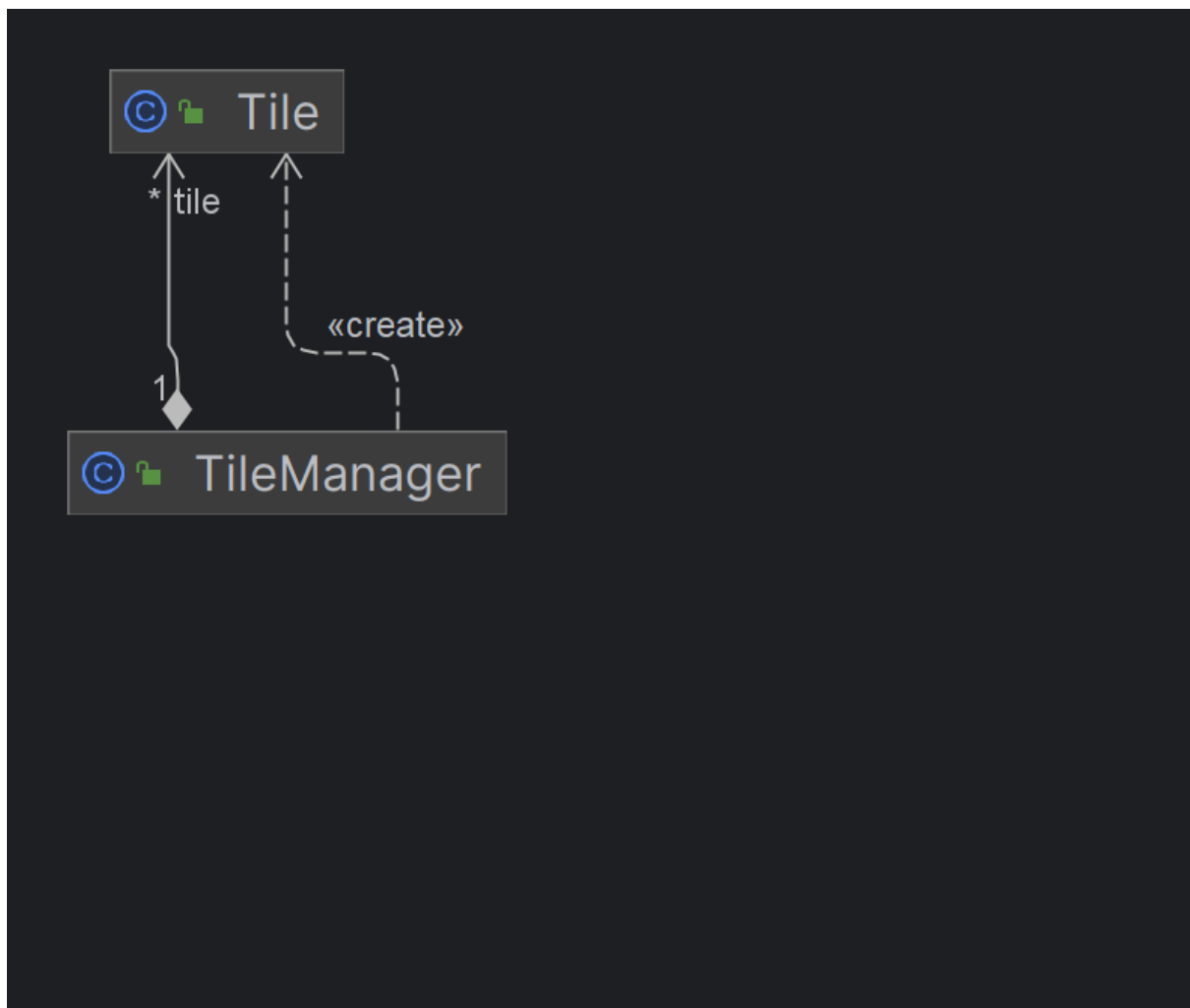
Diagramă pachet main



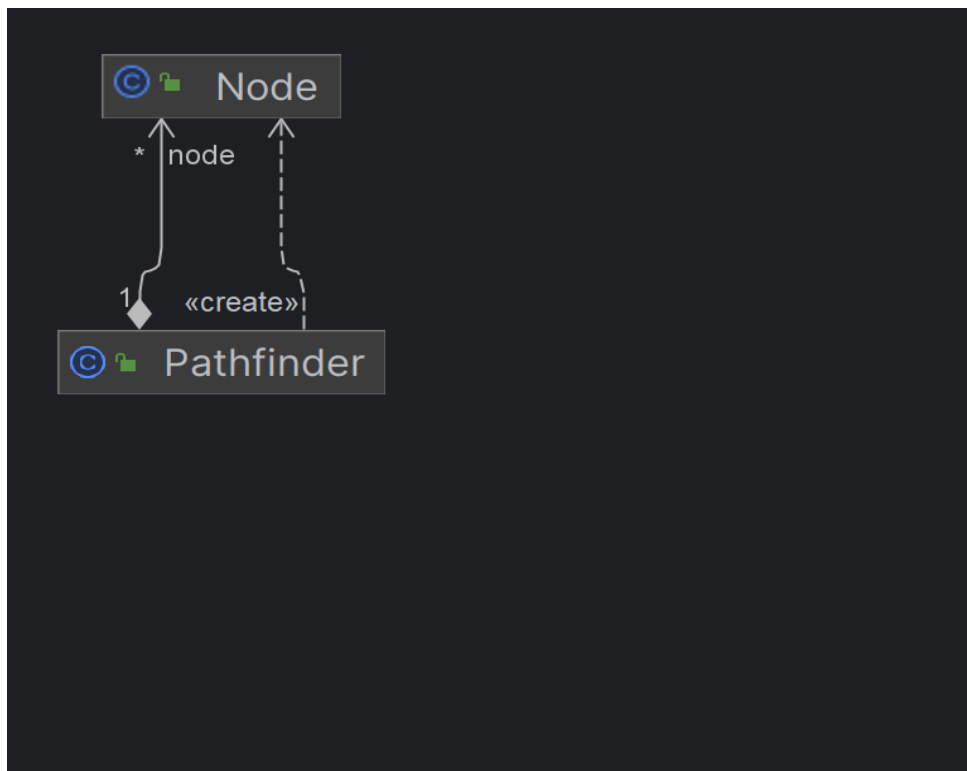
Diagramă pachet monster



Diagramă pachet tile



## Diagramă pachet AI



## Descriere clase proiect

### Main

După cum se poate observa din diagrama prezentată mai sus, această clasă creează o instanță de tipul **GamePanel**, iar mai apoi pornește fluxul activităților prin apelul metodei `startGameThread()` din **GamePanel**.

### GamePanel

Rolul acestei clase este de a face inițializările (de exemplu de a crea instanțe pentru jucători/inamici, pentru a inițializa harta, etc.), dar și de a menține interfața grafică la zi cu ceea ce se întâmplă în joc. Clasa **GamePanel** este o componentă crucială într-un joc, extinzând **JPanel** și implementând interfața **Runnable** pentru a gestiona bucla principală a jocului.

Clasa definește dimensiunile inițiale ale dalelor și scala acestora pentru a calcula dimensiunile ferestrei jocului. Dimensiunile ecranului sunt stabilite pe baza unor valori constante pentru coloane (**maxScreenCol**) și rânduri (**maxScreenRow**), scalate de la mărimea originală a tile-ului.

Această clasă inițializează diferiți manageri și servicii necesare jocului, cum ar fi **TileManager**, **KeyHandler**, și altele pentru manipularea coliziunilor, evenimentelor și obiectelor de joc. De asemenea, sunt definite constante pentru diferite stări de joc (`titleState`, `playState`, `pauseState`). De asemenea, un **Thread** este creat pentru a rula bucla de joc, controlând astfel fluxul principal al jocului.

Metoda **run()** implementează bucla de joc, care este esențială pentru actualizarea stării și redesenarea frame-urilor la un interval controlat pentru a menține un FPS constant.



Metoda **update()** actualizează logica jocului, manipulând starea jucătorului și alte elemente ale jocului în funcție de starea curentă a acestuia.

Metoda **paintComponent()** este responsabilă pentru desenarea componentelor vizuale ale jocului pe ecran. Aceasta verifică starea jocului și desenează diferite ecrane sau entități în funcție de aceasta. Desenează elemente cum ar fi dale, obiecte, jucătorul și interfața utilizatorului, folosind un obiect **Graphics2D** pentru a îmbunătăți performanța grafică.

Metodele **playMusic()** și **playSE()** gestionează muzica de fundal și efectele sonore, demonstrând gestionarea resurselor audio în joc.

Metoda **setupGame()** Inițializează diferite setări ale jocului și poate fi folosită pentru a încărca resurse, seta obiecte sau pregăti starea inițială a jocului.

**GamePanel** este clasa centrală în arhitectura jocului, întrunind inițializarea, actualizarea stării jocului, desenarea grafică, și gestionarea sunetului, asigurând astfel buna funcționare și interactivitatea jocului.

### KeyHandler

Clasa **KeyHandler** este responsabilă pentru gestionarea evenimentelor tastaturii în joc, implementând interfața **KeyListener**.

Metoda **keyPressed(KeyEvent e)** se activează atunci când o tastă este apăsată. Această metodă verifică starea jocului și reacționează în consecință. În starea titlului, detectează tastele W, S și Enter pentru navigare în meniu. În starea de joc, detectează tastele W, S, A, D pentru mișcarea jucătorului și tasta P pentru pauză.

Metoda **keyReleased(KeyEvent e)** se activează atunci când o tastă este eliberată. Această metodă actualizează variabilele de stare ale tastelor pentru a reflecta faptul că o anumită tastă nu mai este apăsată.

Metoda **keyTyped(KeyEvent e)** nu este implementată, deoarece nu este necesară pentru gestionarea evenimentelor tastaturii în joc și este lăsată goală, momentan.

### Tile

Clasa nu definește metode specifice, deoarece este folosită pentru a stoca date despre tile-uri și pentru a le face disponibile în alte clase care se ocupă de gestionarea lor și a coliziunilor acestora.

Obiectele **Tile** sunt create pentru fiecare tile din joc și sunt utilizate pentru a reprezenta și manipula proprietățile tile-urilor în cadrul jocului.

### TileManager

Clasa **TileManager** este responsabilă pentru gestionarea tile-urilor jocului, inclusiv încărcarea, stocarea și desenarea acestora în funcție de harta jocului și poziția jucătorului.

Metoda **getTileImage()** inițializează imaginile dalelor prin apelarea metodei **setup()** pentru fiecare tile cu un indice și calea către imaginea sa asociată.

Metoda **setup(int index, String imagePath, boolean collision)** inițializează un tile cu un anumit indice, imagine asociată și proprietate de coliziune.

Metoda **loadMap(String filePath)** încarcă harta jocului dintr-un fișier text dat prin calea **filePath**.

Metoda **draw(Graphics2D g2)** desenează tile-urile pe ecran în funcție de poziția și vizibilitatea lor relativă față de jucător.

### Entity

Clasa **Entity** reprezintă o entitate din joc (jucătorul) și stochează informații despre acesta, inclusiv poziția sa în lumea jocului, viteză, imagini pentru animații, direcția de

mișcare, coliziuni și alte proprietăți legate de statut. Clasele derivate suprascriu metodele `setAction()` și `damageReaction()` pentru a implementa comportamente specifice, iar `Pathfinder` este utilizat pentru a calcula drumul optim pe care entitatea trebuie să îl urmeze pentru a ajunge la o țintă.

`Entity(GamePanel gp)` este constructorul clasei care inițializează obiectul `Entity` și setează referința către `GamePanel`. Este folosit pentru a crea instanțe ale entității în joc.

`void setAction()` reprezintă o metodă goală destinată a fi suprascrisă în clasele derivate. Este folosită pentru a defini acțiunile specifice ale entității.

`void damageReaction()` este o metodă goală destinată a fi suprascrisă în clasele derivate. Este utilizată pentru a defini reacția entității la primirea de daune.

`void checkCollisionMonster()`. verifică coliziunile entității cu plăcile de joc, obiectele, jucătorul și alte entități. Dacă nu există coliziuni, entitatea se deplasează în direcția setată. Gestionează invincibilitatea temporară după primirea de daune și actualizează animația.

`BufferedImage setup(String imagePath, int width, int height)` încarcă și scalează o imagine dintr-o cale specificată. Utilizează `UtilityTool` pentru a scala imaginea la dimensiunile dorite și returnează imaginea scalată.

`void searchPath(int goalCol, int goalRow)` caută un drum de la poziția curentă a entității către o țintă specificată. Calculează coloana și rândul de start și setează nodurile de start și de țintă în `Pathfinder`. Dacă drumul este găsit, actualizează direcția entității.

`void draw(Graphics2D g2)` desenează entitatea pe ecran calculând coordonatele ecranului pe baza poziției jucătorului. Selectează imaginea corespunzătoare direcției și sprite-ului curent și desenează bara de viață pentru monștri, dacă este cazul. Gestionează efectele vizuale pentru invincibilitate și animația de moarte.

`void dyingAnimation(Graphics2D g2)` gestionează animația de moarte a entității alternează transparența pentru a crea un efect de "clipire". După o serie de clipiri, setează `dying` la `false` și `alive` la `false`.

`void changeAlpha(Graphics2D g2, float alphaValue)` schimbă transparența pentru desenarea grafică. Setează transparența desenării la valoarea specificată utilizând `AlphaComposite`.

`void updateMonster()` actualizează starea monștrilor apelând `setAction` pentru a defini acțiunile acestora. Verifică coliziunile monștrilor utilizând `checkCollisionMonster`.

## Player

Clasa **Player** este responsabilă pentru gestionarea jucătorului din joc și implementează logica asociată mișcării, interacțiunii cu obiectele din joc și afișarea acestuia pe ecran. Metoda **setDefaultValues()** setează valorile implicite ale jucătorului, cum ar fi poziția inițială, viteză și viața maximă.

Metoda **setup(String imageName)** încarcă o imagine din resursele jocului și o redimensionează la dimensiunea dorită.

Metoda **getPlayerImage()** inițializează imaginile pentru animațiile jucătorului.

Metoda **update()** actualizează starea jucătorului în funcție de inputul utilizatorului și de interacțiunile cu obiectele din joc, verifică coliziunile cu tile-urile și obiectele din joc și gestionează mișcarea jucătorului în consecință. Aceasta actualizează totodată și animația jucătorului în funcție de direcția de mișcare.

Metoda **pickUpObject(int i)** gestionează interacțiunea jucătorului cu obiectele din joc (colectarea alimentelor, modificarea vitezei etc.).

Metoda **draw(Graphics2D g2)** desenează jucătorul pe ecran utilizând animația și direcția corespunzătoare.

## **CollisionChecker**

Clasa **CollisionChecker** este responsabilă pentru verificarea coliziunilor dintre entități și obiecte în joc și gestionarea lor.

Metoda **checkTile(Entity entity)** verifică coliziunea entității cu tile-urile hărții, calculează coordonatele colțurilor ariei solide ale entității și determină tile-urile corespunzătoare din harta jocului. Aceasta verifică dacă tile-urile respective au coliziuni și setează flag-ul **collisionOn** al entității în consecință.

Metoda **checkObject(Entity entity, boolean player)** verifică coliziunea entității cu obiectele din joc, iar pentru fiecare obiect din joc care există, calculează coordonatele colțurilor ariei solide a entității și ale obiectului. În funcție de direcția entității și tipul de obiect, setează flag-ul **collisionOn** al entității și returnează indexul obiectului cu care s-a produs coliziunea.

Metoda **checkEntity(Entity entity, Entity[][] target)**

Această metodă este responsabilă pentru verificarea coliziunilor între entități și alte entități din joc. Prin parcurgerea unei matrice de entități, ea compară aria solidă a unei entități date cu ariile solide ale altor entități și detectează coliziunile. Dacă o coliziune este detectată, metoda returnează indexul entității cu care s-a produs coliziunea.

Metoda **checkPlayer(Entity entity)**

Această metodă verifică dacă o entitate dată se intersectează cu jucătorul în joc. Prin compararea ariei solide a entității date cu aria solidă a jucătorului, metoda determină dacă acestea se intersectează. Dacă se produce o intersectare, metoda returnează **true**, altfel returnează **false**. Această metodă este utilizată pentru a detecta coliziunile între entități și jucătorul în timpul jocului.

## **UI**



Clasa **UI** din pachetul `main` este responsabilă de gestionarea interfeței utilizatorului într-un joc, afișând diferite ecrane de stare, cum ar fi ecranul de pauză și ecranul de titlu, precum și de gestionarea afișării vieții jucătorului și a mesajelor. Clasa folosește mai multe imagini pentru a reprezenta starea vieții jucătorului și scorul.

De asemenea, clasa are metode pentru centrarea textului pe ecran, pentru afișarea unor mesaje temporare și pentru desenarea ecranelor specifice în funcție de starea jocului. Constructorul inițializează fonturile și încarcă resursele grafice necesare pentru afișare. Metoda **draw()** este punctul central pentru redarea graficelor UI, care verifică starea jocului și apelează metodele corespunzătoare pentru a desena interfața adecvată. Clasa este strâns cuplată cu clasele de gestionare a jocului, accesând direct proprietățile și metodele obiectului `GamePanel` asociat.

### UtilityTool

Clasa **UtilityTool** este o clasă utilitară care furnizează o metodă pentru redimensionarea imaginilor. Metoda **scaleImage()** primește o imagine și noile dimensiuni dorite și returnează imaginea redimensionată conform acestor dimensiuni. Utilizând un obiect `Graphics2D`, aceasta redimensionează și desenează imaginea originală într-o imagine nouă cu dimensiunile specificate.

### AssetSetter

Clasa **AssetSetter** este responsabilă pentru inițializarea și setarea obiectelor din joc. Aceasta primește o referință către obiectul `GamePanel` și își folosește metoda **setObject()** pentru a atribui obiectele specifice jocului la un array de obiecte.

Metoda **setObject()** instantiază obiectele de tipul specific (cum ar fi waffle, donut, portal etc.) și le atribuie coordonatele în lumea jocului. Aceste coordonate sunt setate manual pentru fiecare obiect.

**Obj\_donut, Obj\_portal, Obj\_waffle, Obj\_greenApple, Obj\_redApple, Obj\_pretzel, Obj\_score, Obj\_heart etc.**

Aceste clase reprezintă subclase ale clasei **Entity** și sunt responsabile pentru definirea caracteristicilor specifice ale obiectelor în joc.

### Clasa Node

Clasa `Node` reprezintă un nod individual într-o grilă, utilizat în algoritmul de pathfinding.

Atribute:

- `Node parent`: Nodul părinte, utilizat pentru a urmări traseul înapoi de la nodul final la nodul de start.
- `int col`: Coloana în care se află nodul în grilă.
- `int row`: Rândul în care se află nodul în grilă.
- `int gCost`: Costul de la nodul de start până la acest nod.
- `int hCost`: Costul estimat de la acest nod până la nodul final (heuristică).
- `int fCost`: Suma dintre `gCost` și `hCost`.
- `boolean solid`: Indică dacă nodul este solid (obstacol) și nu poate fi traversat.
- `boolean open`: Indică dacă nodul este în lista de noduri deschise (de explorat).

- **boolean checked:** Indică dacă nodul a fost deja verificat.

### **Constructor:**

**Node(int col, int row):** Inițializează nodul cu coordonatele coloană și rând specificate.

### **Clasa Pathfinder**

Clasa **Pathfinder** implementează algoritmul de pathfinding (căutare a drumului optim) într-o grilă bidimensională.

- **Pathfinder(GamePanel gp):** Inițializează obiectul **Pathfinder** și creează matricea de noduri.
- **Metode:**
- **void instantiateNodes():** Creează și inițializează nodurile în matricea de noduri pe baza dimensiunilor din **GamePanel**.
- **void resetNodes():** Resetează starea nodurilor și listele de pathfinding pentru a începe o nouă căutare.
- **void setNodes(int startCol, int startRow, int goalCol, int goalRow, Entity entity):** Setează nodurile de start și de final și resetează starea nodurilor. De asemenea, setează starea de coliziune a nodurilor pe baza hărții curente.
- **void getCost(Node node):** Calculează costurile **gCost**, **hCost** și **fCost** pentru un nod dat.
- **boolean search():** Execută algoritmul de pathfinding pentru a găsi drumul de la nodul de start la nodul final. Returnează **true** dacă nodul final a fost atins.
- **void trackThePath():** Urmărește traseul de la nodul final la nodul de start și îl adaugă în **pathList**.
- **void openNode(Node node):** Deschide un nod (îl adaugă în **openList**) dacă nu este deja deschis, verificat sau solid.

### *Legătura dintre clasele Node și Pathfinder*

Clasa **Pathfinder** utilizează instanțe ale clasei **Node** pentru a reprezenta fiecare celulă din grilă și pentru a calcula drumul optim de la un punct de start la un punct de final. Fiecare nod conține informații despre costuri și starea sa (deschis, verificat, solid), care sunt folosite de algoritmul de pathfinding implementat în **Pathfinder**.

Clasa **EventRect** este o subclasă a clasei **Rectangle** din biblioteca **java.awt**, extinzând funcționalitatea acesteia pentru a gestiona evenimentele în joc.

Clasa **EventRect** definește dreptunghiuri care sunt asociate cu diverse evenimente în joc. Acestea sunt utilizate pentru a detecta coliziuni și interacțiuni cu jucătorul sau alte entități din joc. **EventRect** permite gestionarea și urmărirea stării evenimentelor asociate acestor dreptunghiuri pentru a asigura o experiență de joc coerentă și interactivă.

### **Clasa EventHandler**

**EventHandler** permite gestionarea interacțiunilor jucătorului cu elementele de pe hartă, precum și declanșarea acțiunilor în funcție de acestea. Este esențială pentru implementarea mecanicilor de joc care implică evenimente și reacții specifice.

public void **checkEvent()** verifică evenimentele din joc și gestionează acțiunile jucătorului în funcție de acestea.

public void **teleport**(int map, int col, int row) teleportează jucătorul la o altă locație pe hartă.

public boolean **hit**(int map, int col, int row, String reqDirection) verifică dacă jucătorul a lovit un anumit eveniment pe hartă.

## Clasa Save

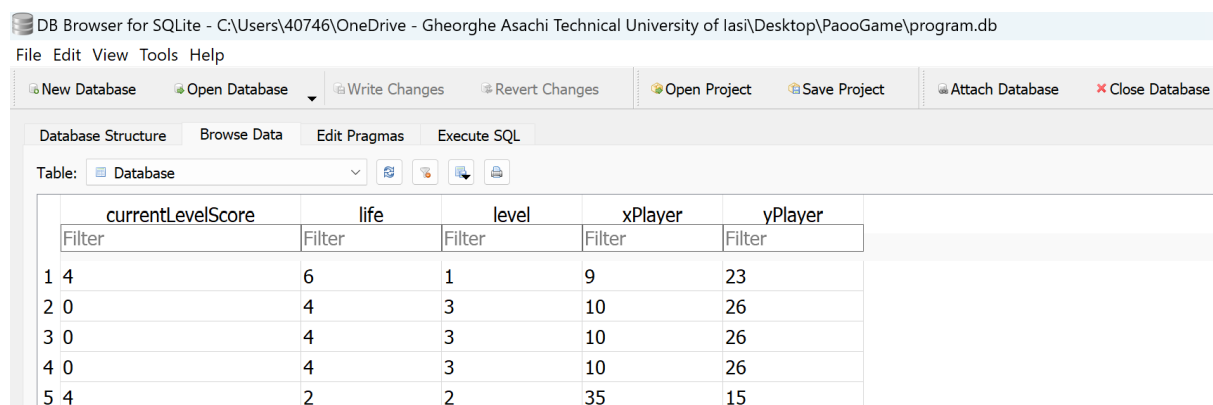
Clasa Save este folosită pentru gestionarea salvării și recuperării datelor de joc într-o bază de date SQLite. Metodele sale permit crearea tabelului necesare pentru stocarea datelor, inserarea de noi înregistrări și recuperarea datelor existente din tabelă.

void **bazaDeDate()** această metodă este responsabilă pentru a iniția crearea tabelului în baza de date. Apelul metodei **createTable()** este efectuat pentru a se asigura că tabela există.

static void **createTable()** creează o conexiune la baza de date și inițializează o declarație SQL pentru a crea o tabelă numită **Database** dacă aceasta nu există deja. Tabela conține coloanele currentLevelScore, life, level, xPlayer, și yPlayer.

static void **insertData**(int currentLevelScore, int life, int level, int xPlayer, int yPlayer) este o metodă ce inserează o nouă înregistrare în tabela Database cu valorile furnizate pentru currentLevelScore, life, level, xPlayer, și yPlayer.

static List<String[]> **getData()** această metodă recuperează toate datele din tabela Database. Deschide o conexiune la baza de date și inițializează o declarație pentru a executa o interogare SQL SELECT \* FROM Database. Datele sunt citite rând cu rând și stocate într-o listă de array-uri de string-uri.



The screenshot shows the DB Browser for SQLite interface. The title bar indicates the file path: C:\Users\40746\OneDrive - Gheorghe Asachi Technical University of Iasi\Desktop\PaooGame\program.db. The menu bar includes File, Edit, View, Tools, and Help. The toolbar contains buttons for New Database, Open Database, Write Changes, Revert Changes, Open Project, Save Project, Attach Database, and Close Database. The main window has tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL. The 'Database Structure' tab is active, showing a table named 'Database'. The table has five columns: currentLevelScore, life, level, xPlayer, and yPlayer. Each column has a 'Filter' button. The table contains five rows of data:

	currentLevelScore	life	level	xPlayer	yPlayer
1	4	6	1	9	23
2	0	4	3	10	26
3	0	4	3	10	26
4	0	4	3	10	26
5	4	2	2	35	15

**Clasa Score** este utilizată pentru gestionarea și stocarea scorurilor obținute în cadrul jocului. Ea oferă metode pentru adăugarea și recuperarea scorurilor din baza



de date SQLite asociată jocului. Această funcționalitate permite păstrarea și afișarea scorurilor într-un mod persistent.

void **bazaDeDate()** inițiază crearea tabelului în baza de date. Apelează metoda **createTable()** pentru a asigura existența tabelului necesare.

private static void **createTable()** creează o conexiune cu baza de date și inițializează o declarație SQL pentru a crea tabela Scor, dacă aceasta nu există deja. Tabela are două coloane: name de tip TEXT și score de tip INT. După creare, conexiunea și declarația sunt închise.

public static void **insertData**(String name, int score) inserează un nou rând în tabela Scor

public static List<String[]> **getData()** recuperează toate datele din tabela Scor. Se deschide o conexiune cu baza de date, se inițializează o declarație pentru a executa o interogare SELECT \* FROM Scor.

Database Contents	
Nume	Scor
player	1
player1	0
player2	5
player3	6
player5	2
player6	0
player7	5
player8	3
player7	1

Anterior, am atașat baza de date în care se salvează scorul final al jucătorului obținut pe parcursul întregului joc.

## Șabloane de proiectare

Clasa AssetSetter urmează în principal șablonul de proiectare Factory Method, având rolul de a crea și plasa obiecte și monștri în joc. Șablonul Factory Method este utilizat pentru a abstractiza și centraliza crearea de obiecte complexe, astfel încât să fie ușor de gestionat și de modificat.

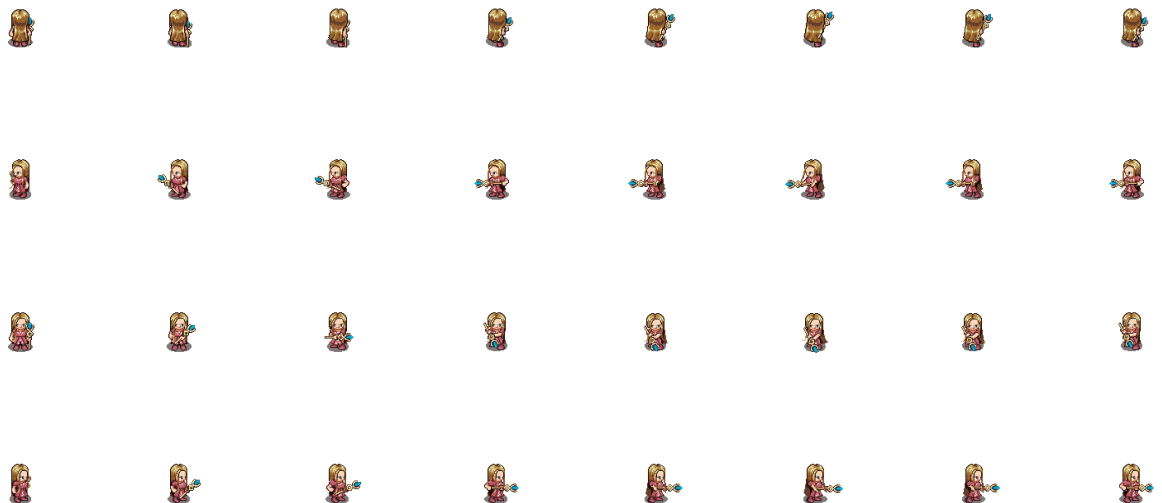
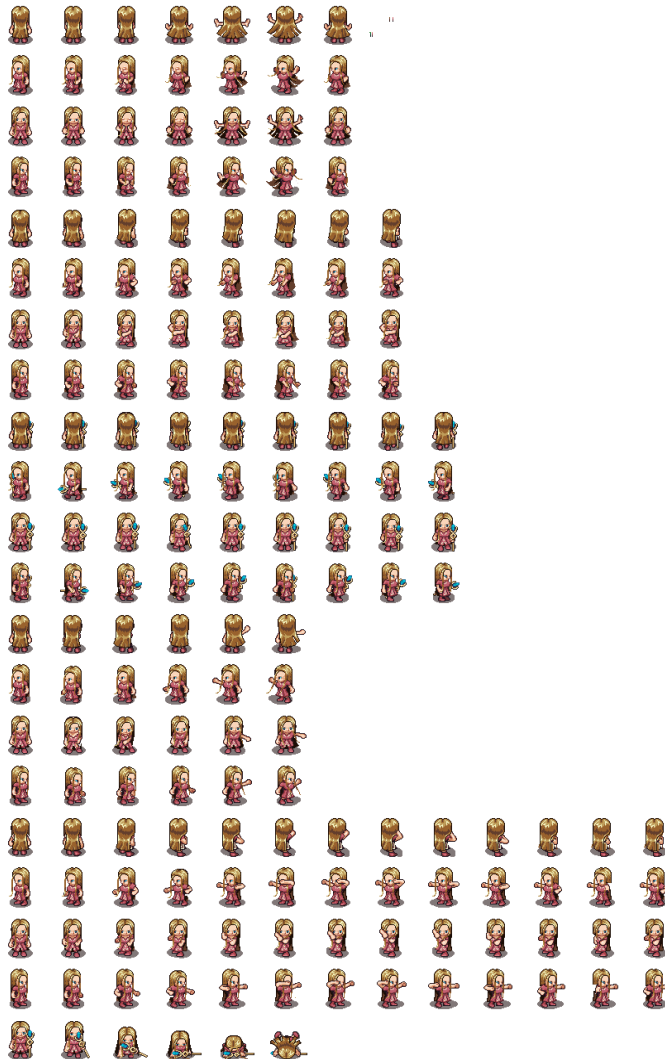
Caracteristici ale șablonului Factory

Centralizarea creării obiectelor. Metodele setObject() și setMonster() se ocupă de instanțierea și plasarea tuturor obiectelor și monștrilor în lume. Acest lucru ajută la centralizarea codului care gestionează aceste creații, facilitând modificările ulterioare și întreținerea.

Abstractizarea logicii de creare. În loc ca fiecare parte a codului să conțină logica necesară pentru a crea obiecte și monștri, AssetSetter preia această responsabilitate. Astfel, codul devine mai modular și mai ușor de citit și înțeles.

## Game sprite-uri ce vor fi utilizate:

- Personajul principal (Ella)





- Goblinii



- Vrajitoarea(mama vitregă)



- Dragonul

Heather Harvey  
cind\_rella@  
hotmail.com

*HH*



## Bibliografie:

1. Image generator: <https://gencraft.com/generate>
2. Game sprite-uri:  
[Universal LPC Sprite Sheet Character Generator \(sanderfrenken.github.io\)](https://opengameart.org/content/lpc-goblin)  
<https://opengameart.org/content/lpc-goblin>  
<https://opengameart.org/content/witch-on-broomstick>