The Mystery of The Lost Key

Romașcu Ștefan

1211B

În drumul său spre casă Fran, un student în anul III, se întâlnește cu un bătrân. Tânărul este oprit de către acesta, iar el nu doar că îi știa numele studentului, ba chiar și facultatea la care învață și materiile la care nu se descurcă. Fran nu apucă să îl întrebe de unde știe atâtea despre viata lui că bătrânul îl întrerupe. Se prezintă drept Geralt și îi spune că dacă este de acord să îl ajute să găsească cheile pe care le-a pierdut, va primii o carte cu toate rezolvările de la examenele ce urmeaza să le susțină. Deși Geralt îl sperie, Fran acceptă oferta și îi cere mai multe informații depsre locul în care au fost pierdute cheile și cum arată. Geralt zâmbește și pleacă.

Dimineața următoare Fran se trezește într-o pădure. Din spatele sau se aud zgomote înfricoșătoare ce par ca se apropie de el.

Jocul incepe intr-o padure cu Fran fiind atacat de catre un Throgg. El realizeaza ca trebuie sa gaseasca cheile de care vorbea Geralt. Acest prim nivel are rolul unui tutorial, avand o dificultate scazuta. Odata cu avansarea in nivel creste si dificultatea jocului. Pe harta se vor gasi cufere care contin chei sau potiuni pentru regenerarea vietii.

Un nivel este terminat daca jucatorul reuseste sa gaseasca tote cheile ascunse pe harta.

Jucatorul este eliminat daca el ramane fara viata. La eliminarea eroului nivelul este resetat si tot progresul facut pierdut.

Pentru a te misca pe harta este necesar un input al jucatorului. Pentru deplasarea stanga/dreapta A/D, pentru a sari SPACE.

La intrarea in joc va aparea un meniu de start ce contine 3 optiuni: NEW GAME, LEADERBOARD si QUIT. Odata ce a inceput jocul exista posibilitatea de a pune pauza prin apasarea tastei ESC. La apsarea acestului buton va aparea un overlay cu 2 optiuni: RESUME si QUIT. Aceasta ultima optiune va trimite jucatorul catre meniul de start al jocului.


NEW GAME
LEADERBOARD
QUIT


GAME PAUSED
MUSIC
SFX

Pe ecran vor exista informatii despre viata jucatorului.

Pe harta sunt 3 tipuri de inamici:

- **Dusk** 

- **Scorch** 

O rasa de catei expusa unei cantitati imense de radiatii. Ei au capatat puterea de a controla focul. Desi rapizi si puternici, Scorch si-au pierdut vederea. In ciuda acestui lucru ei pot sa simta prada. Singura lor slabiciune este apa.

- **Throgg** 

O familie de goblini foarte puternia, dar leanta. Daca esti lovit de ciocanul lor esti eliminat si pierzi jocul. Viziunea lor este scazuta, datorita expunerii la lumina a ochilor.

Nivelul 1: TUTORIAL

Acest nivel are rolul de familiariza jucatorul cu mecanicile jocului si cu modul in care se va desfasura.

Nivelul 2: THE ENTRY

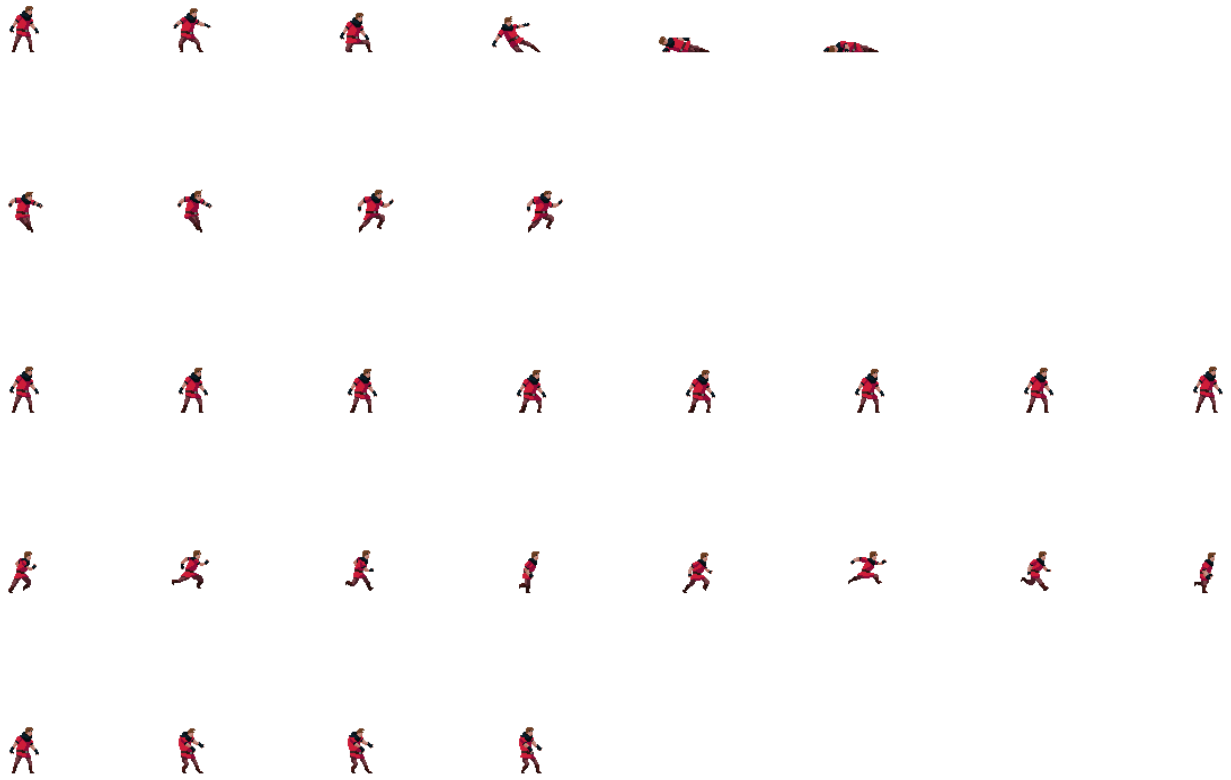
Acest nivel introduce jucatorul cu un nou tip de adversar.

Nivelul 3: THE END?

In acest nivel jucatorul este introdus cu ultimul tip de adversar, Dusk. Dupa finalizarea acestui nivel Fran se trezeste in camera la el.

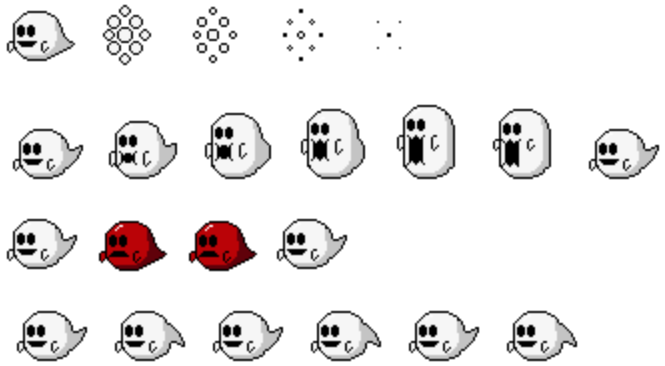
Sprite-uri folosite:

Fran:

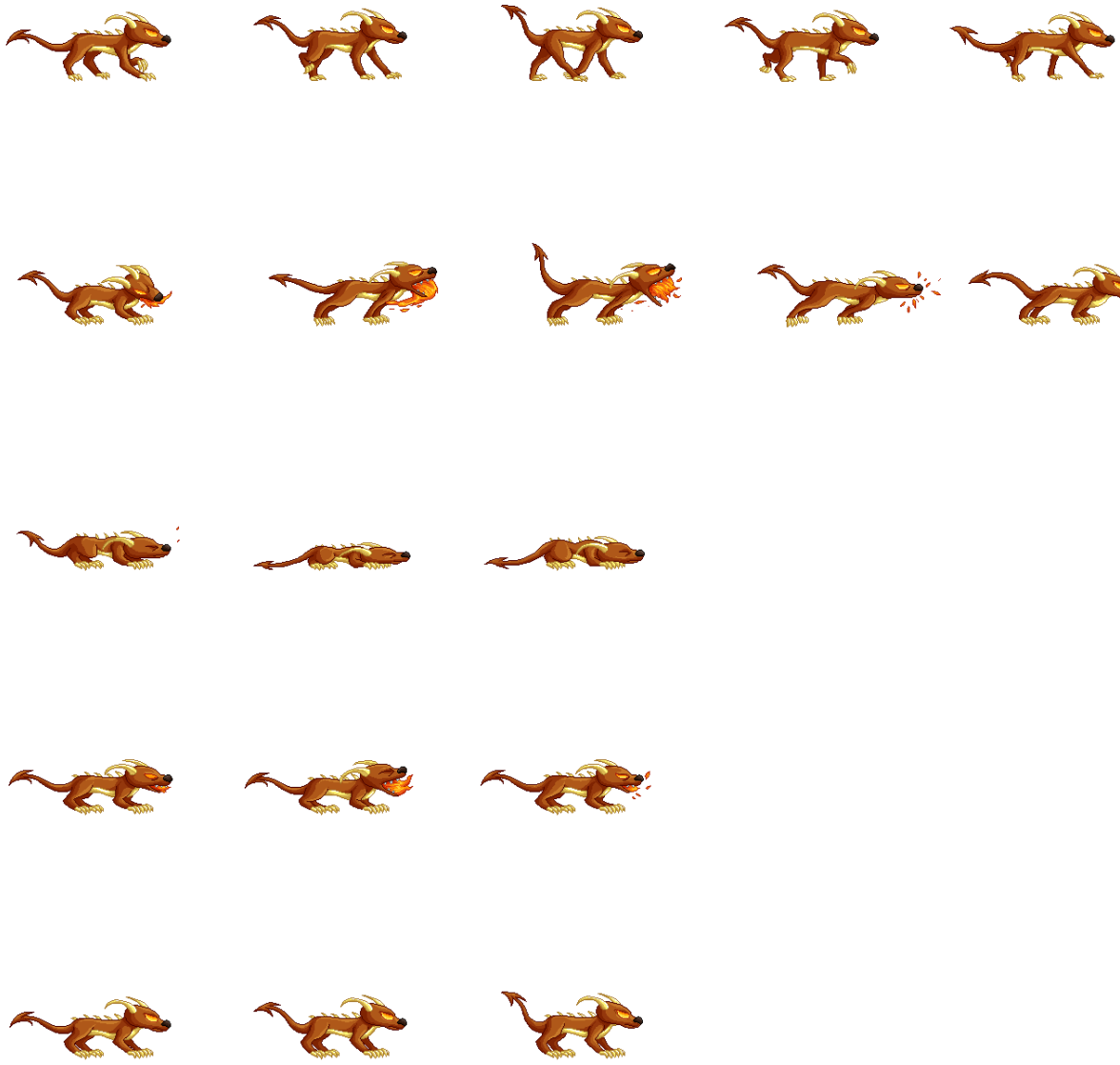


Throgg:

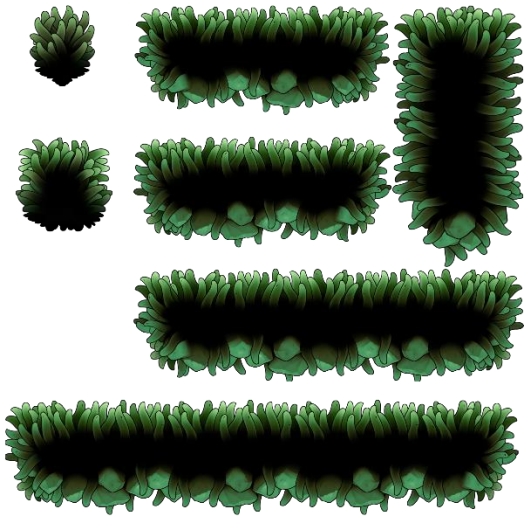
Dusk:

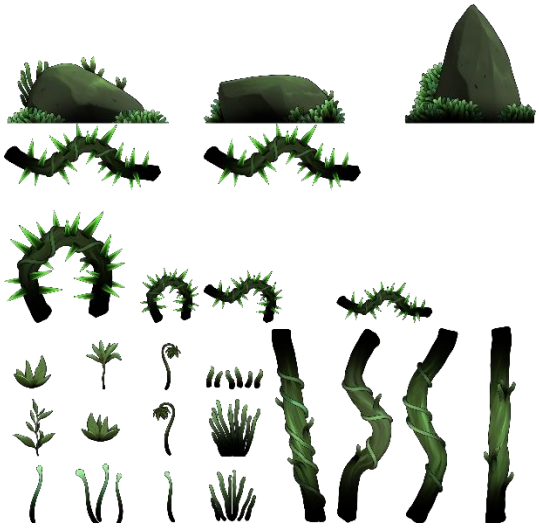


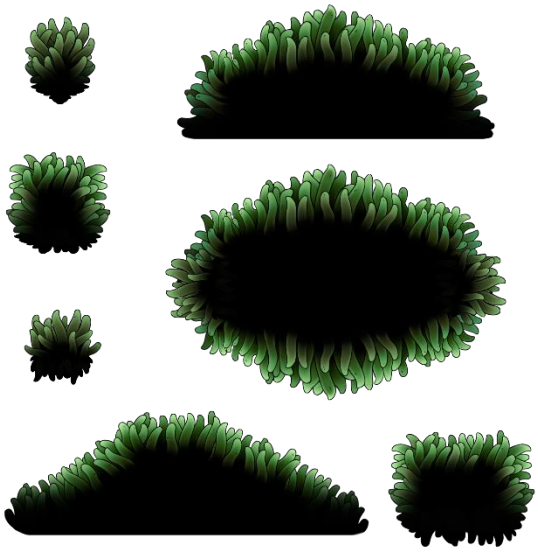
Scorch:



Harta:







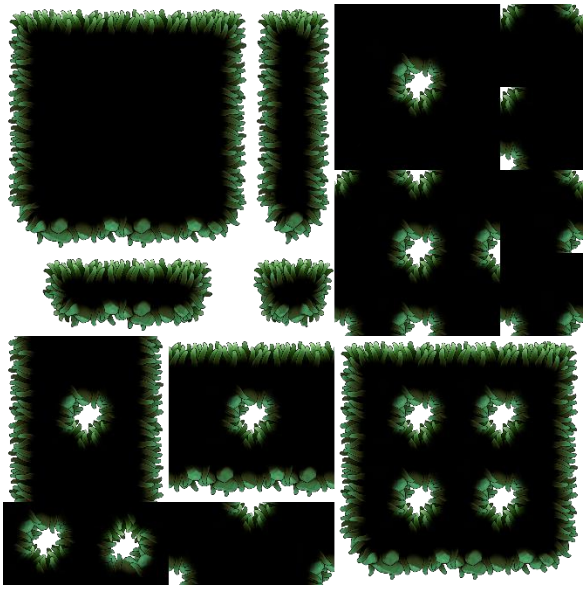
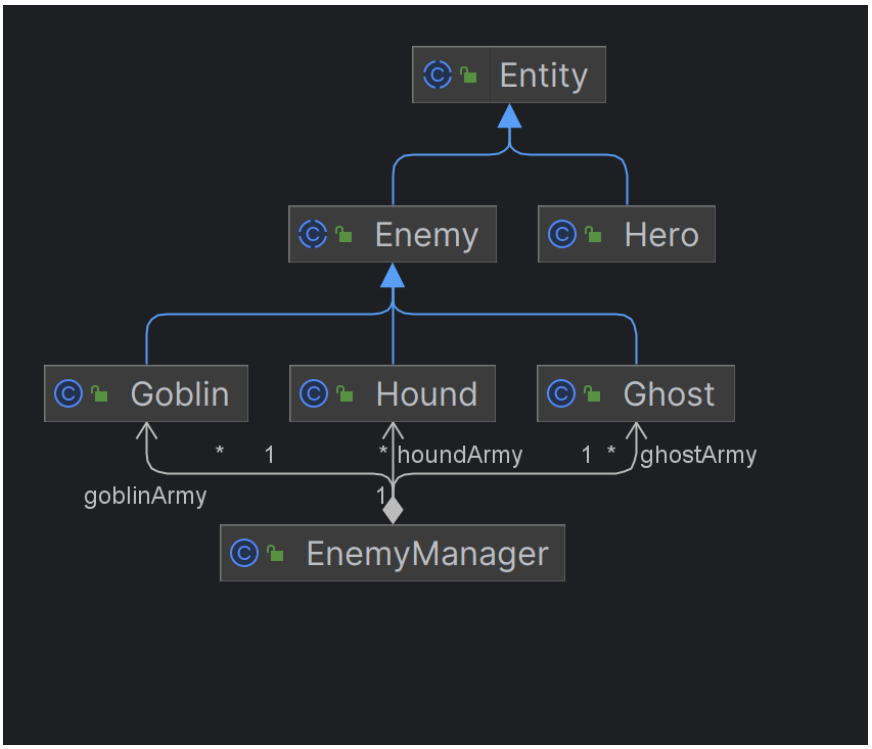
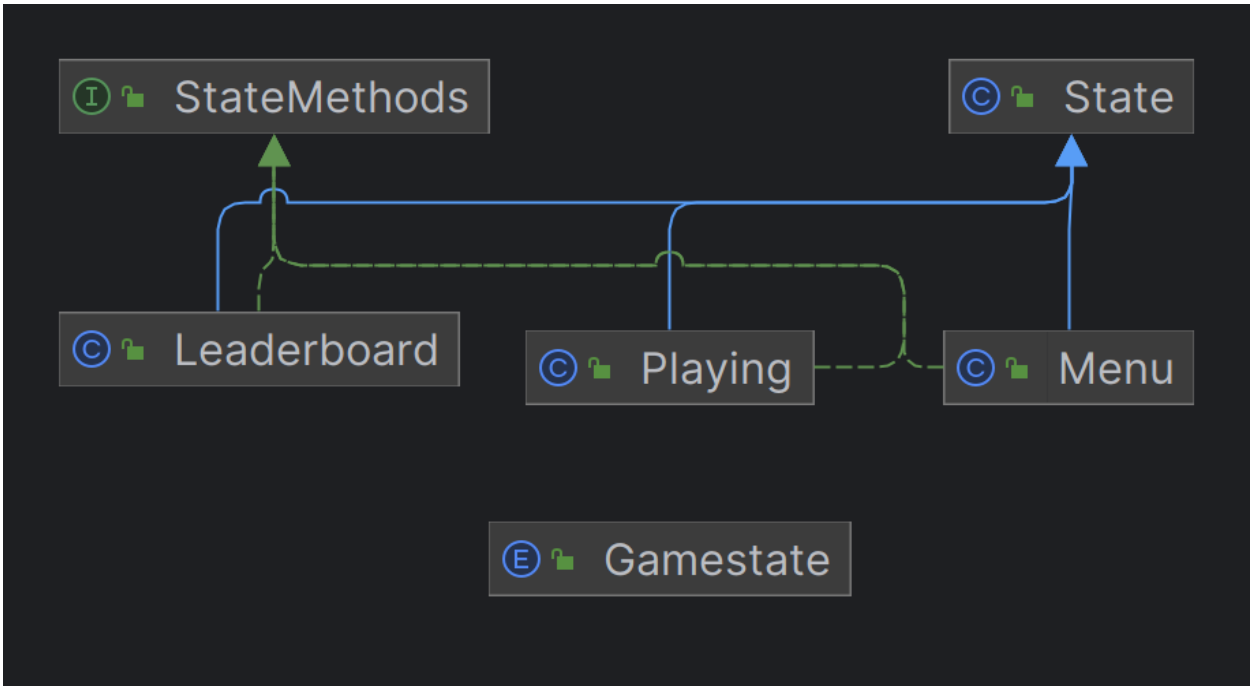


Diagrama UML:

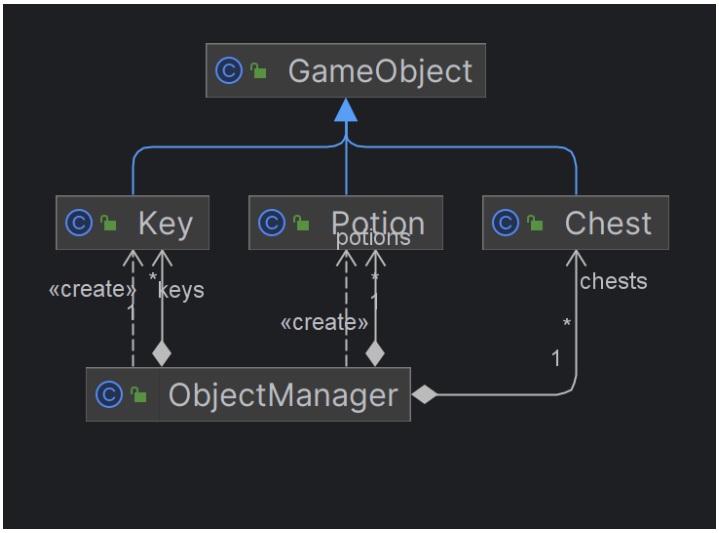
Entity:



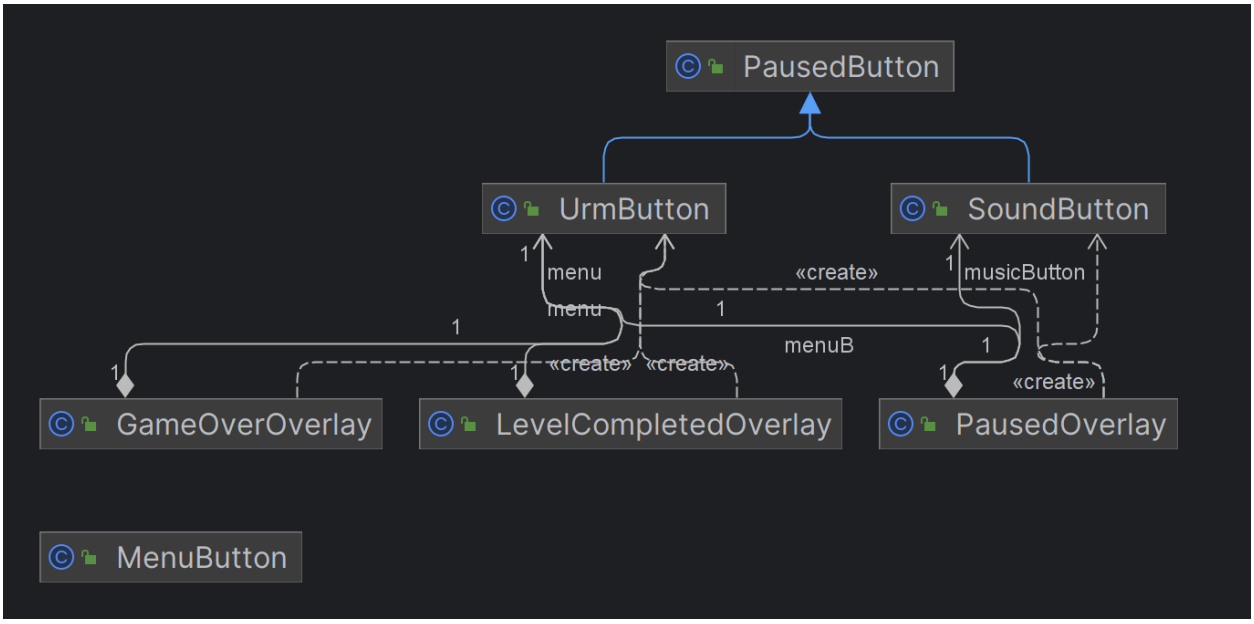
Gamestate:

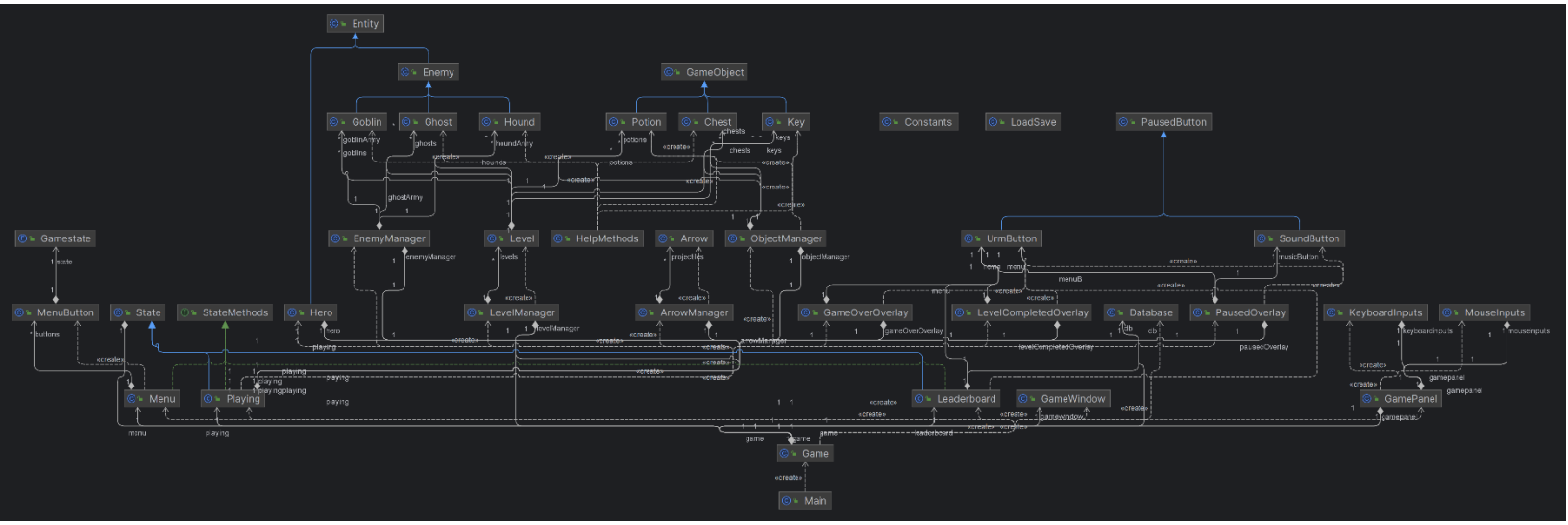


Objects:


GameObject
Key
Potion
Chest
potions
«create»
keys
«create»
chests
ObjectManager

UI:


PausedButton
UrmButton
SoundButton
menu
«create»
musicButton
menu
menuB
«create»
«create»
«create»
GameOverOverlay
LevelCompletedOverlay
PausedOverlay
MenuButton


Entity
Enemy
GameObject
Goblin
Ghost
Hound
Potion
Chest
Key
Constants
LoadSave
PausedButton
Gamestate
EnemyManager
Level
HelpMethods
Arrow
ObjectManager
UrmButton
SoundButton
MenuButton
State
StateMethods
Hero
LevelManager
ArrowManager
GameOverOverlay
LevelCompletedOverlay
Database
PausedOverlay
KeyboardInputs
MouseInputs
Menu
Playing
Leaderboard
GameWindow
GamePanel
Game
Main

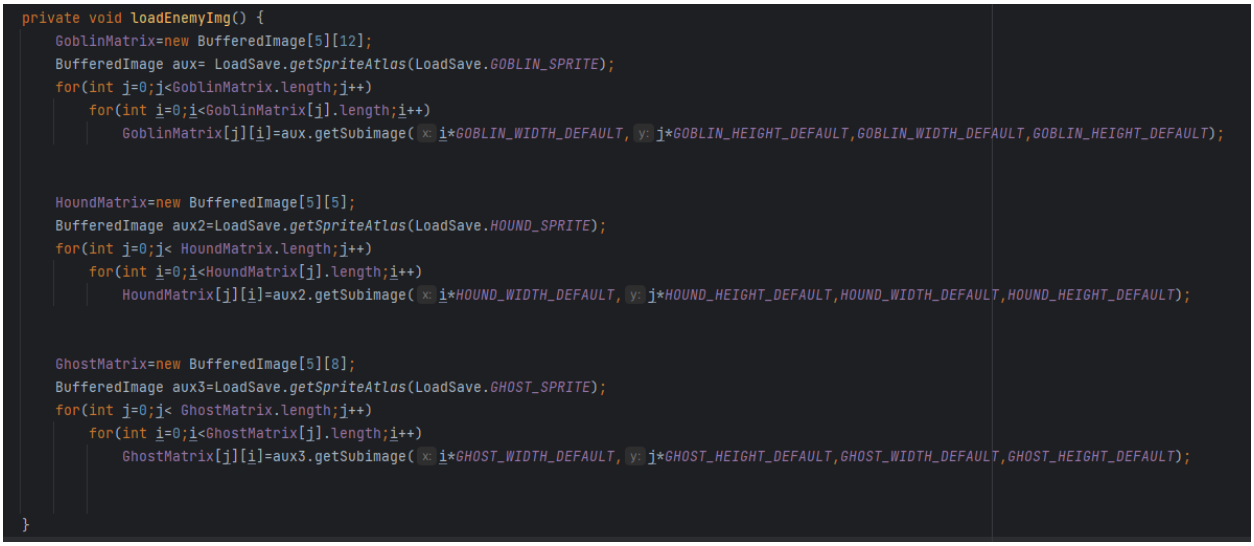
Design patterns:

Flyweight**:**

Idea principală a pattern-ului Flyweight constă în separarea datelor intrinseci ale obiectelor (denumite "stare intrinsecă") de datele extrinseci (denumite "stare extrinsecă"). Starea intrinsecă este compartimentată și partajată între obiectele multiple, în timp ce starea extrinsecă este gestionată individual pentru fiecare obiect.

Acest lucru duce la economisirea resurselor și la reducerea consumului de memorie, deoarece obiectele pot partaja aceleași date intrinseci în loc să le reproducă pentru fiecare obiect în parte. În schimb, fiecare obiect conține doar datele extrinseci unice care îl diferențiază de celelalte obiecte.

Astfel, clasele specifice inamicilor(Goblin,Hound,Ghost) nu retin spriteul specific fiecarui tip de inamic. Spriteul fiecarui inamic este retinut in interiorul unui EnemyManager, iar acesta se ocupa de incarcarea imaginilor in memorie.



```
private void loadEnemyImg() {
    GoblinMatrix=new BufferedImage[5][12];
    BufferedImage aux= LoadSave.getSpriteAtlas(LoadSave.GOBLIN_SPRITE);
    for(int j=0;j<GoblinMatrix.length;j++)
        for(int i=0;i<GoblinMatrix[j].length;i++)
            GoblinMatrix[j][i]=aux.getSubimage( x: i*GOBLIN_WIDTH_DEFAULT, y: j*GOBLIN_HEIGHT_DEFAULT,GOBLIN_WIDTH_DEFAULT,GOBLIN_HEIGHT_DEFAULT);


    HoundMatrix=new BufferedImage[5][5];
    BufferedImage aux2=LoadSave.getSpriteAtlas(LoadSave.HOUND_SPRITE);
    for(int j=0;j< HoundMatrix.length;j++)
        for(int i=0;i<HoundMatrix[j].length;i++)
            HoundMatrix[j][i]=aux2.getSubimage( x: i*HOUND_WIDTH_DEFAULT, y: j*HOUND_HEIGHT_DEFAULT,HOUND_WIDTH_DEFAULT,HOUND_HEIGHT_DEFAULT);


    GhostMatrix=new BufferedImage[5][8];
    BufferedImage aux3=LoadSave.getSpriteAtlas(LoadSave.GHOST_SPRITE);
    for(int j=0;j< GhostMatrix.length;j++)
        for(int i=0;i<GhostMatrix[j].length;i++)
            GhostMatrix[j][i]=aux3.getSubimage( x: i*GHOST_WIDTH_DEFAULT, y: j*GHOST_HEIGHT_DEFAULT,GHOST_WIDTH_DEFAULT,GHOST_HEIGHT_DEFAULT);


}
```

State:

State este un model de proiectare care permite obiectelor să își schimbe comportamentul în funcție de starea internă în care se află.

În cadrul acestui pattern, un obiect poate avea mai multe stări diferite, iar comportamentul său poate varia în funcție de starea în care se află. În loc să implementăm toate aceste comportamente într-o singură clasă mare și complexă, le împărțim în clase separate pentru fiecare stare. Fiecare clasă de stare definește comportamentul specific al obiectului în respectiva stare.

Astfel, in cod exista o interfata StateMethods care defineste metodele obligatorii pentru implementarea unui GameState. In porgram exista 4 state-uri:PLAYING, MENU, LEADERBOARD, QUIT, salvate intr-un enum. Fiecare state are o clasa specifica in care se descrie comportamentul jocului in functie de starea in care se afla.

MouseInputs:

```
    STEFAN +1 *
14  @Override
15  public void mouseClicked(MouseEvent e) {
16      switch (Gamestate.state){
17          case PLAYING:
18              gamepanel.getGame().getPlaying().mouseClicked(e);
19              break;
20          case LEADERBOARD:
21              gamepanel.getGame().getLeaderboard().mouseClicked(e);
22              break;
23      }
24  }
25
    STEFAN +1
26  @Override
27  public void mousePressed(MouseEvent e) {
28      switch (Gamestate.state){
29          case MENU:
30              gamepanel.getGame().getMenu().mousePressed(e);
31              break;
32          case PLAYING:
33              gamepanel.getGame().getPlaying().mousePressed(e);
34              break;
35          case LEADERBOARD:
36              gamepanel.getGame().getLeaderboard().mousePressed(e);
37              break;
38      }
39  }
40
    STEFAN +1
41  @Override
42  public void mouseReleased(MouseEvent e) {
43      switch (Gamestate.state){
44          case MENU:
45              gamepanel.getGame().getMenu().mouseReleased(e);
46              break;
47          case PLAYING:
48              gamepanel.getGame().getPlaying().mouseReleased(e);
49              break;
50          case LEADERBOARD:
```

Game:

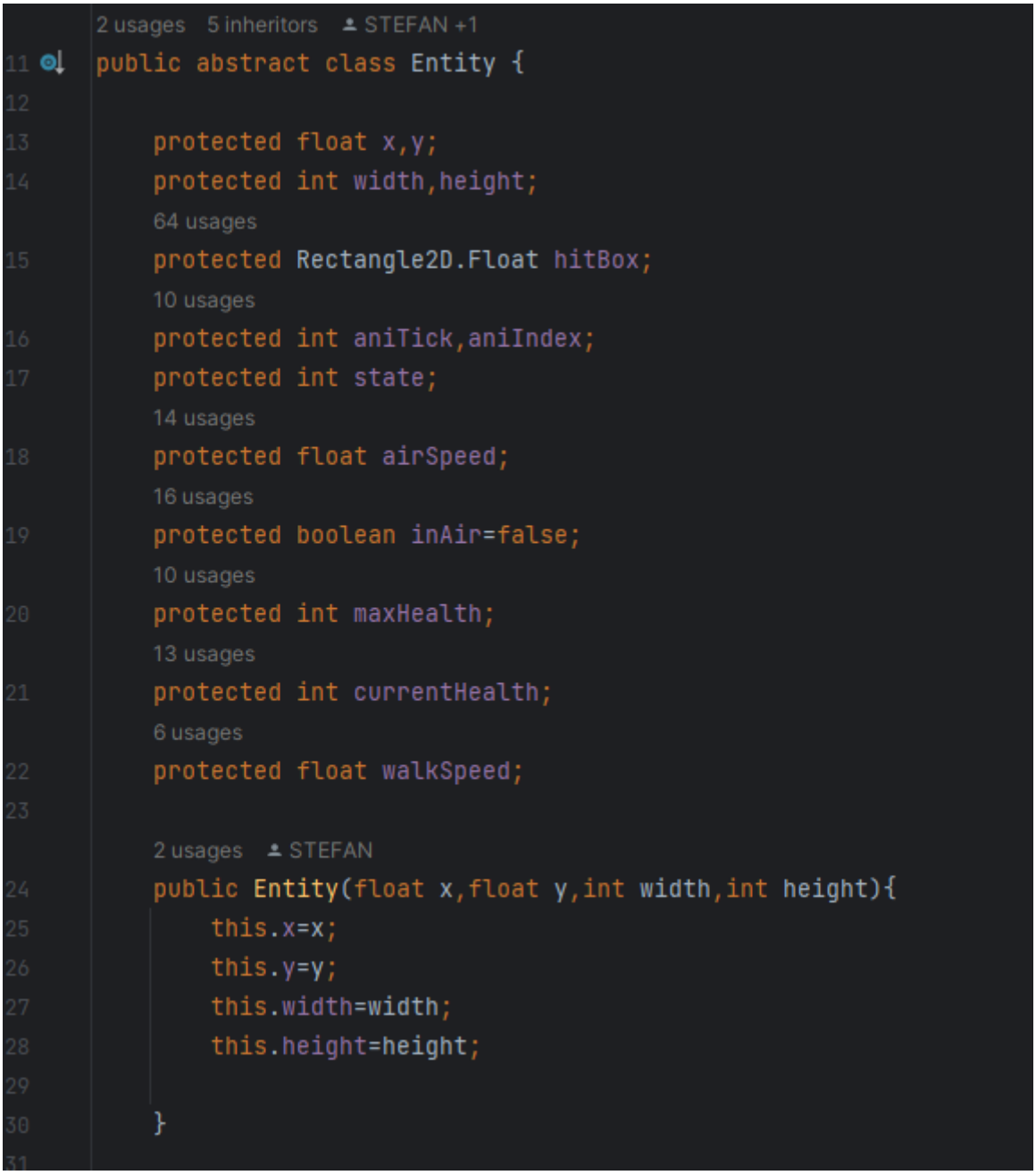
```
64      public void update() {
65
66          switch (Gamestate.state){
67              case MENU:
68                  score=0;
69                  menu.update();
70                  break;
71              case PLAYING:
72                  if(name==null){
73                      getName();
74                  }
75                  playing.update();
76                  break;
77              case LEADERBOARD:
78                  leaderboard.update();
79                  break;
80              case QUIT:
81                  closeDB();
82                  System.exit( status: 0);
83          }
84      }
85
        1 usage   STEFAN +1
86      public void render(Graphics g){
87          switch (Gamestate.state){
88              case MENU:
89                  menu.draw(g);
90                  break;
91              case PLAYING:
92                  playing.draw(g);
93                  break;
94              case LEADERBOARD:
95                  leaderboard.draw(g);
96
97          }
98
99      }
```

Template:

Template este un model de proiectare care definește o structură de bază pentru o anumită operație, permițând subclaselor să redefinească anumite pași specifici.

Acest pattern se bazează pe conceptul de metoda șablon, unde o clasă de bază (numită clasă șablon) conține o metodă principală care definește algoritmul general și utilizează metode auxiliare care pot fi suprascrise de către subclase pentru a oferi implementări specifice.

Astfel, in interiorul package-ului ENTITY exista o clasa de baza pentru toate clasele: Entity. Aceasta clasa contine date specifice fiecarei entitati: x,y,viata,viteza,etc. In pus, la baza fiecarui inamic sta o clasa numita Enemy care mosteneste aceasta clasa abstracta. In enemy se afla metode specifice unui inamic: turnToPlayer, canSeePlayer,etc.



```
2 usages  5 inheritors  STEFAN +1
public abstract class Entity {

    protected float x,y;
    protected int width,height;
    64 usages
    protected Rectangle2D.Float hitBox;
    10 usages
    protected int aniTick,aniIndex;
    protected int state;
    14 usages
    protected float airSpeed;
    16 usages
    protected boolean inAir=false;
    10 usages
    protected int maxHealth;
    13 usages
    protected int currentHealth;
    6 usages
    protected float walkSpeed;

    2 usages  STEFAN
    public Entity(float x,float y,int width,int height){
        this.x=x;
        this.y=y;
        this.width=width;
        this.height=height;

    }

}
```

```java
3 usages  3 inheritors  STEFAN +1
public abstract class Enemy extends Entity{

    4 usages
    protected int enemyType;
    4 usages
    protected int aniSpeed=25;
    5 usages
    protected boolean firstUpdate=true;


    14 usages
    protected int walkDir=RIGHT;
    3 usages
    protected int tileY;
    4 usages
    protected float attackRange=1.75f*Game.TILE_SIZE;

    3 usages
    protected boolean active=true;
    7 usages
    protected boolean attackCheck;



    3 usages  STEFAN +1
    public Enemy(float x, float y, int width, int height,int enemyType) {
        super(x, y, width, height);
        this.enemyType=enemyType;

        maxHealth=getMaxHealth(enemyType);
        currentHealth=maxHealth;
        walkSpeed=0.25f * Game.SCALE;
    }
```

```java
public class Ghost extends Enemy{
    8 usages
    private Rectangle2D.Float attackBox;
    2 usages
    private int attackBoxOffsetX;
    2 usages
    private int attackBoxOffsetY;
    1 usage  Stefan
    public Ghost(float x, float y) {
        super(x, y, GHOST_WIDTH, GHOST_HEIGHT, GHOST);
        aniSpeed=19;
        walkDir=LEFT;
        initHitBox( width: 30, height: 20);
        initAttackBox();
    }
```

Baza de date:

Jocul salveaza cel mai bun scor al jucatorilor. Primii 3 ajung in leaderboard. Scorul este reprezentat de numarul de secunde necesar terminarii nivelelor. In baza de date exista un tabel LEADERBOARD in care se salveaza numele jucatorului(Name) si scorul(Score).

In cod exista o clasa Database. Ea retine conexiunea intr-o variabila privata connection. Aceasta clasa are un constructor in care se creeaza conexiunea cu baza de date, o metoda pentru a inchide aceasta conexiune. Metoda saveScoreToDatabase salveaza numele si scorul jucatorului in baza de date. In plus, se verifica daca userul exista deja, caz in care se compara scorul actual cu cel salvat in baza de date. Daca o obtinut un scor mai bun scorul va fi actualizat. Metoda bestScore returneaza un vector de String care contine pe fiecare element un jucator. Acesti jucatori sunt jucatorii care au obtinut cele mai bune 3 scoruri.

```
public class Database {
    6 usages
    private Connection connection;
    1 usage  Stefan
    public Database()
    {
        connection = null;
        try {
            Class.forName( className: "org.sqlite.JDBC");
            connection = DriverManager.getConnection( url: "jdbc:sqlite:database.db");
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit( status: 0);
        }
        System.out.println("Opened database successfully");
    }
```

```
    1 usage  Stefan
    public void close(){
        try {
            if (connection != null) {
                connection.close();
                System.out.println("Database connection closed.");
            }
        } catch (SQLException e) {
            System.err.println("Failed to close the database connection.");
            e.printStackTrace();
        }
    }
```

```
1 usage  Stefan
public void saveScoreToDatabase(String playerName, int score) {
    try {
        Statement statement = connection.createStatement();
        String query = "SELECT Score FROM LEADERBOARD WHERE Name= '"+playerName+"'";
        ResultSet resultSet = statement.executeQuery(query);

        if (resultSet.next()) {
            int existingScore = resultSet.getInt( columnLabel: "Score");

            if (score < existingScore) {
                String updateQuery = "UPDATE LEADERBOARD SET Score = " + score + " WHERE Name = '" + playerName + "'";
                statement.executeUpdate(updateQuery);
            }
        } else {
            String insertQuery = "INSERT INTO LEADERBOARD (Name, Score) VALUES ('" + playerName + "', " + score + ")";
            statement.executeUpdate(insertQuery);
        }

        resultSet.close();
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
1 usage  Stefan
public String[] bestScore() {
    String[] bestScores = new String[3];
    try {
        Statement statement = connection.createStatement();
        String query = "SELECT Name, Score FROM LEADERBOARD ORDER BY Score ASC LIMIT 3";
        ResultSet resultSet = statement.executeQuery(query);

        int index = 0;
        while (resultSet.next() && index < 3) {
            String playerName = resultSet.getString( columnLabel: "Name");
            int score = resultSet.getInt( columnLabel: "Score");
            String scoreEntry = playerName + ": " + score;
            bestScores[index] = scoreEntry;
            index++;
        }

        resultSet.close();
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return bestScores;
```

Incarcare harta si inamici:

Pentru a incarca harta am folosit o poza. Numarul de pixeli reprezinta dmiensiunile hartii. Pentru fiecare pixel din harta am verificat ce cod RGB are. Pentru valorile R am desenat un tile specific(daca pixelul are R=20 desenz al 20 element din vectorul de tile-uri). Pentru valorile lui G am instantiat tipuri de inamici si spawnpointul jucatorului, iar pentru valorile lui B am instantiat obiecte tip potiuni, chest, chei.

```
1 usage  Stefan
public static ArrayList<Goblin> getGoblin(BufferedImage img){
    ArrayList<Goblin> list=new ArrayList<>();
    for(int j=0;j<img.getHeight();j++)
        for(int i=0;i<img.getWidth();i++) {
            Color color=new Color(img.getRGB(i,j));
            int value=color.getGreen();
            if(value==GOBLIN)
                list.add(new Goblin( x: i*Game.TILE_SIZE, y: j*Game.TILE_SIZE));

        }
    return list;
}
```

```
1 usage  Stefan
public static int[][]GetLevelData(BufferedImage img){
    int air=145;
    int[][]LevelData=new int[img.getHeight()][img.getWidth()];

    for(int j=0;j<img.getHeight();j++)
        for(int i=0;i<img.getWidth();i++) {
            Color color=new Color(img.getRGB(i,j));
            int value=color.getRed();
            if(value>=air)
                value=air;
            LevelData[j][i]=value;

        }

    return  LevelData;
}
```

BIBLIOGRAFIE:

Sprite-uri folosite pentru harta si caractere: itch.io

Logo si fundal meniu generat cu ajutorul unui AI: midjourney