

Titlu proiect (Springston Woodland)

Autor (Iuliana-Diana Colțuneac)

Grupa(1210B)

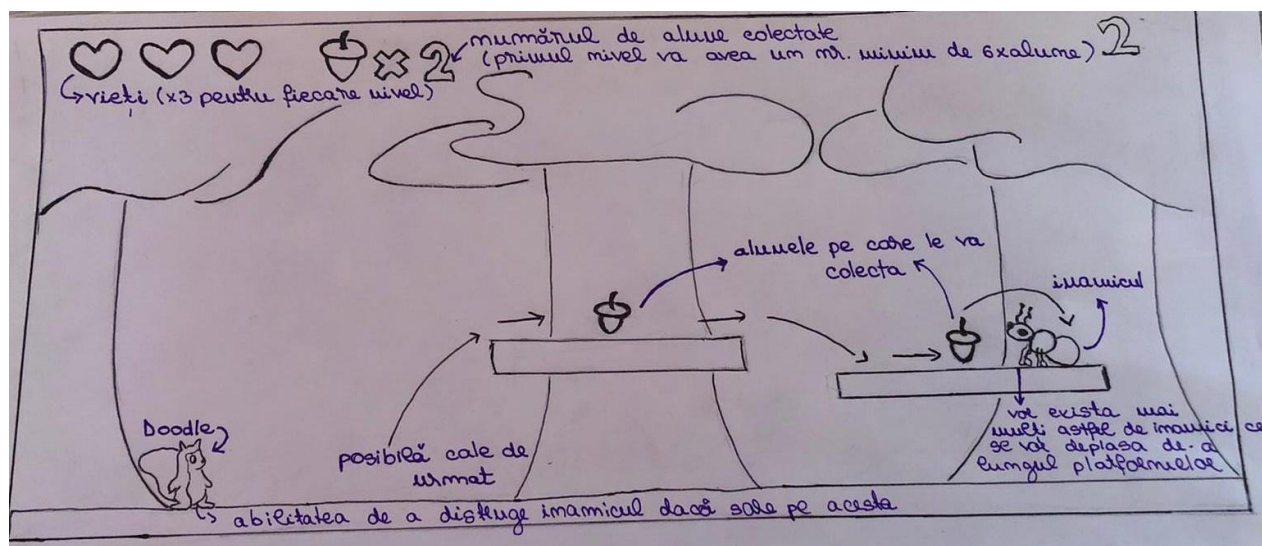
Proiectarea contextului și descriere detaliată:

Povestea incipientă:

Acțiunea jocului se situează în Springston Woodland, pădure în care se află laboratorul de clonare a doctorului Grief. Centrul său de cercetare se află ascuns în mijlocul liniștitei păduri, unde nimeni nu i-ar putea descoperi suspecta sa muncă. Singurul care a luat la cunoștință despre munca biologului este asistentul său, Plague, care neîndemânatic fiind a scăpat de sub control un experiment ce se afla în desfășurare în timp ce doctorul Grief luase o pauză. Stângăcia tânărului Plague a condus la eliberarea unui număr imens de insecte cu mutații genetice și puteri nemaîntâlnite ce vor pune stăpânire pe Springston Woodland. Singura slăbiciune cunoscută a armatei de insecte este nerezistența la temperaturile scăzute, dar din nefericire primăvara abia s-a instalat în pădure și toate viețuitoarele vor trebui să reziste încă 3 anotimpuri sub amenințarea experimentelor doctorului Grief. Misiunea jocului este de a o ajuta pe veverița Doodle să își adune mâncarea necesară până când iarna va veni și pădurea se va întoarce la pacea inițială.

Prezentare:

Jocul se încadrează în categoria jocurilor de acțiune, Platformer 2D, Singleplayer în care personajul controlat de jucător sare pe platforme suspendate sau peste obstacole pentru a avansa în joc. Veverița Doodle trebuie să colecteze un anumit număr de articole (alune) pentru a-și asigura hrana și să ajungă, fără a fi doborâtă de antagoniști, într-un anumit punct al hărții pentru a promova nivelul.



schita jocului

Proiectarea sistemului și descriere detaliată:

- ✓ Componente pasive: platforme peste care personajul va trebui să sară pentru a ajunge la obiectele pe care va trebui să le colecteze;
- ✓ Componente active: alunele pe care le va colecta personajul principal, trape care vor duce la marta protagonistului;
- ✓ Structura tablei de joc: Tabla de joc va fi inițiată la începutul fiecărui nivel, harta fiind alcătuită din trei nivele;

Mecanica Jocului:

Pentru a câștiga jucătorul va trebui să parcurgă toate cele trei nivele omorând toți adversarii.

La începutul fiecărui nivel jucătorul va avea o anumită măsură a vieții. O interacțiune cu inamicii de la primele două nivele va scădea din numărul inițial o viață, însă interacțiunea cu inamicul final va duce la pierderea nivelului (toate viețile de la momentul respectiv vor fi luate).

Doodle va fi nevoită să se ferească atât de venin cât și de inamicii propriu-ziși. Modul în care protagonistul va putea să ucidă inamicii constă în săritura pe ei.

La finalul fiecărui nivel jucătorul va primi un scor ce se va calcula în funcție de numărul de adunate fructe de zmeură adunate.

Controalele jucatorului:



Deplasarea în sus



Deplasarea în jos



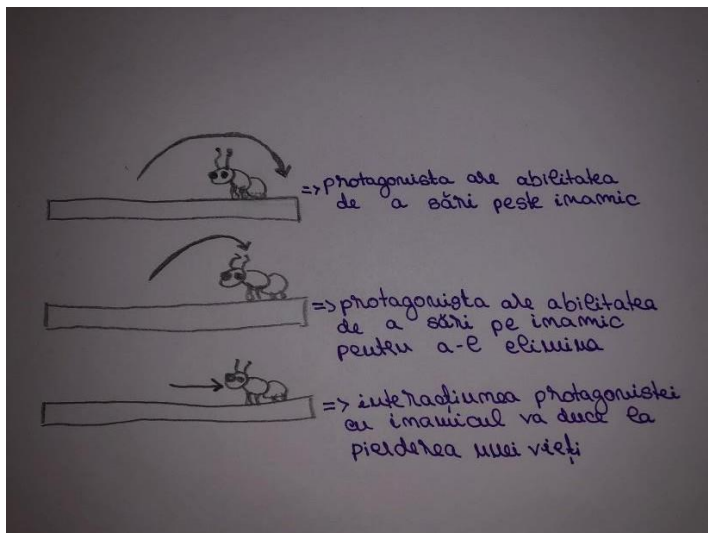
Deplasarea în dreapta



+



Atac



Proiectarea conținutului și descriere detaliată:

Prezentarea personajelor și item-urilor:

Doodle: protagonista jocului, personajul controlat de jucător, este o veveriță din pădurea Springston ce are mult curaj și pornește să confrunte experimentele doctorului Grief cu scopul adunării resurselor de hrană necesare atât ei, cât și prietenilor ei. Aceasta execută sărituri în înălțime și aleargă rapid, agerimi ce o vor ajuta în depășirea obstacolelor.



Giant Ants: inamicii primului nivel, ce vor patrula prin Springston Woodland și o vor împiedica pe Doodle în colectarea celor necesare. Înțepătura lor este periculoasă, astfel a treia înțepătură va fi letală.



GrassHoppers: inamicii celui de-al doilea nivel.



Gators: inamicii nivelului 3.



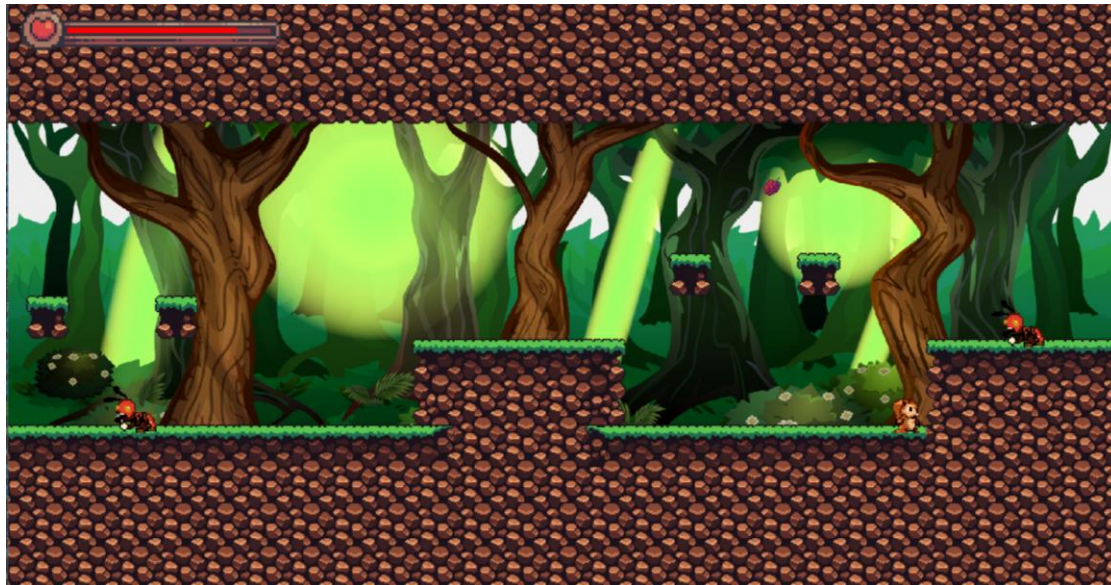
Nuts: obiectele ce vor fi colectate de Doodle, iar acestea vor crește valoarea scorului cu o anumită valoare(15 pct);



Proiectarea nivelurilor și descriere detaliată:

Jocul va conține 3 niveluri ce respectă firul narativ al poveștii și se vor desfășura în cele 3 anotimpuri în care Springston Woodland va fi infestată cu insectele mutante. Jocul va începe primăvara și se va încheia atunci când playerul va reuși să o ajute pe veverița Doodle să treacă de toate nivelurile și să ajungă în timpul iernii când temperaturile scăzute din pădure vor combate intrușii eliberați de laboratoarele doctorului Grief.

Astfel, primul nivel se va desfășura primăvara, iar Doodle va fi nevoită să înfrunte un singur tip de inamic, Giant Ants(prezentate în pagina 2). Aceasta va trebui să omoare toți inamicii și să ajungă la finalul hărții fără să fie înțepată de furnicile mutante de mai mult de 5 ori.



primul nivel

Nivelul al doilea va semnifica încercarea lui Doodle să supraviețuiască infestării pe timpul verii. De această dată protagonistă va fi pusă în dificultate de 2 tipuri de adversari. La Giant Ants se vor adăuga alte greșeli de-ale doctorului Grief ce vor putea arunca venin spre protagonistă, infestarea cu această ducând la scăderea vieții lui Doodle. Pe lângă amenințarea mai mare, acest nivel va aduce și cerințe mai mari în ce privește numărul adversarilor.



Al doilea nivel

Al treilea nivel, cel final, se va desfășura în timpul toamnei și va crește dificultatea jocului prin adăugarea unui nou tip de inamic a cărui înțepătură va fi letală(dacă Doodle va atinge inamicul final sau va fi atinsă de veninul aruncat de antagonist aceasta își va pierde toate cele trei vieți sau viețile rămase). De asemenea, prezența trapelor ce vor fi letale entru protagonistă va crește dificultatea.



Al treilea nivel

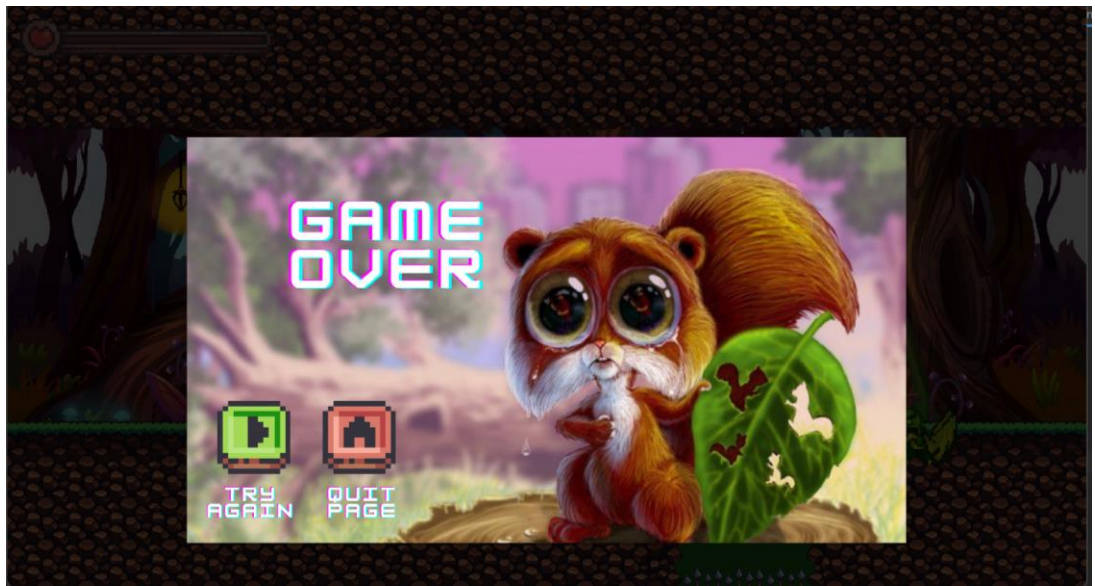
Proiectarea interfeței cu utilizatorul și descriere detaliată:

Ui-ul va conține viața actuală. Scorul obținut va fi afișat la încheierea nivelului.

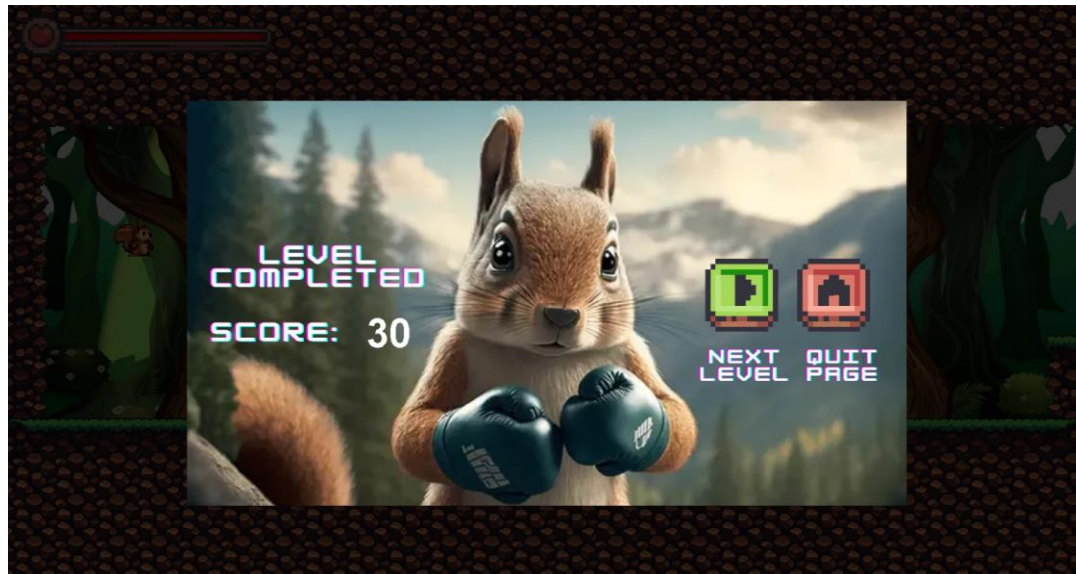
Opțiunile pentru începerea jocului vor fi play și quit.



Meniu



GameOver



LevelCompleted

Descriere algoritmi utilizați:

- Coliziuni:

Pentru managerierea coliziunilor dintre protagonist și inamici s-a folosit o funcție ce verifică dacă attackBoxul protagonistului se intersectează cu attackBoxul inamicilor astfel:

```
protected void checkEnemyHit(Rectangle2D.Float attackBox, Player player) {
    if(attackBox.intersects(player.hitBox))
        player.changeHealth(-GetEnemyDmg(enemyType));
    attackChecked=true;
}
```

Pentru tratarea coliziunilor dintre protagonist și obiecte s-a folosit, în mod similar, o serie de funcții ce lucrează, de această dată cu hitBoxurile:

```
public void checkSpikesTouched(Player p){
    objectManager.checkSpikesTouched(p);
}

public void checkBerriesTouched(Rectangle2D.Float hitbox){
    objectManager.checkObjectTouched(hitbox);
}
```

```
public void checkSpikesTouched(Player player){
    for(Spike s: spikes)
        if(s.getHitbox().intersects(player.getHitbox()))
            player.kill();
} //daca atinge spikeurile moare
```

```

public void checkObjectTouched(Rectangle2D.Float hitbox) {
    for (Berries b : berries)
        if (b.isActive()) {
            if (hitbox.intersects(b.getHitbox())) {
                b.setActive(false);
                applyEffectToPlayer(b);
            }
        }
}
}

```

- **gestiune obiecte/inventar:**

Obiectele vor fi încărcate într-un array. Acestea vor avea o proprietate numită `isActive`, care va determina dacă obiectul a fost sau nu colectat.

- **deplasarea inamicilor:**

Deplasarea inamicilor se va efectua cu limitarea zonei în care au fost așezați(atât timp cât nu este perete și se află pe tilesuri). Aceștia vor urmări direcția de deplasare a protagonistului(funcția `turnTowardsPlayer`) atunci când veverița va fi „în câmpul lor visual”, adică la același nivel pe axa Oy și o anumită distanță pe axa Ox(`canSeePlayer`).

```

• protected void move(int[][] lvlData){
    float xSpeed = 0;

    if (walkDir == LEFT)
        xSpeed = -walkSpeed;
    else
        xSpeed = walkSpeed;

    if (CanMoveHere(hitBox.x + xSpeed, hitBox.y, hitBox.width,
hitBox.height, lvlData))
        if (IsFloor(hitBox, xSpeed, lvlData)) {
            hitBox.x += xSpeed;
            return;
        }

    changeWalkDirection();
}

protected void turnTowardsPlayer(Player player){
    if(player.hitBox.x>hitBox.x)
        walkDir=RIGHT;
    else
        walkDir=LEFT;
}

protected boolean canSeePlayer(int[][] lvlData, Player player){
    int playerTileY=(int) (player.getHitbox().y/Game.TILES_SIZE);
    if(playerTileY==tileY)
        if(isPlayerInRange(player)){
            if(IsSightClear(lvlData, hitBox, player.hitBox, tileY))
                return true;
        }
    return false;
}

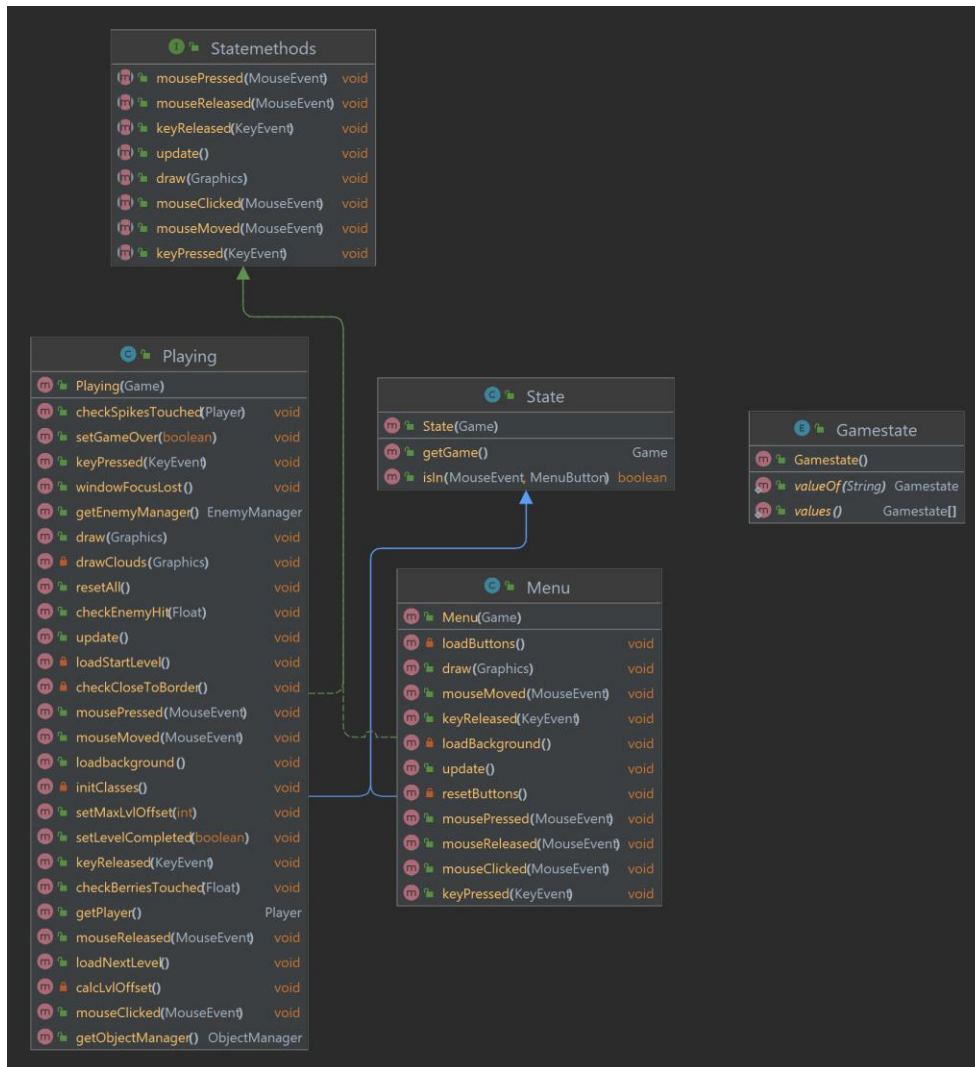
```

Diagrama întregului proiect:
























Pentru o vizualizare mai ușoară:

Package-ul gamestates:

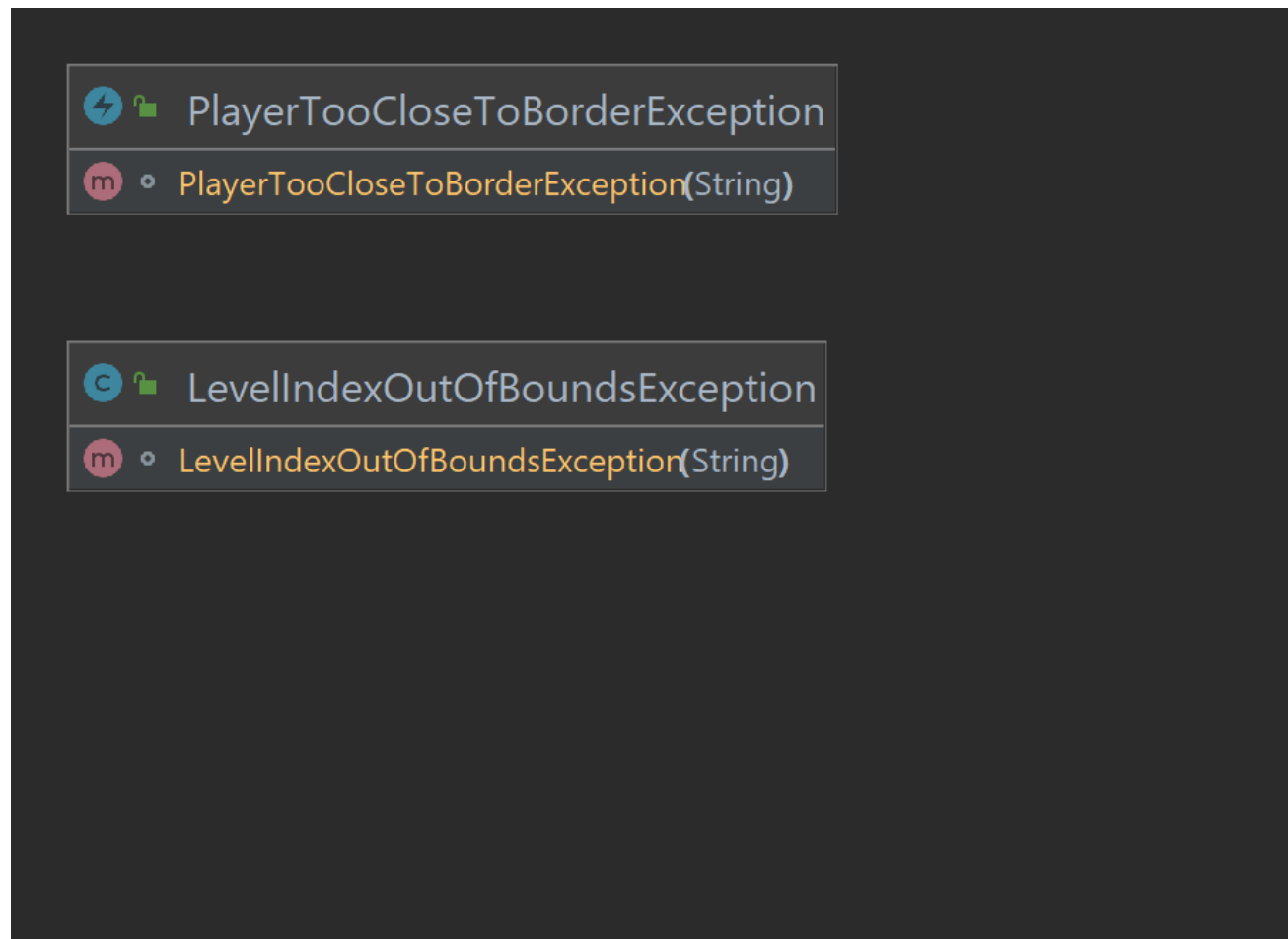
Design Pattern State:



Package-ul DataBase:

 DataBase		
	 DataBase()	
	 <i>score</i>	int
	 <i>instance</i>	DataBase
	 stmt	Statement
	 c	Connection
	 getConnection()	void
	 stmt	Statement
	 scoreInDatabase	int
	 c	Connection
	 <i>score</i>	int
	 <i>instance</i>	DataBase

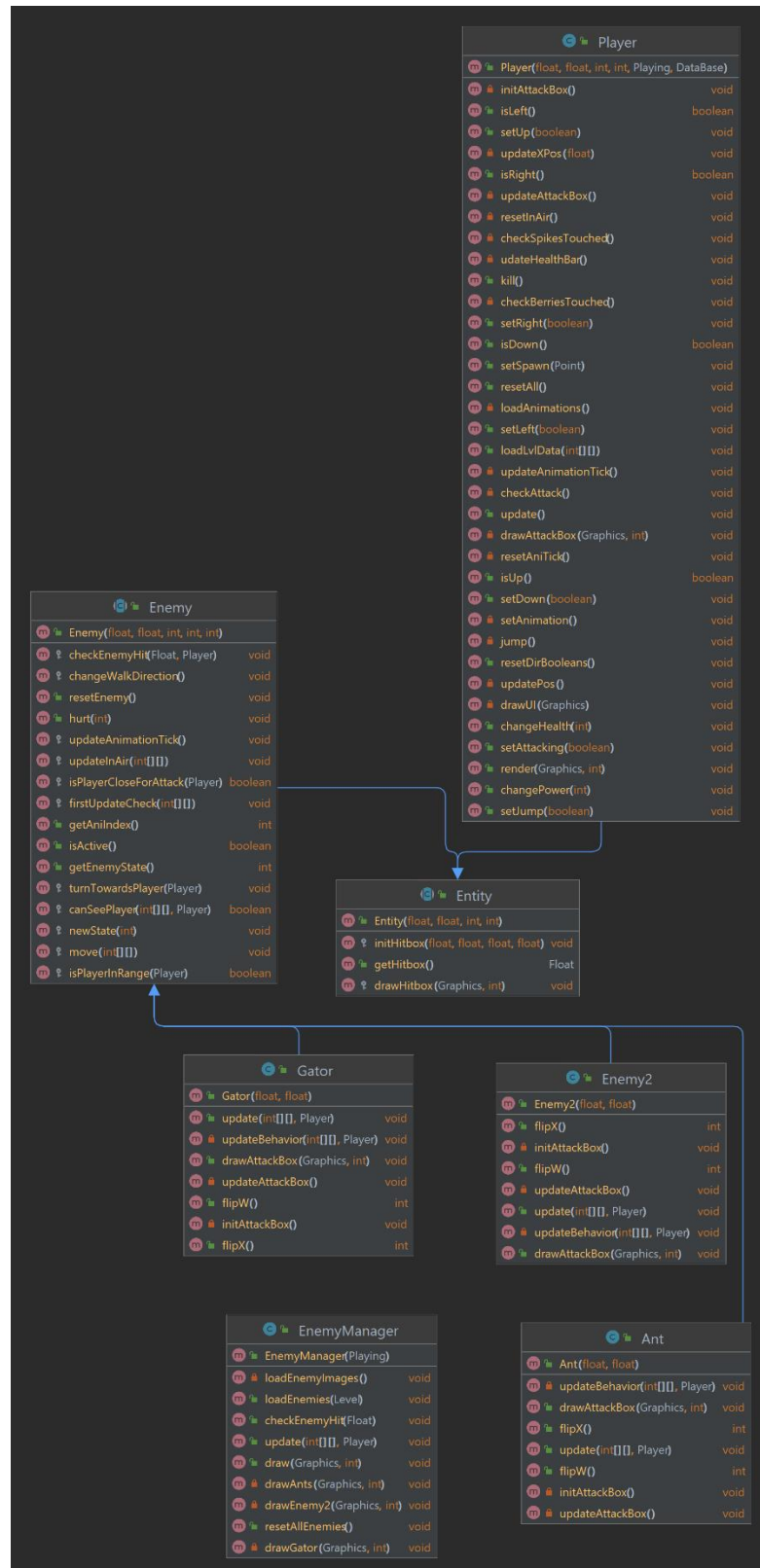
Package-ul Exception:



Acesta implementează 2 clase de excepții custom pentru tratarea următoarelor cazuri:

1. Indexul ce stabilește numărul nivelului depășește numărul maxim de nivele implementate prin incrementări multiple;
2. Personajul principal se află într-o zonă a hărții cu acces restricționat;

Package-ul Entities:



Conține atât clasa ce implementează abilitățile protagonistului, cât și inamicii pentru fiecare nivel ce sunt coordonați de un EnemyManger;

Package-ul inputs:

KeyboardInputs		
m	KeyboardInputs(GamePanel)	
m	keyPressed(KeyEvent)	void
m	keyTyped(KeyEvent)	void
m	keyReleased(KeyEvent)	void

MouseInputs		
m	MouseInputs(GamePanel)	
m	mousePressed(MouseEvent)	void
m	mouseReleased(MouseEvent)	void
m	mouseExited(MouseEvent)	void
m	mouseMoved(MouseEvent)	void
m	mouseClicked(MouseEvent)	void
m	mouseEntered(MouseEvent)	void
m	mouseDragged(MouseEvent)	void

Managerierea intrărilor de la mouse și tastatură

Package-ul levels:

Level		
m	Level(BufferedImage)	
m	createEnemies()	void
m	getSpriteIndex(int, int)	int
m	calcPlayerSpawn()	void
m	createLevelData()	void
m	createBerries()	void
m	createSpikes()	void
m	createNuts()	void
m	calcLvloffsets()	void
p	ants	ArrayList<Ant>
p	enemy2	ArrayList<Enemy2>
p	playerSpawn	Point
p	nuts	ArrayList<Nuts>
p	gator	ArrayList<Gator>
p	berries	ArrayList<Berries>
p	maxLvloffset	int
p	levelData	int[][]
p	spikes	ArrayList<Spike>

LevelManager		
m	LevelManager(Game)	
m	draw(Graphics, int)	void
m	update()	void
m	importOutsideSprites()	void
m	loadNextLevel()	void
m	buildAllLevels()	void
p	currentLevel	Level
p	lvlIndex	int
p	amountOfLevels	int

Pentru managerierea celor 3 nivele am construit un LevelManger ce va încărca nivelele în ordinea corectă; Clasa Level se ocupă cu crearea personajelor, hărții și obiectelor pentru fiecare nivel în parte

Package-ui:

<div><div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div>MenuButton</div></div></div></div> <div><div><div>m</div><div></div></div><div>MenuButton(int, int, int, Gamestate)</div></div> <div><div><div>m</div><div></div></div><div>resetBools()</div><div>void</div></div> <div><div><div>m</div><div></div></div><div>loadImgs()</div><div>void</div></div> <div><div><div>m</div><div></div></div><div>draw(Graphics)</div><div>void</div></div> <div><div><div>m</div><div></div></div><div>update()</div><div>void</div></div> <div><div><div>m</div><div></div></div><div>applyGamestate()</div><div>void</div></div> <div><div><div>m</div><div></div></div><div>initBounds()</div><div>void</div></div> <div><div><div>p</div><div></div></div><div>mouseOver</div><div>boolean</div></div> <div><div><div>p</div><div></div></div><div>bounds</div><div>Rectangle</div></div> <div><div><div>p</div><div></div></div><div>mousePressed</div><div>boolean</div></div>
--

Acest package conține 3 clase ce reprezintă stări ale ecranului de joc și au ca scop atât implementarea comenzilor de intrare-ieșire, cât și desenarea ecranelor corespunzătoare: Meniu, GameOver & LevelCompleted;

Package-ul util:

Constants

Constants()

HelpMethods

HelpMethods()

CanMoveHere(float, float, float, float, int[][]) boolean

IsAllTileWalkable(int, int, int, int[][]) boolean

IsEntityOnFloor(Float, int[][]) boolean

GetPlayerSpawn(BufferedImage) Point

GetAnts(BufferedImage) ArrayList<Ant>

GetBerries(BufferedImage) ArrayList<Berries>

IsTileSolid(int, int, int[][]) boolean

IsSolid(float, float, int[][]) boolean

GetLevelData(BufferedImage) int[]

GetSpikes(BufferedImage) ArrayList<Spike>

GetEntityXPosNextToWall(Float, float) float

IsFloor(Float, float, int[][]) boolean

IsSightClear(int[][], Float, Float, int) boolean

GetNuts(BufferedImage) ArrayList<Nuts>

GetEntityPosUnderRooforAboveFloor(Float, float) float

LoadSave

LoadSave()

GetBerries(BufferedImage) ArrayList<Berries>

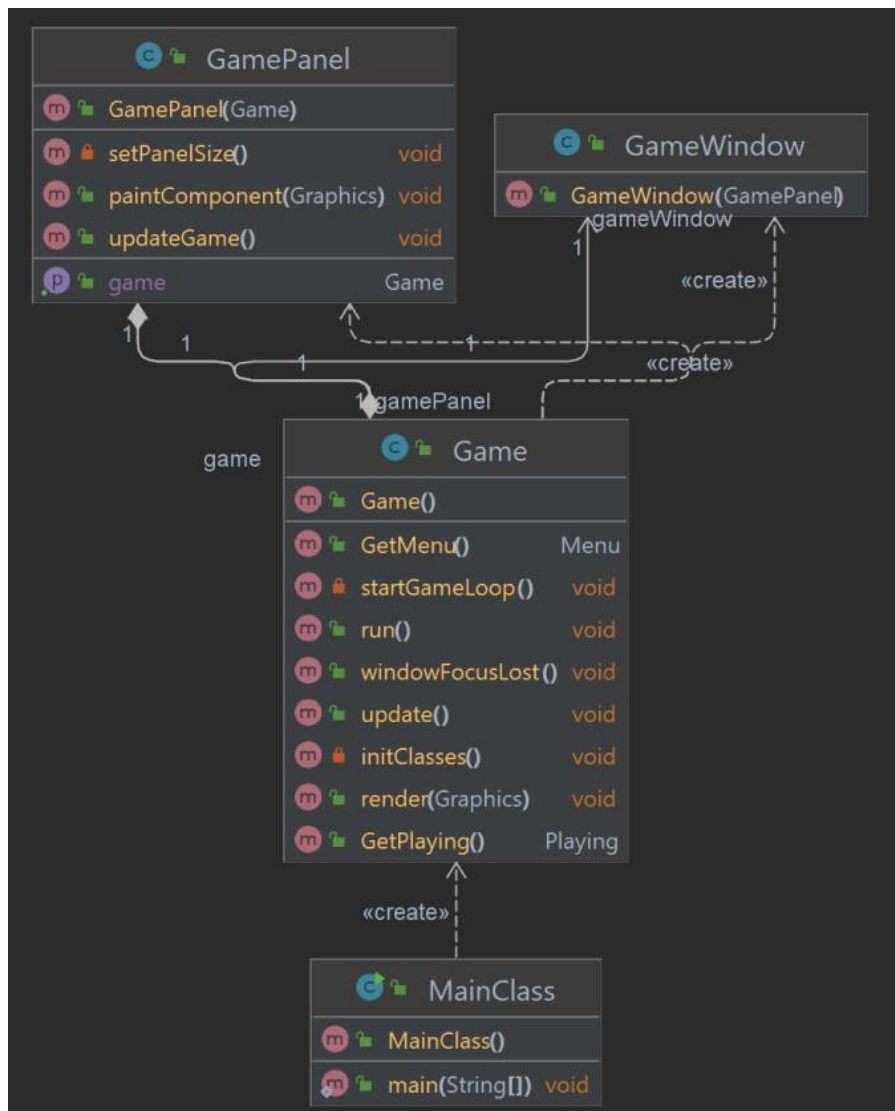
GetSpriteAtlas(String) BufferedImage

GetNuts(BufferedImage) ArrayList<Nuts>

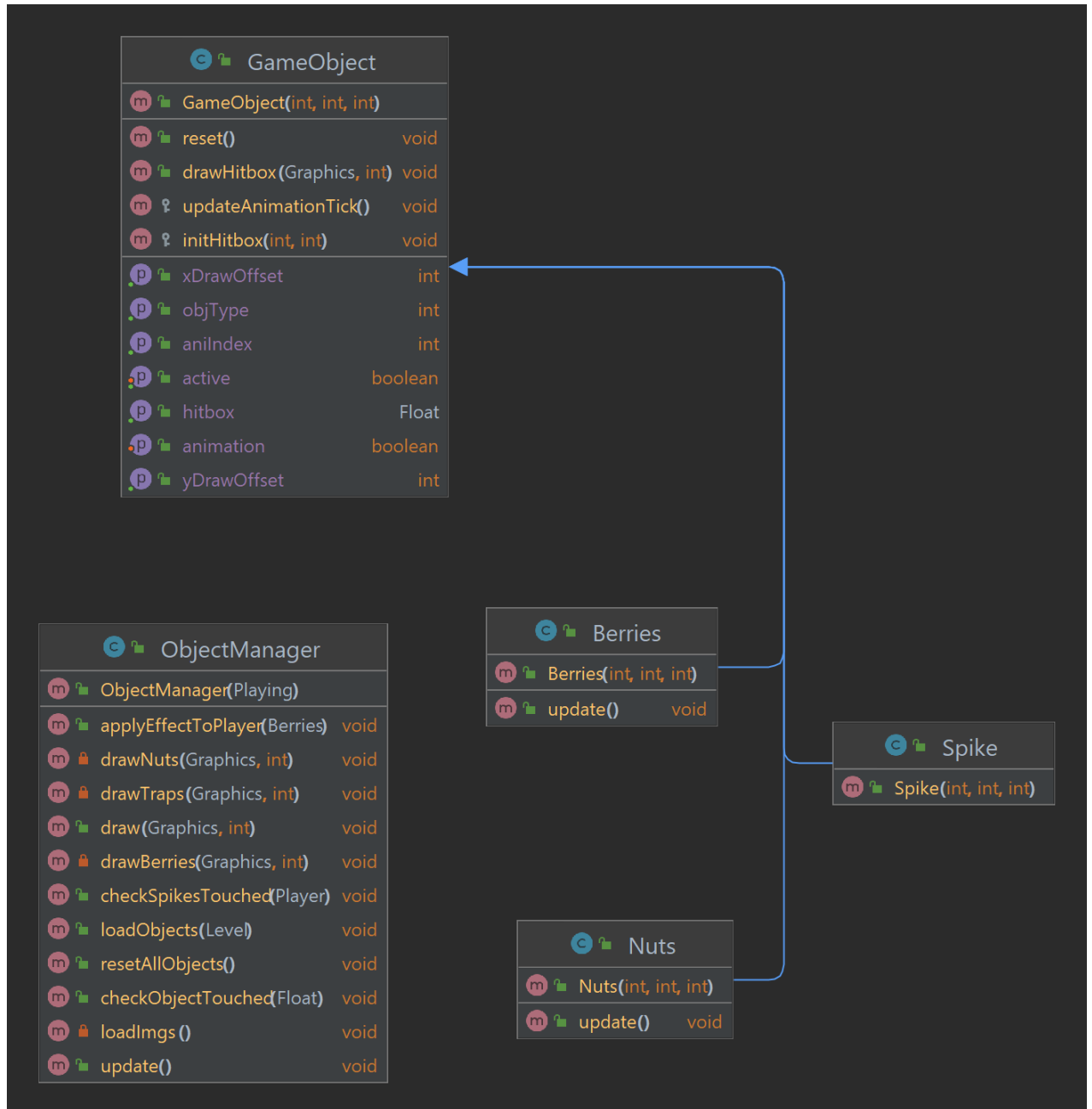
GetAllLevels() BufferedImage[]

Clasa HelpMethods conține metode pentru tratarea coliziunilor dintre protagonist și inamici sau obiecte;
Clasa LoadSave încarcă sprite-urile folosite în joc;

Package-ul main:



Package-ul objects:

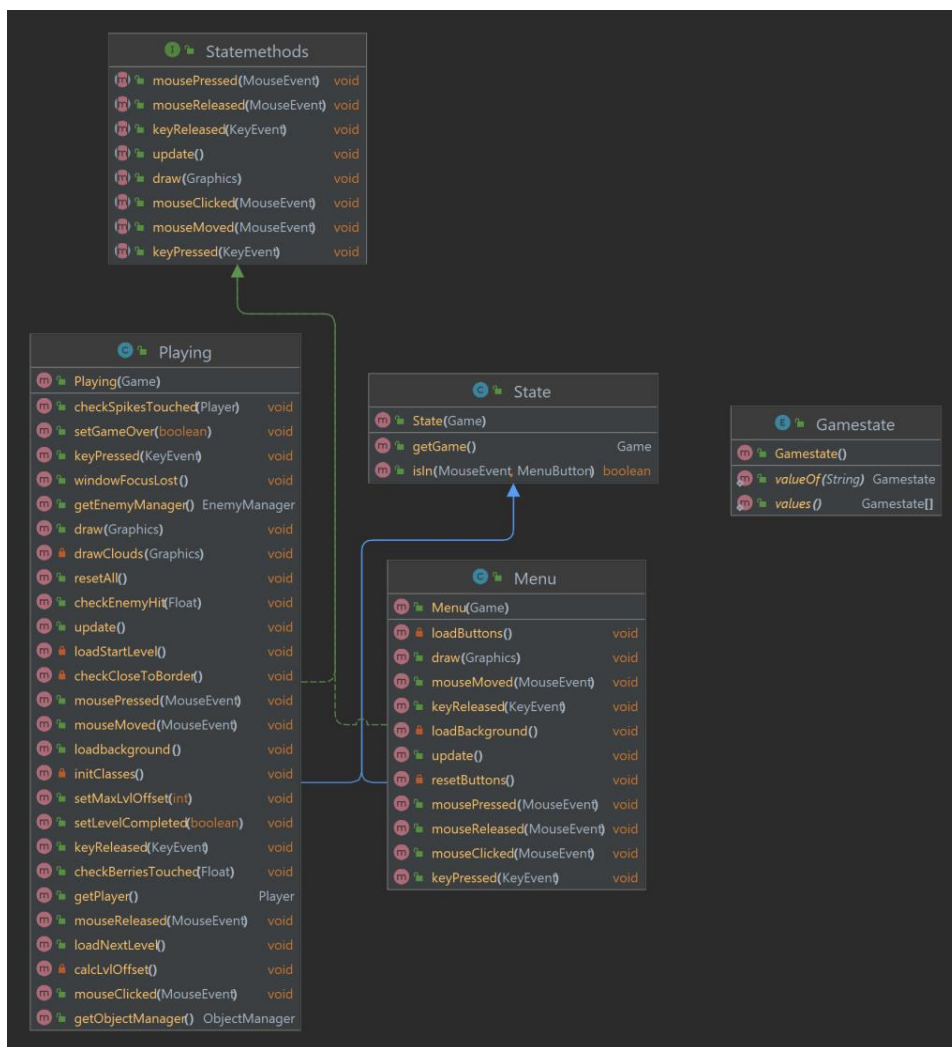


Design Patterns:

✓ State:

Ideea principală este de a permite obiectului să-și schimbe comportamentul fără a-și schimba clasa. Logica din spatele schimbării state-urilor este separată și adăugarea de noi state-uri este simplă.

Astfel, clasele Menu și Playing implementează interfața StateMethods și extind clasa State.



✓ Singleton:

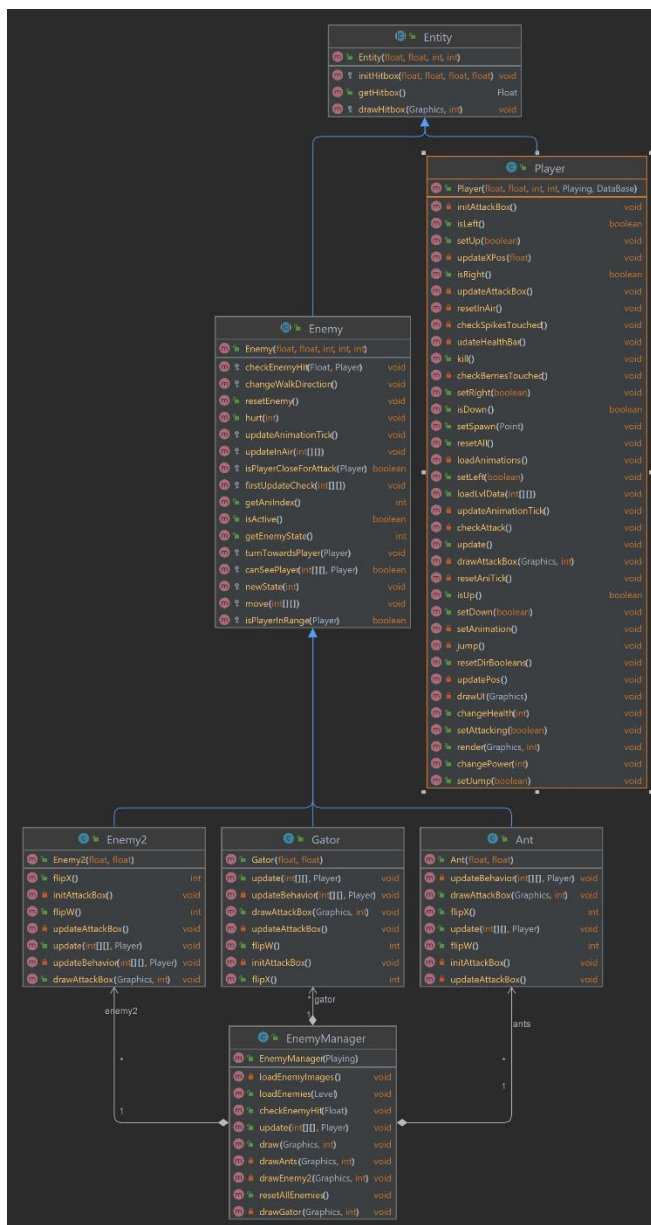
Acesta s-a folosit pentru crearea bazei de date. Deoarece crearea bazei de date trebuie să fie unică s-a recurs la folosirea unui constructor privat și la folosirea unei metode(`getInstance`) ce va construi un obiect global, unic.

✓ TypeObject:

Acest Design Pattern oferă posibilitatea adăugării unor noi tipuri de obiecte într-un mod flexibil.

Astfel, clasa Enemy și clasa Player extind clasa Entity, iar Ant, Gator & GrassHopper extind clasa Enemy.

Pentru crearea unui nou inamic se folosește constructorul clasei Enemy.



Bibliografie:

Grafica jucător + inamici:

<https://opengameart.org/content/sunnyland-woods>

Backgrounds:

<https://free-game-assets.itch.io/free-cartoon-forest-2d-backgrounds>

Imagini meniu+ Butoane:

<https://www.freepik.com/>