

**The treasure from
the depths
Buțu Alexandra
Grupa 1207B**

**Proiectare context si descriere:
Povestea jocului:**

Acțiunea are loc în anul 3000, iar lumea se află într-un haos total din cauza suprapopulației și a poluării. În efortul de a găsi noi resurse și noi locuri de populație, omenirea își îndreaptă atenția către adâncurile oceanului. Orion, un scafandru calificat, cu abilități magice, este recrutat de o echipă de oameni de știință și ingineri pentru a explora Marea Paradis, o zonă recent descoperită care deține bogății și un nespus potențial.

Echipat cu un costum de scafandru de ultimă generație și o sferă mistică, Orion pornește în misiune. Curând descoperă că nu este singur în colțurile adânci și întunecate ale mării, căci există creațuri ciudate și ruine antice ce zace ascunse și așteaptă să fie descoperite.

În timp ce Orion navighează în adâncurile Mării Paradis, el întâlnește fințe subacvatice care îi amenință misiunea: sirene mortale cu abilități magice puternice, pești sălbatici și cea mai mare amenințare dintre toate: un rechin masiv și terifiant care păzește cele mai valoroase comori.

Folosind sfera sa mistică, Orion se luptă cu valuri de inamici și obstacole, adunând perle și resurse valoroase pe parcurs. Dar cu cât merge mai adânc, cu atât își dă seama că Marea Paradis are mai mult decât o comoară. Tehnologia antică și secretele zace îngropate în adâncuri și așteaptă să fie descoperite.

Cu soarta omenirii atârnând în balanță, Orion trebuie să-și folosească abilitățile și puterile nu numai pentru a colecta perle neprețuite, ci și pentru a descoperi misterele Mării Paradis și pentru a găsi o modalitate de a salva lumea de soarta ei crudă.



Prezentare joc:

„The treasure from the depths „ este un palpitant joc 2D a cărui acțiune se petrece în mediul subacvatic. Alătură-te scafandrului Orion, personajul principal, în încercarea sa de a colecta cât mai multe perle, în timp ce luptă cu dușmanii periculoși și obstacolele din calea lui.

Jocul este format din 4 niveluri, fiecare cu un grad diferit de dificultate, în care Orion navighează în adâncuri, încercând să evite peștii fioroși și sirenele mortale, care sunt sub controlul rechinului amenințător Metis.

Pentru a trece de la un nivel la altul, scafandrul trebuie să acumuleze un anumit scor și să evite să-și piardă viață. Modul de joc este plin de diverse elemente care fie vor ajuta, fie vor împiedica progresul lui Orion. Adună perle pentru a-ți crește scorul, dar ai grija la peștii care îl pot scădea. Inimile mistice împrăștiate pe tot parcursul jocului te ajută să aduni viață, dar scad punctajul.

Pe măsură ce nivelurile devin mai complexe, Orion trebuie să-și folosească strategic sfera mystică pentru a elimina obstacolele din calea lui. Ai grija să nu te ciocnești cu sirenele, deoarece fiecare ciocnire te costă o viață. Testul suprem te aşteaptă la nivelul 4, unde vei lupta cu rechinul nemilos. Învingând rechinul, vei

ieși câștigător, dar coliziunea cu acesta va pune capăt jocului și toate viețile tale vor fi pierdute.

Ești gata să te scufunzi în aventura vieții? Joacă acum „The treasure from the depths”, și vezi dacă ai ceea ce este nevoie pentru a ieși învingător.

Reguli joc:

- **Obiectivul:** Obiectivul jocului este ca personajul principal să adune perle și să elimine inamicii evitând în același timp obstacolele pentru a ajunge la nivelul final și a învinge inamicul suprem, rechinul. Jucătorul trebuie să acumuleze un anumit punctaj pentru fiecare nivel și să treacă prin toate cele patru niveluri fără a-și pierde toată viața.
- **Mecanica jocului:** Jucătorul îl controlează pe Orion mutându-l în sus sau în jos, în stânga sau dreapta pe ecran cu ajutorul săgeților pentru a evita inamicii și obstacolele în timp ce colectează perle și alte elemente. Jucătorul folosește sfera mistică pe care o posedă Orion pentru a elimina inamicii, apăsând tastă SPACE.
- **Inamicii:** Dușmanii apar în număr tot mai mare în funcție de nivel. Peștii și sirenele sunt inamici pasivi care scad puncte din scorul jucătorului și pot, de asemenea, să distrugă viața lui Orion. Rechinul este inamicul suprem care apare doar la nivelul 3 și trebuie învins pentru a câștiga jocul.
- **Niveluri:** Jocul are trei niveluri, fiecare cu propriul set de inamici, obstacole și cerințe de scor. Pentru a trece la nivelul următor, jucătorul trebuie să obțină un scor minim pentru fiecare nivel. Dacă jucătorul nu reușește să atingă scorul necesar, trebuie să reia acel nivel. Jucătorul poate pierde respectivul nivel dacă intra în coliziune cu inamici pana la terminarea vieților.
- **Scor:** Scorul jucătorului crește prin colectarea de perle și eliminarea sirenelor. Fiecare perlă adaugă 5 puncte la scor, în timp ce fiecare sirenă adaugă 10 puncte. Scorul jucătorului scade prin atingerea peștilor și a inimii. Fiecare pește scade 5 puncte din scor, în timp ce fiecare inimă

adaugă 5 vieți, dar scade 30 de puncte din scor. Scorul și viața raman de la nivel la nivel.

- **Viața:** Jucătorul începe jocul cu 10 vieți, iar fiecare ciocnire cu un inamic sau obstacol scade o viață, cinci sau zece. Dacă jucătorul își pierde toată viață, jocul se termină. Jucătorul poate colecta inimi pentru a câștiga vieți suplimentare.
- **Condiția de câștig:** Rechinul este inamicul final care apare doar la nivelul 3. Jucătorul trebuie să-l atingă cu sfera mistică de patru ori pentru a-l elibera și a câștiga jocul. Dacă jucătorul se ciocnește de rechin sau rechinuliese din ecran, jocul este pierdut.
- **Taste+butoane:**
 - Apasand tasta ESC se ieșe din joc și se intră în meniul secundar
 - Apasand tasta SPACE se eliberează sfera mistică
 - Apasand butonul NEW GAME se începe joc nou de la nivelul 1
 - Apasand butonul CONTINUE GAME se reia jocul de unde a ramas
 - Apasand butonul LOAD se reia jocul de unde a ramas înainte de inchiderea acestuia
 - Apasand butonul EXIT se închide jocul
 - Apasand butonul INFO apare meniul cu informații
- **Conditii trecere nivele:**
 - Pentru a trece din primul nivel la al doilea sunt necesare 40 de puncte
 - Pentru a trece din al doilea nivel la al treilea sunt necesare 80 de puncte
 - Pentru a câștiga jocul este necesara uciderea rechinului

Personajele jocului:

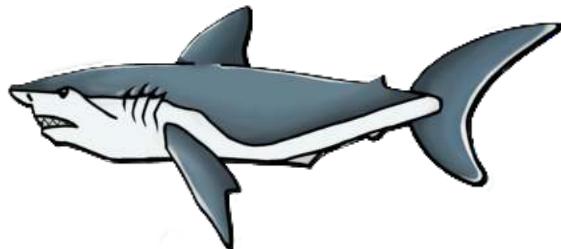
- **Orion** este numele protagonistului, el fiind jucătorul-personaj. El este un scafandru priceput și experimentat, care este hotărât să adune comorile ascunse în adâncurile Mării Paradis. Este curajos și plin de resurse, nu se da înapoi de la o provocare și e întotdeauna gata să înfrunte orice obstacol

care îi iese în cale. S-a antrenat intens pentru a-și dezvolta abilitatea de a se scufunda și a stăpâni sfera sa mistică, un instrument puternic pe care îl folosește pentru a se proteja de inamicii săi.



(Personaj desenat in photoshop)

- **Rechinul ucigaș Metis** joacă rolul personajului secundar, fiind un NPC. este o creatură masivă și amenintătoare, cu dinți ascuțiti și o coadă puternică care îl propulsează prin apă cu o viteză incredibilă. Pielea lui cenușie închisă este acoperită de cicatrici de la nenumărate bătălii, iar ochii lui negri și reci dezvăluie o inteligență aprigă și nemiloasă. Metis este regele oceanului, temut de toate creaturile care îndrăznesc să-i traverseze calea. La nivelul 3, Metis iese din adâncuri, gata să provoace eroul într-o luptă finală. Eroul trebuie să-și folosească toată îndemânarea și curajul pentru a evita atacurile rechinului și pentru a riposta cu sfera sa mistică.



(Personaj desenat in photoshop)

- **Sirenele mării** (de asemenea NPC-uri) sunt dușmani periculoși cu care Orion trebuie să-i înfrunte pe tot parcursul călătoriei sale. Ei sunt subalterni ai rechinului, care ascultă ordinele sale de a-l captura pe curajosul scafandru. Sirenele sunt agile și înloată rapid, ceea ce face dificil pentru Orion să-și evite atacurile. Dacă jucătorul se ciocnește de o sirenă, personajul își pierde treptat viața, aducându-l cu un pas mai aproape de eșec. Sirenele apar în număr tot mai mare la fiecare nivel, ceea ce face ca Orion să progreseze mai mult.



Din față
(Personaj desenat in photoshop)



Lateral

Tabla de joc:

Componente pasive:

- **Scoicile cu perlă** (regăsite în fiecare nivel) sunt elemente care plutesc în apă, iar la acumularea lor, jucătorului îi crește scorul (cu 5 p).



- **Peștii** (regăsiți în fiecare nivel) sunt elemente care plutesc în apă, iar la acumularea lor, jucătorului îi scade scorul (cu 5 p).



- facut in photoshop

- **Inimile** (regăsite în fiecare nivel incepând cu al treilea) sunt elemente care nu se mișcă (plutesc în apă), iar la acumularea lor, jucătorului îi crește viața (cu 5), dar totodată îi scade scorul (cu 30 p).

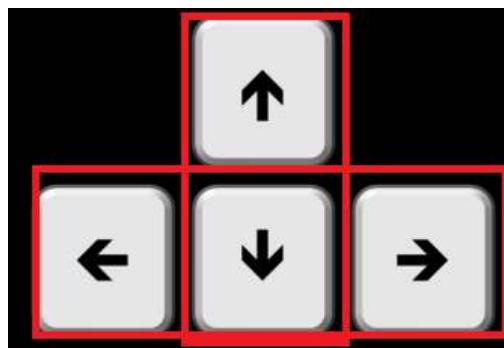


- **Bombele** (regăsite în fiecare nivel incepand cu al doilea) sunt elemente care plutesc în apă cu o viteza mai mare decât a celorlalte elemente, iar la atingerea lor jucătorului îi scade viața (cu 5).



Componente active:

- Personajul principal se deplasează sus, jos, stanga, dreapta cu ajutorul săgețiilor de la tastatură.



- **Inamicii** (NPC-urile) îi apar în cale personajului principal și permit trecerea prin ele. Cand intra în coliziune cu acestea i se scade din viață.

*Rechinul este inamicul final care apare în ultimul nivel și va necesita o

forță suplimentară pentru a fi anihilat (lovit de mai multe ori cu sfera mistică -4). Dacă personajul principal va intra în contact cu acesta, jocul se va sfârși cu înfrângerea lui.

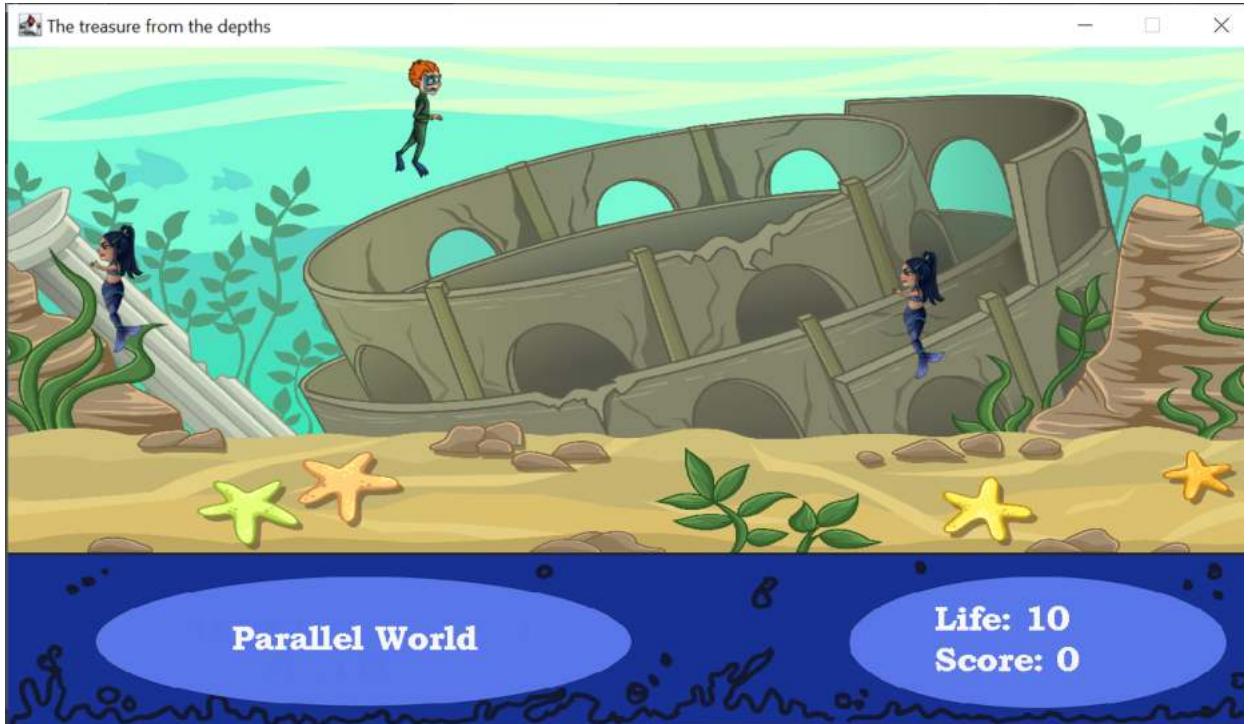
- **Sfera mistică** este puterea cu care personajul principal e înzestrat. E un element activ ce țintește dușmanii. Cand ei intră în contact cu aceasta, vor mori(dispar de pe ecran).



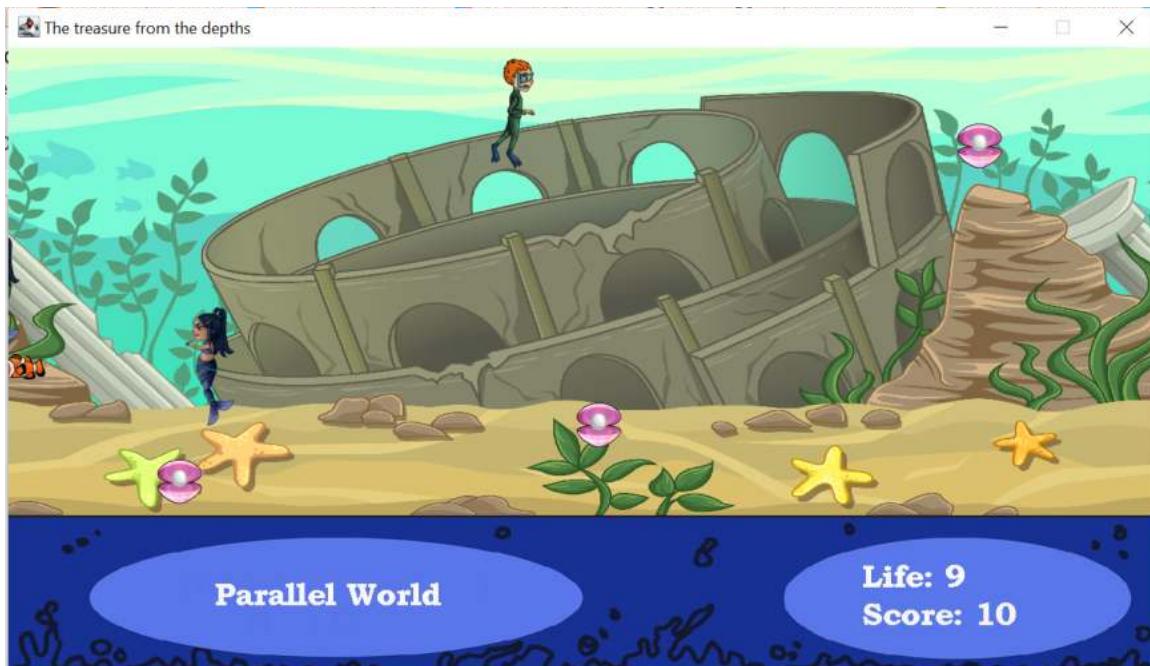
Tabla de joc reprezintă background-ul unde se petrece acțiunea jocului și aceasta va fi diferită de la un nivel la altul. Ea este creată astfel încât să dea impresia de mișcare a personajelor și a tileset-urilor folosite, fiind o imagine ce rulează continuu.

Am implementat pentru etapa 2 :

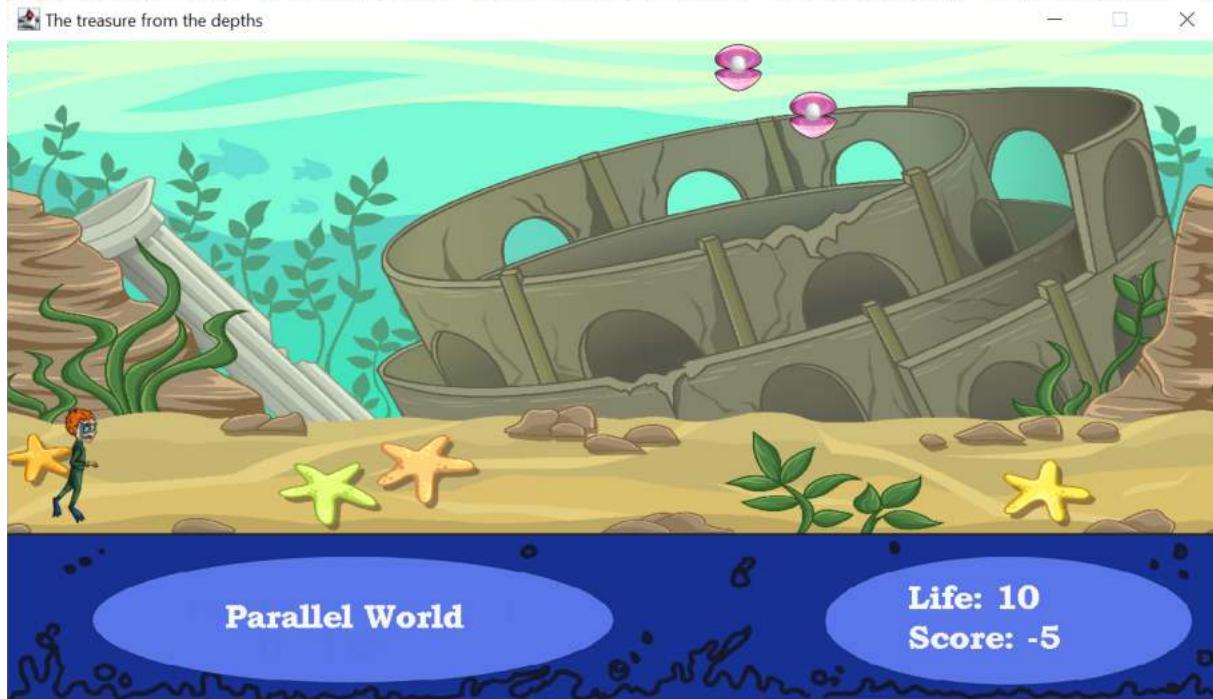
Initial la inceperea jocului cu scor=0, viata=10



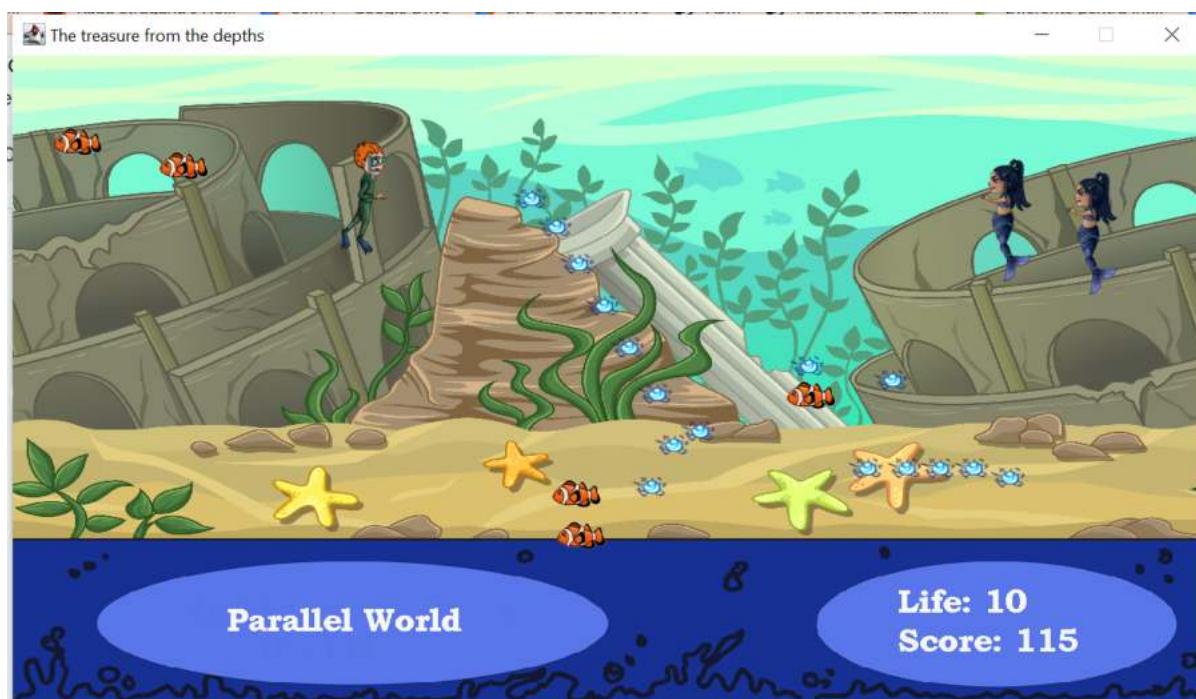
După colectarea unei sirene și eliminarea uneia, scorul crește cu 10, dar viata scade cu 1



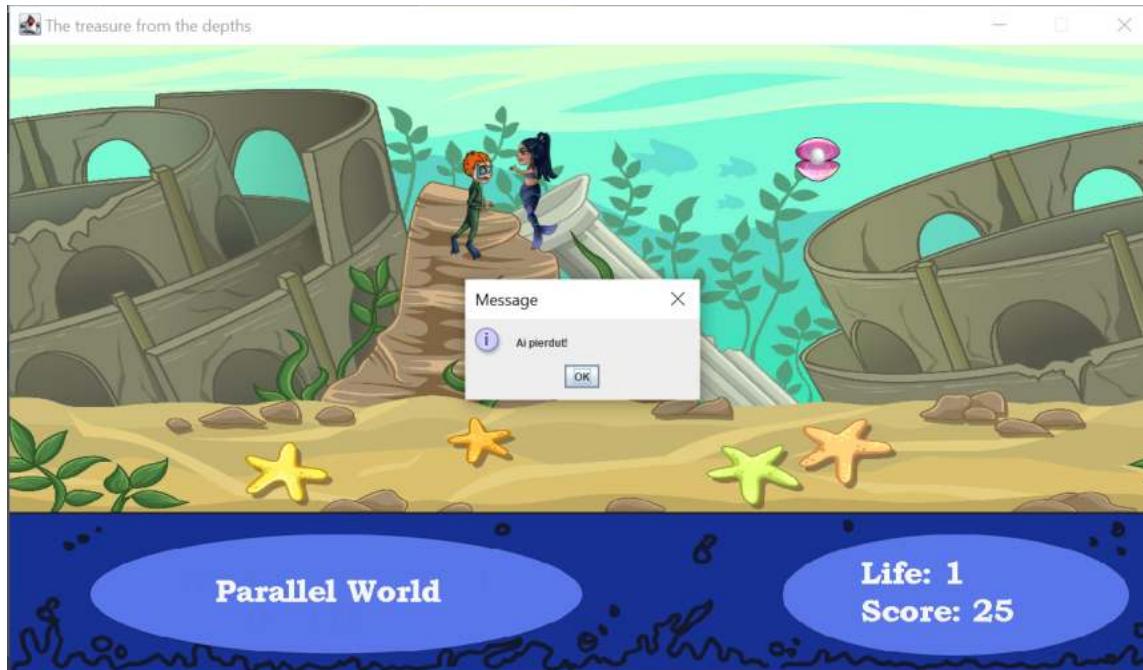
După colectarea unui peste (se scad 5 puncte)



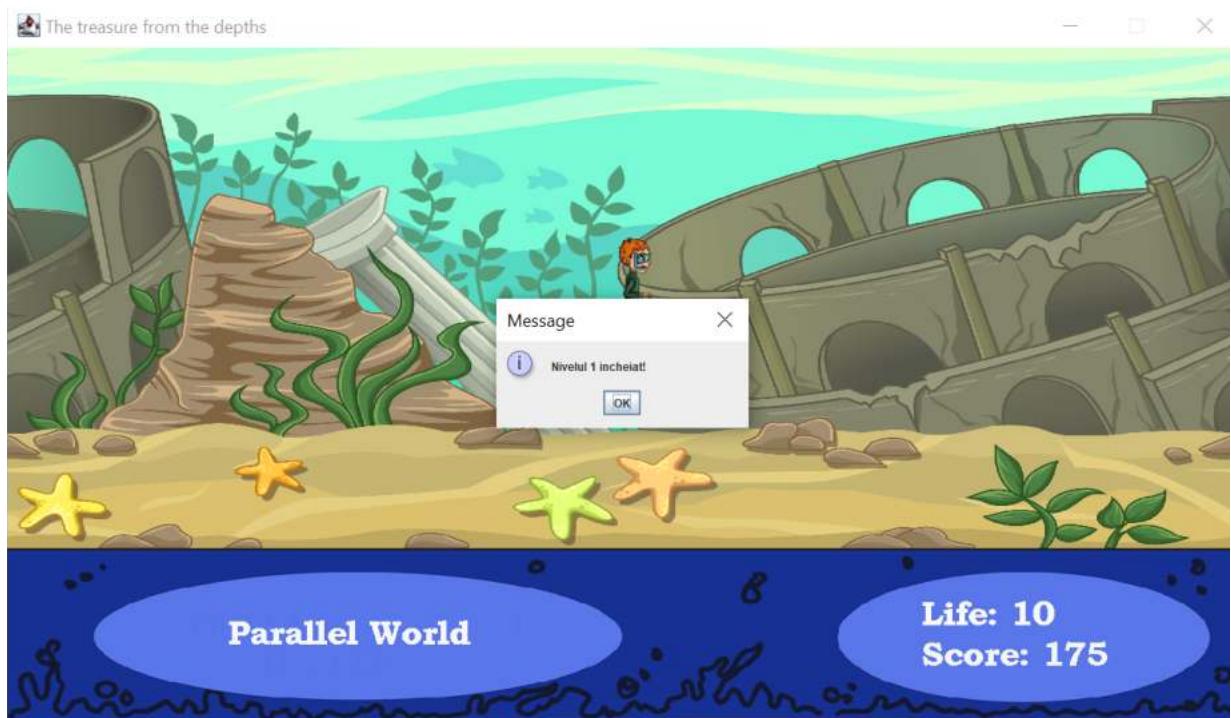
Aruncare sfere multiple



Pierderea jocului (s-au colectat 10 sirene și s-au pierdut toate viețile)

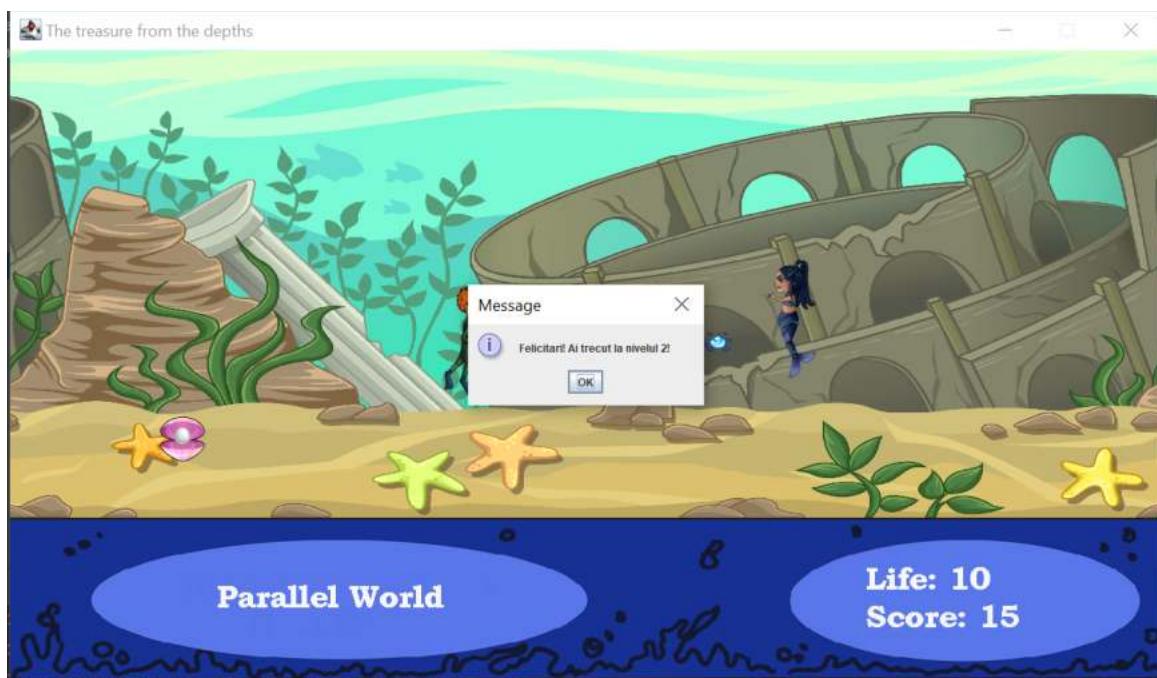
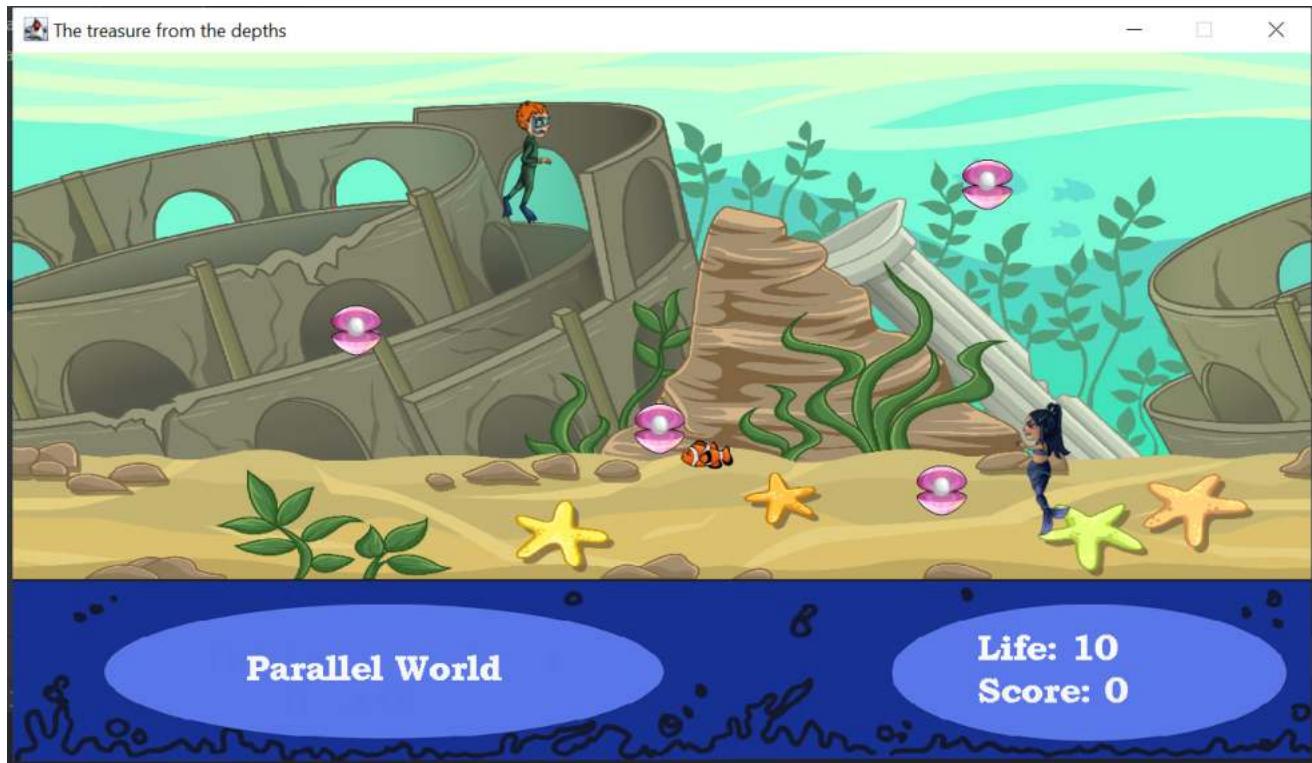


Castigarea nivelului (evitarea sirenelor)

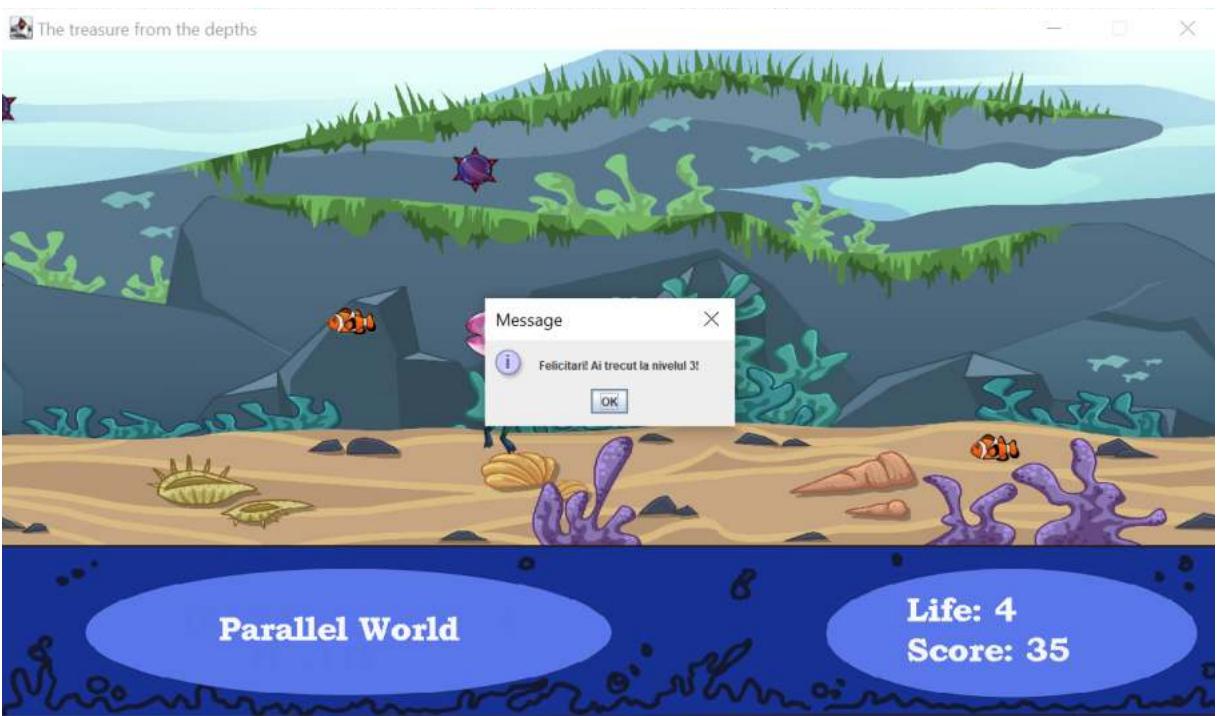


ETAPA 3

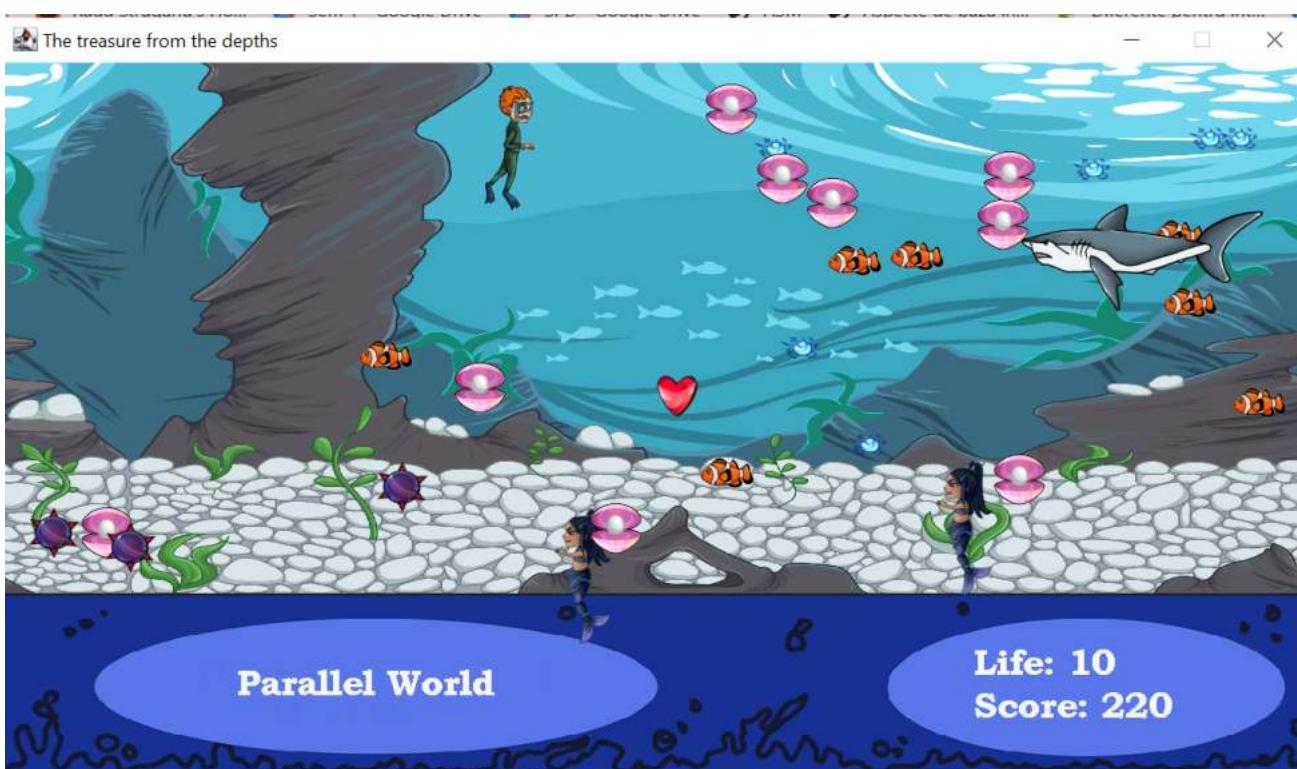
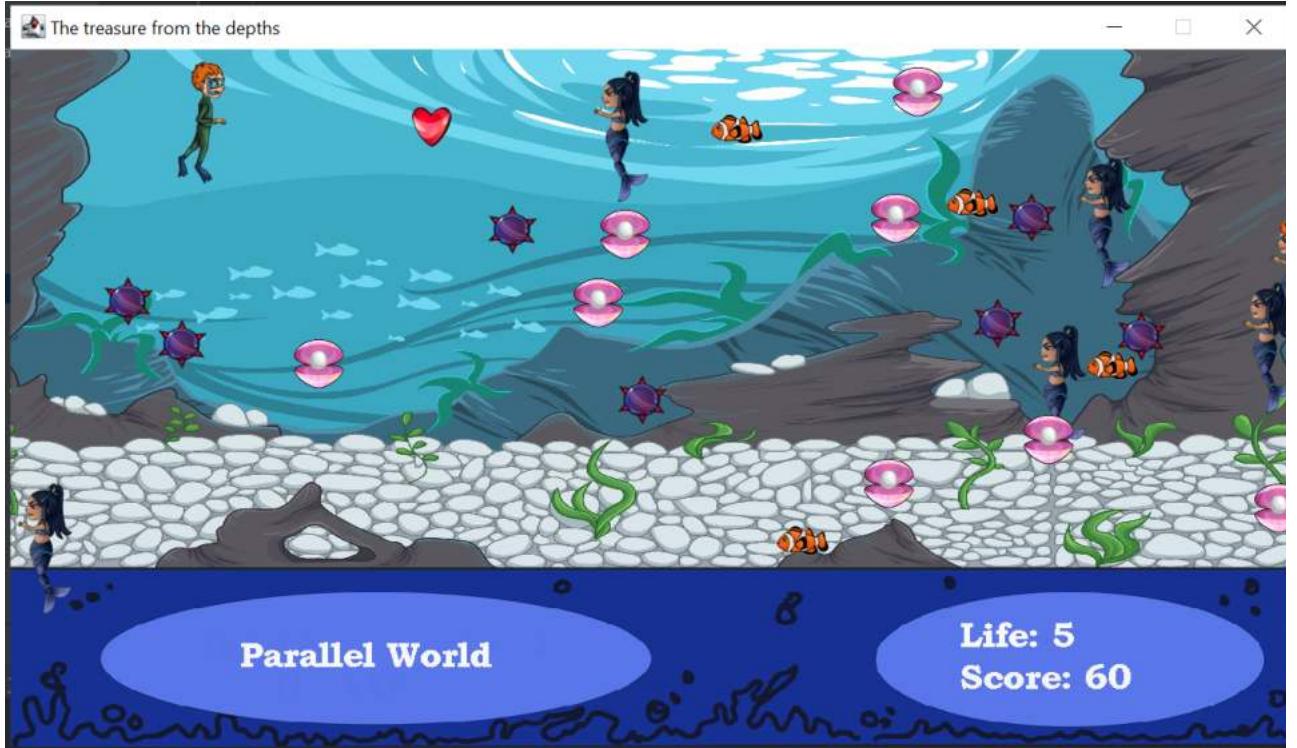
NIVEL 1:

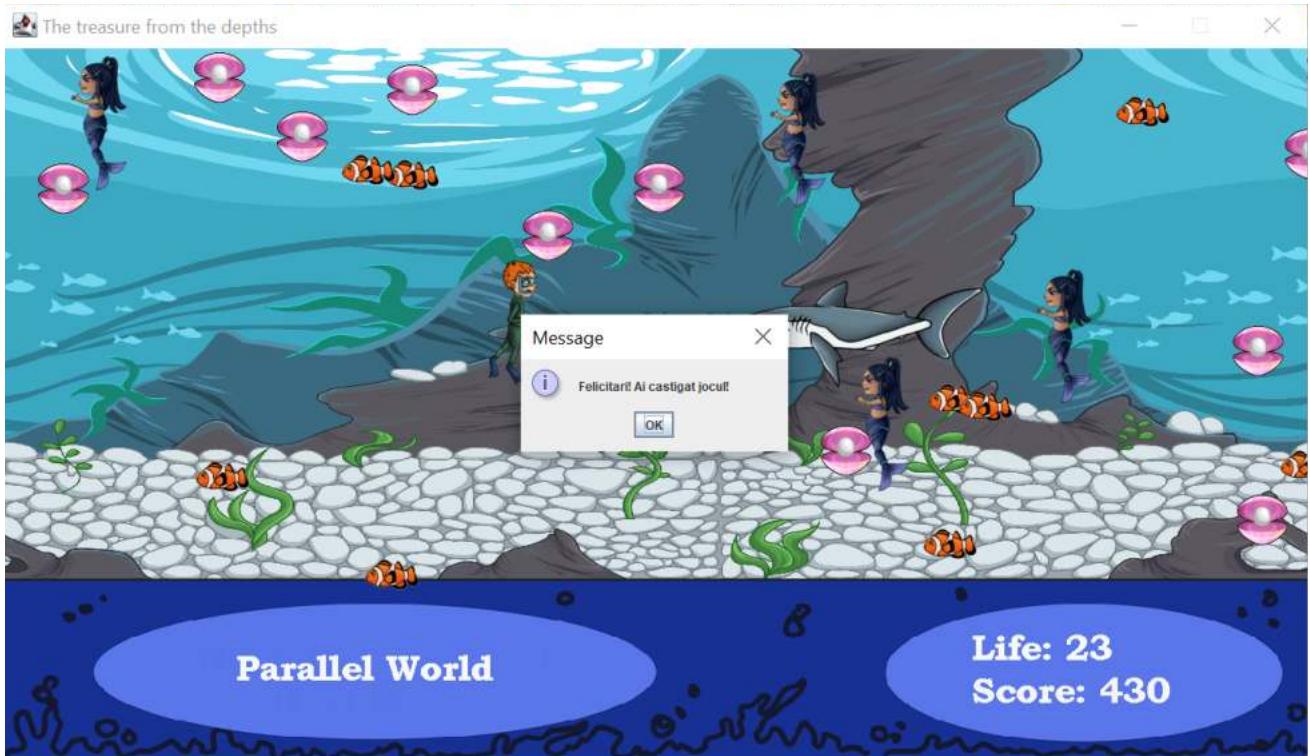


NIVEL 2:



NIVEL 3:





Mecanica jocului:

Jucătorul se va putea mișca astfel:

- În sus, prin intermediul tastei **sâgeata în sus ↑**
- În jos, prin intermediul tastei **sâgeata în jos ↓**
- În stanga, prin intermediul tastei **sâgeata la stânga <-**
- În dreapta, prin intermediul tastei **sâgeata la dreapta ->**
- Emite lovitura sferei mistice prin intermediul tastei **SPACE** pentru a ataca inamicii

Interacțiuni:

Personajul principal interacționează cu elementele pasive: perlele pe care trebuie să le adune(îi cresc scorul), inimile care îi sunt de folos (adună viață), dar îi scad scorul , peștii și sirenele de care trebuie să se ferească(îi scad scorul și îl pot nimici) și nu în ultimul rand de inamicul principal, rechinul ucigaș.

Fiecare perlă valorează +5 puncte la scor (la toate nivelele)

Fiecare pește valorează -5 puncte la scor (la toate nivelele)

Fiecare bombă valorează -5 vieți (apare începând cu nivelul 2)

Fiecare inimă valorează + 5 vieți și -30 puncte scor (apare la nivelul 3)

Fiecare sirenă eliminată valorează +10 puncte , dar dacă intră în coliziune cu una se scade o viață (prezenta în toate nivelele).

Inamicul final (prezent doar în nivelul 3) eliminat valorează +100 puncte , dar dacă intră în coliziune cu acesta se pierd toate viețile și jocul se pierde .

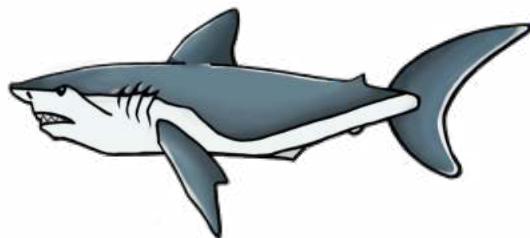
Jocul se pierde de asemenea dacă rechinuliese din ecran pe partea stanga.

Game sprite :

- Personajul principal :



- Personajul secundar (inamicul):



- Subalternii sirene:



Tileset-uri utilizate:

- Nivel 1



- Nivel 2

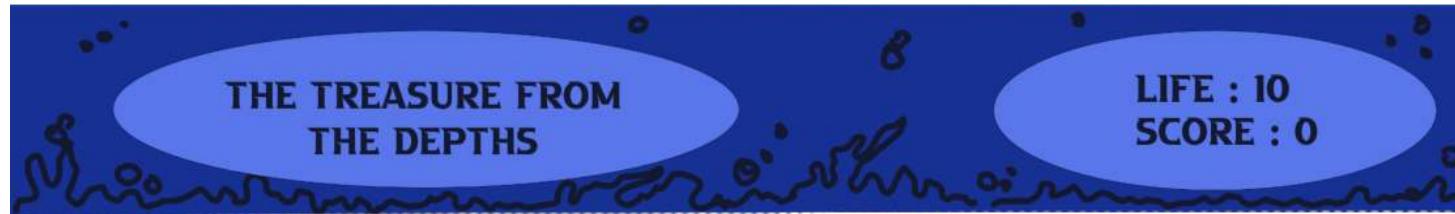


- Nivel 3



Descriere nivele:

Fiecare nivel are un fundal diferit (am ales 3 harti distincte), dar si un fundal comun in partea de jos cu doua casute. Casuta din stanga va indica numele jocului : The treasure from the depths , iar casuta din dreapta va indica viața și scorul personajului, acestea se vor actualiza constant.



Desenat in photoshop

Nivel 1

Fundalul:



În primul nivel al jocului, Orion, un scafandru înzestrat cu puteri magice, se scufundă pentru început la o adâncime mică în apa mării . Scopul principal este de

a aduna cât mai multe perle pentru a-i crește scorul, însă trebuie să fie vigilant. Îi ies în cale sirenele malefice și peștii răpitori care vor să-l opreasă cu orice preț. Cu ajutorul sferei sale mistice, Orion reușește să eliminate orice obstacol.

Nivelul este conceput pentru a testa abilitățile și reacțiile jucătorului în timp ce navighează în lumea subacvatica.

Nivel 2

Fundal:



Nivelul doi devine mai provocator, necesare fiind reflexe mai rapide și utilizarea strategică a sferei mistice.

Orion se confruntă cu noi provocări în timp ce își continuă aventura subacvatică. Golful pașnic de la nivelul 1 a fost înlocuit cu o lume subacvatică mai adâncă, mai întunecată, cu mai multe obstacole și inamici. Pentru a înrăutăți lucrurile, apar noi elemente : bombele. Dacă o bombă intră în contact cu personajul scorul va scădea cu 5 puncte, de aceea trebuie evitata cu orice preț.

Nivel 3

Fundal:



La nivelul 3, Orion ajunge în cea mai adâncă parte a oceanului, unde pândește legendarul rechin ucigaș. Aceasta este provocarea finală a aventurii sale subacvatice și trebuie să-și folosească toate abilitățile și puterile pentru a o depăși.

Rechinul este masiv și puternic, cu dinții ascuțiți, însă asta nu-l sperie pe Orion. Dacă Orion poate depăși toate aceste provocări și va învinge rechinul, el va ieși învingător și va finaliza aventura subacvatică. Cu toate acestea, dacă nu reușește să-l nimicească, va pierde jocul și va trebui să o ia de la capăt de la început.

Apare un nou element, un obiect strălucitor în forma de inima care îi oferă un bonus de 5 vieți, însă îi scade 30 de puncte din scor.

Nivelul final al jocului este un adevărat test al îndemânării și hotărârii jucătorului și doar cei mai pricepuți și dedicați jucători vor putea ieși castigatori. Dacă jucătorul poate stăpâni acest nivel, ei se vor dovedi ca adevărați aventurieri subacvatici.

Diferente clare intre niveluri:

- 3 harti **diferite** (imagini care rulează de la dreapta la stanga pentru a crea ideea de mișcare)
- **Elemente noi** in fiecare nivel :

Nivel 1: Scafandru + Sirena + Scoica + Peste

Nivel 2: Pe langa acestea + Bomba (*care se deplaseaza cu o viteza mai mare decat celelalte elemente), putand fi considerata ca un inamic, deoarece îi scade cate 5 puncte din viața jucătorului și el trebuie sa se fereasca de ele

Nivel 3: Pe langa acestea + Inima + Rechin(inamic final care trebuie lovit de 4 ori și nu doar o data ca și ceilalți inamici)

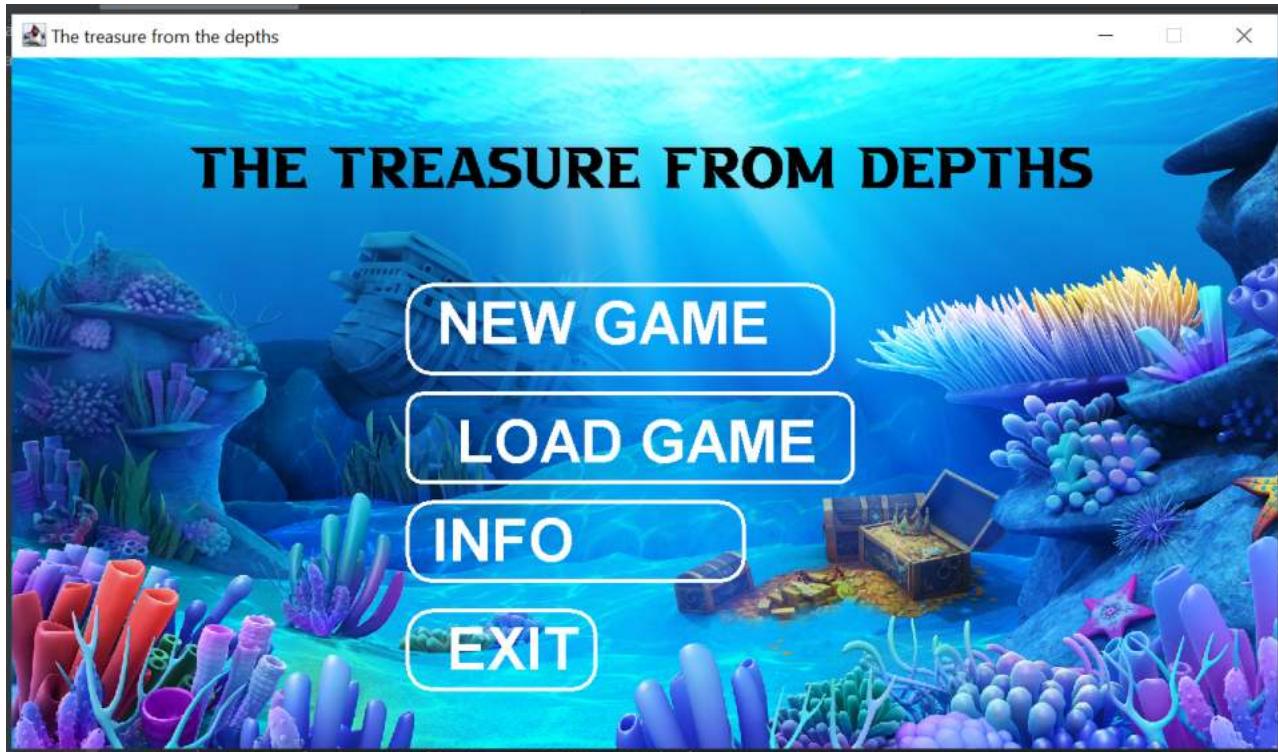
- Numărul crescut de elemente de care personajul trebuie sa se fereasca și viteza cu care se deplaseaza (crește dificultatea)

Meniu:

Exista 3 meniuri

Meniul principal (apare cand rulam jocul) are următoarele butoane:

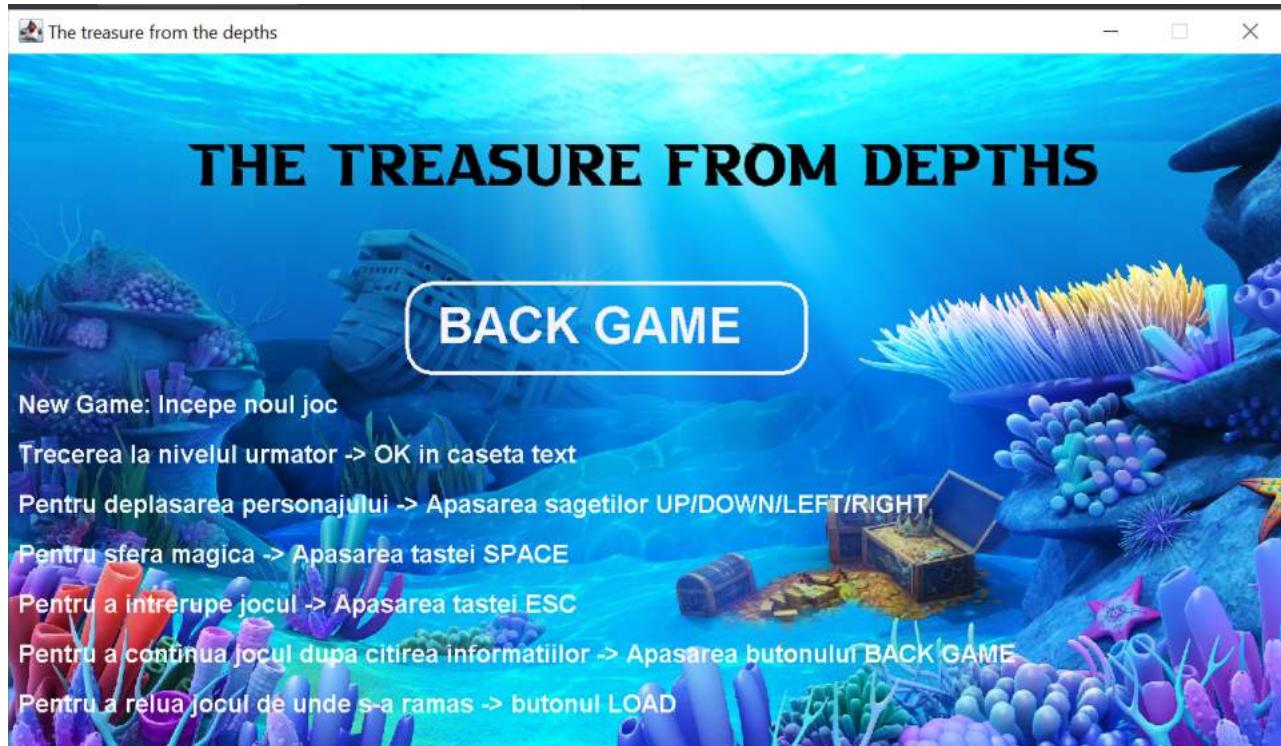
- **NEW GAME**-> la apăsarea acestuia, începe jocul. Nivelurile sunt progresive(dificultatea crește de la nivel la nivel)
- **LOAD GAME** -> la apasarea acestuia reluăm jocul de unde am rămas, scorul și viața păstrate , la fel și poziția personajului
- **INFO** -> la apăsarea acestuia, se va intra în cel de al doilea meniu, cel cu informații despre joc
- **EXIT** -> la apăsarea acestuia, se va părăsi jocul



Meniul cu informații (apare cand apasam pe INFO):

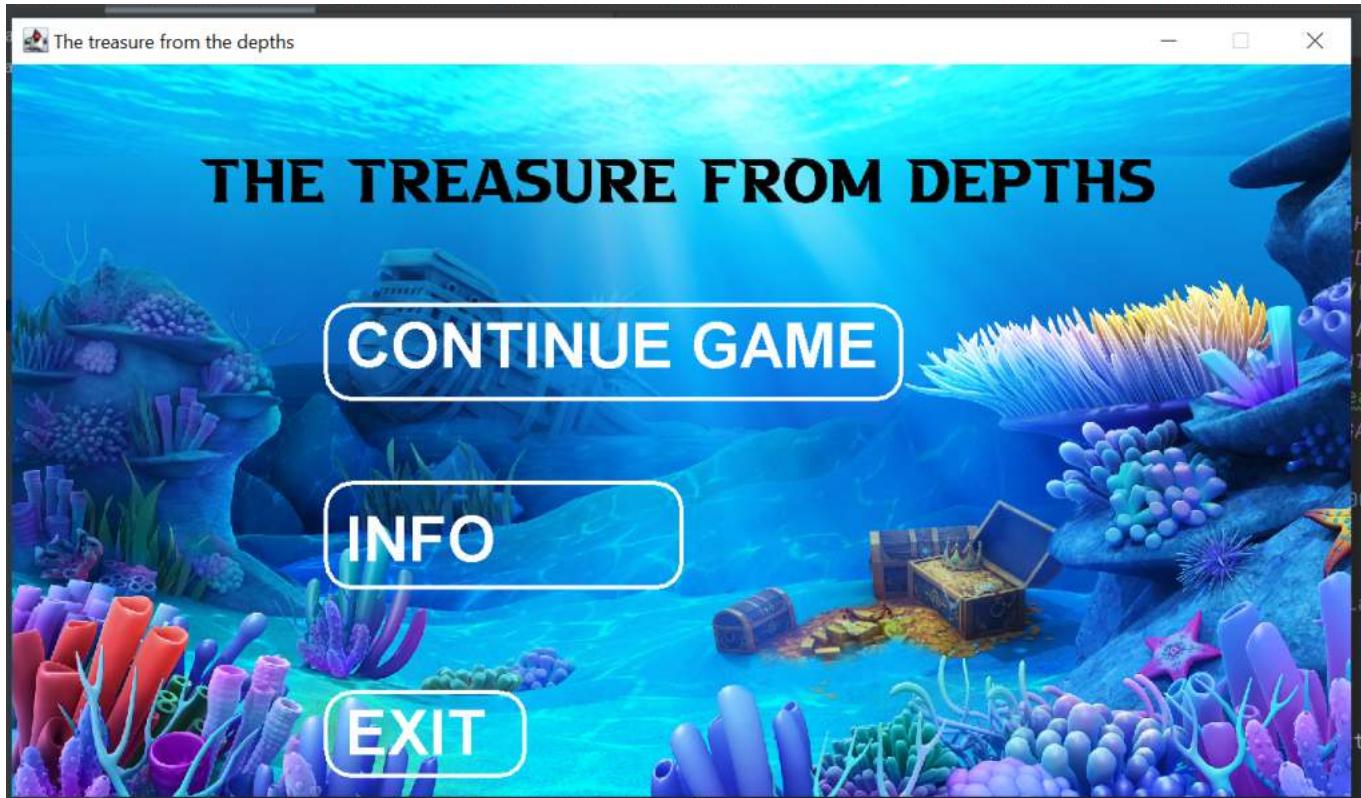
Acest meniu cuprinde informatii cu privire la joc + taste . Are ca buton:

- **BACK GAME** -> la apăsarea acestuia, utilizatorul este trimis înapoi la meniul principal de unde poate relua jocul



Meniul pentru continuarea jocului (apare cand apasam ESC) va avea butoanele:

- **CONTINUE GAME** -> la apăsarea acestuia, va continua jocul de la momentul rămas la apăsarea tastei ESC sau dacă se accesează din meniul de informații va începe un nou joc
- **INFO** -> la apăsarea acestuia, se va intra în meniul cu informații despre joc
- **EXIT** -> la apăsarea acestuia, se va parasi jocul



DESCRIEREA CLASELOR:

Clasa **SCAFANDRU** este responsabilă de gestionarea protagonistului în joc. Extinde clasa JComponent pentru a permite adăugarea acesteia în panoul de joc. Este definită dimensiunea personajului, viteza, poziția pe ecranul jocului. De asemenea, apar și key events, cum ar fi tastele săgeți(sus , jos , stanga , dreapta) și bara de spațiu(enter), pentru a controla mișările și atacul personajului.

Metoda keyPressed actualizează viteza caracterului în funcție de ce tastă săgeată este apăsată, în timp ce metoda keyReleased setează viteza înapoi la zero atunci când tasta este eliberată. Metodele setYDirection și setXDirection actualizează viteza personajului în direcția corespunzătoare. Metoda move actualizează poziția personajului în funcție de viteza acestuia și asigură că personajul rămâne în limitele ecranului de joc.

Metoda getySCAFANDRU returnează coordonatele y a caracterului, în timp ce metoda setySCAFANDRU o actualizează. Metoda draw desenează personajul pe ecranul jocului folosind o imagine a personajului definit în clasa main.

Clasa **ENEMY** este responsabilă de gestionarea inamicilor, respectiv sirenele, în joc. Este observat comportamentul inamicilor, aceștia sunt generați aleatoriu și se îndreaptă către scafandru, iar coliziunile dintre inamici și acesta au ca rezultat o scădere a vieții jucătorului.

Variabilele statice sunt „NumarSirene” și „MultimeSirene”. „NumarSirene” este o variabilă care servește drept contor, în timp ce „MultimeSirene” este o listă `LinkedList` care stochează instanțe ale clasei „Enemy”.

Variabilele „xEnemy” și „yEnemy” reprezintă coordonatele unui obiect inamic.

Constructorul clasei „Inamic” ia doi parametri întregi care reprezintă coordonatele x și y ale obiectului inamic. Setează variabilele de instanță la valorile parametrilor și actualizează valoarea variabilei statice „NumarSirene” la dimensiunea listei „MultimeSirene”.

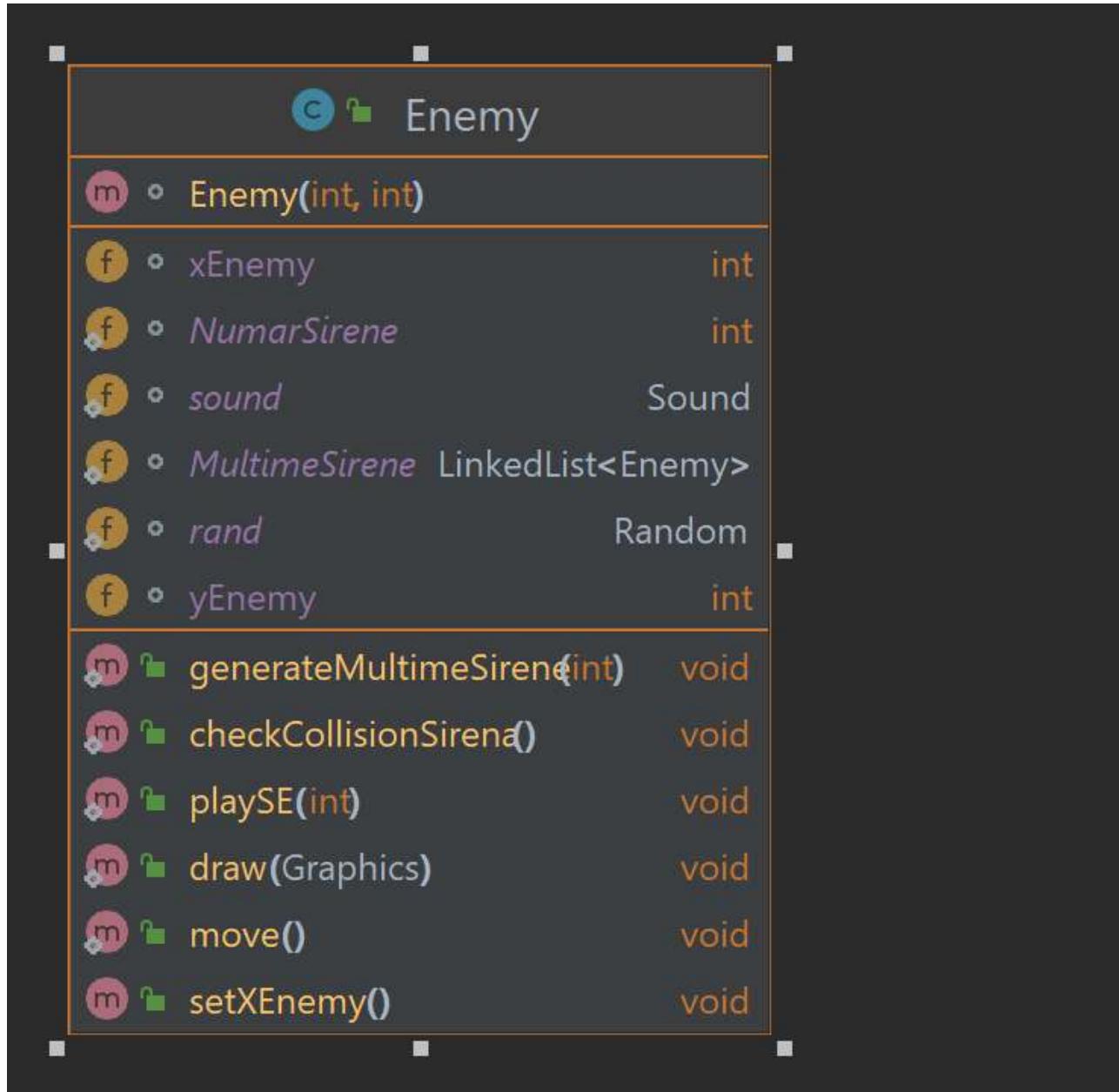
Metoda „generateMultimeSirene” generează un număr specificat de obiecte inamice aleatoriu în limitele unei ferestre de joc. Aceasta creează instanțe noi ale clasei „Inamic” și le adaugă la lista „MultimeSirene”.

Metoda „draw” ia un obiect `Graphics` ca parametru și îl folosește pentru a desena fiecare obiect inamic din lista „MultimeSirene” în poziția sa curentă.

Metoda „move” mută fiecare obiect inamic din lista „MultimeSirene” la stânga cu 2 pixeli. Dacă un obiect inamic se mișcă din partea stângă a ecranului, acesta este eliminat din lista „MultimeSirene” și contorul „NumarSirene” este decrementat.

Metoda „checkCollisionSirena” verifică dacă există coliziuni între obiectele inamice și un obiect de scafandru din joc. Utilizează distanța dintre centrele celor două obiecte și razele acestora pentru a determina dacă a avut loc o coliziune. Dacă are loc o coliziune, obiectul inamic este eliminat din lista „MultimeSirene”, contorul „NumarSirene” este decrementat și metoda „subtractLife” a clasei „Life” este apelată pentru a reduce viața jucătorului cu 1.

În general, acest cod reprezintă comportamentul inamicilor într-un joc cu derulare laterală, în care aceștia sunt generați aleatoriu și se îndreaptă către personajul jucătorului, iar coliziunile dintre inamici și jucător au ca rezultat o scădere a vieții jucătorului.

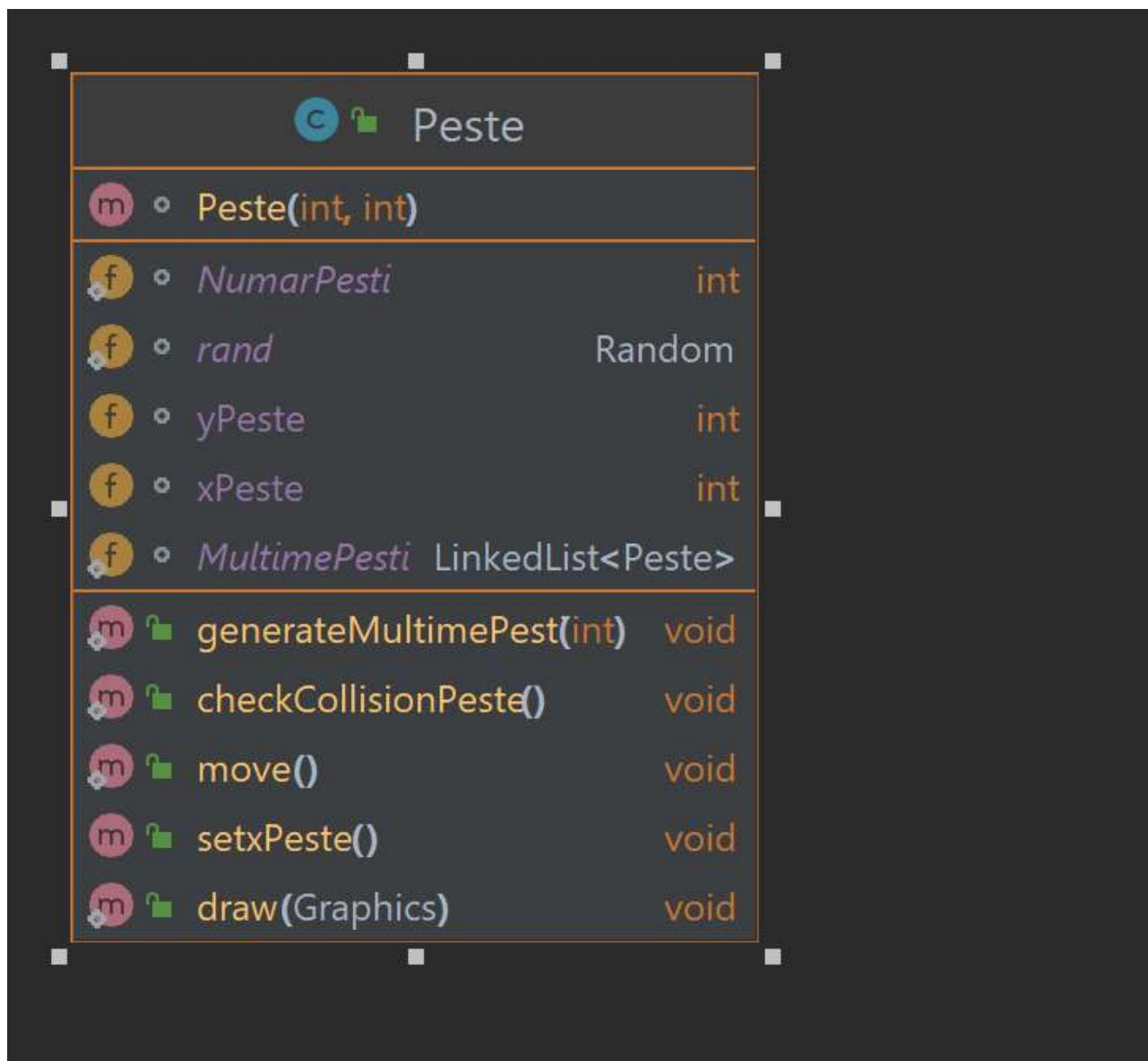


Clasa **PESTE** e responsabilă pentru crearea și gestionarea caracterelor de tip peste în joc.

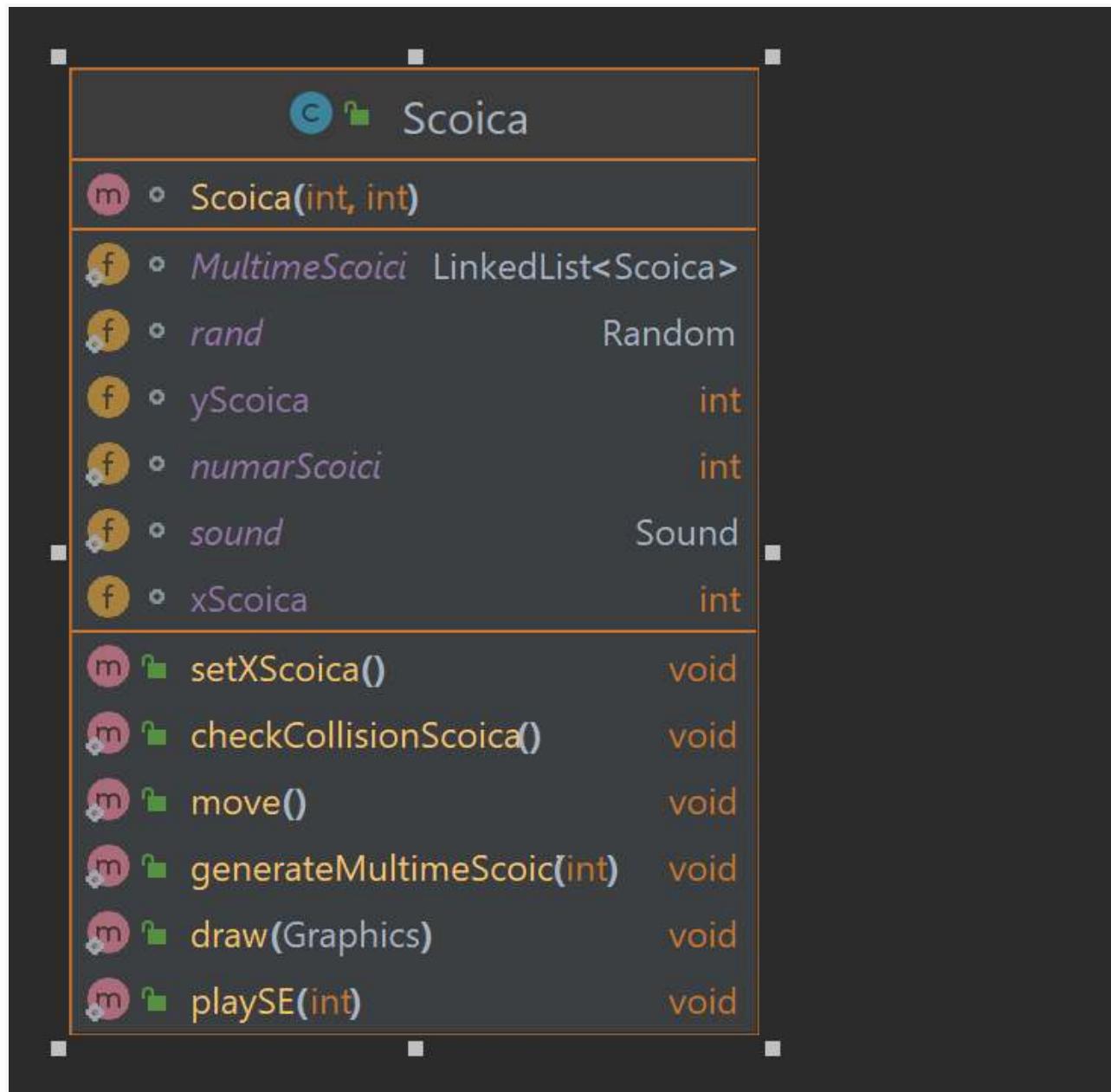
Clasa Are o variabilă statică „NumarPesti” care numără numărul de pești creați și o listă „MultimePesti” pentru a stoca toate obiectele create. Clasa are, de asemenea, un constructor pentru a crea un obiect pește cu coordonatele x și y.

Există trei metode în această clasă, „generateMultimePesti” pentru a crea un număr specificat de obiecte de pește și pentru a le adăuga la lista legată, „draw” pentru a desena toate obiectele de pește de pe ecran și „move” pentru a muta obiectele în partea stângă a ecranului și elibera dacă ieș de pe ecran.

Clasa are și o metodă „checkCollisionPeste” care detectează coliziunea dintre peste și protagonist. Dacă există o coliziune, peștele dispare, contorul scade și scorul este scăzut cu 5 puncte.



Clasa **SCOICA** e responsabilă pentru crearea și gestionarea obiectelor de tip scoica în joc. Are metode pentru generarea, desenarea, mutarea (scoicilor) și verificarea coliziunii acestora cu protagonistul. Obiectele de tip scoica au x și y ca și coordonate, sunt generate random în fereastra jocului și se mută constant la stanga pana ies din ecran. Cand intra în contact cu scafandrul , scoica dispare, contorul scade , iar scorul jucătorului crește cu 5 puncte.



Clasa **BULLET** descrie puterea scafandrului, albastra sferă mistică. Ea conține variabile pentru coordonatele x și y ale sferei, o listă pentru a stoca toate sferele generate și metode pentru adăugarea și desenarea sferelor pe ecran, mutarea acestora și verificarea coliziunilor cu inamicii.

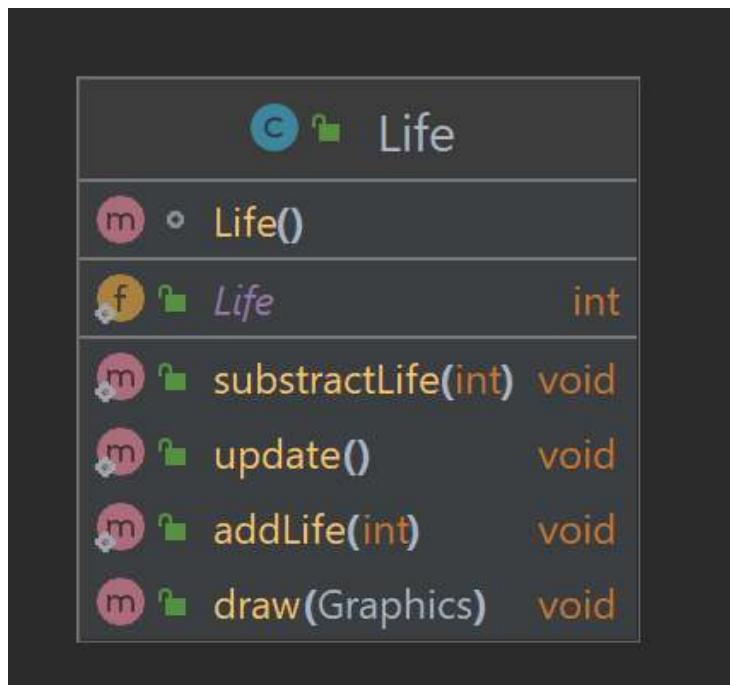
Clasele **SCORE** și **LIFE** țin evidența vieții și a scorului jucătorului.

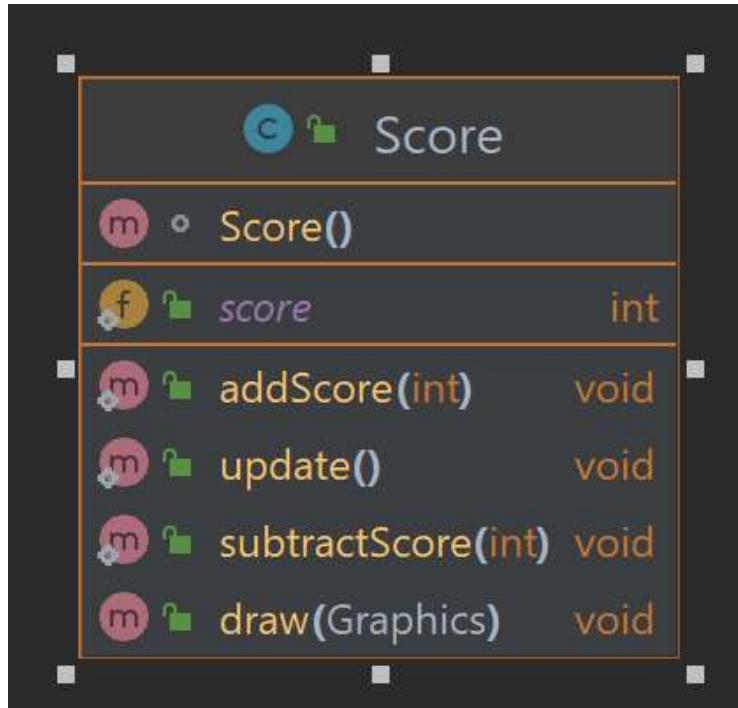
Clasa Life are o variabilă întreagă statică numită Life (initial 10), care reprezintă durata de viață inițială a jocului și metode de a adăuga sau scădea din valoarea de viață. Metoda

draw ia un obiect de tip Graphics ca parametru și îl folosește pentru a desena valoarea de viață pe panoul de joc la anumite coordonate folosind un font și o culoare specificate.

Clasa Score are o variabilă întreagă statică numită scor (initial 0), reprezentând scorul inițial al jocului și metode de adunare sau scădere a scorului. Metoda draw ia un obiect de tip Graphics ca parametru și îl folosește pentru a desena titlul jocului și valoarea punctajului pe panoul jocului la anumite coordonate folosind fonturi și culori specificate.

Ambele clase folosesc clasa Color pentru a specifica culoarea textului și clasa Font pentru a specifica fontul folosit pentru text. De asemenea, folosesc coordonate specifice pentru a plasa textul pe panoul de joc.

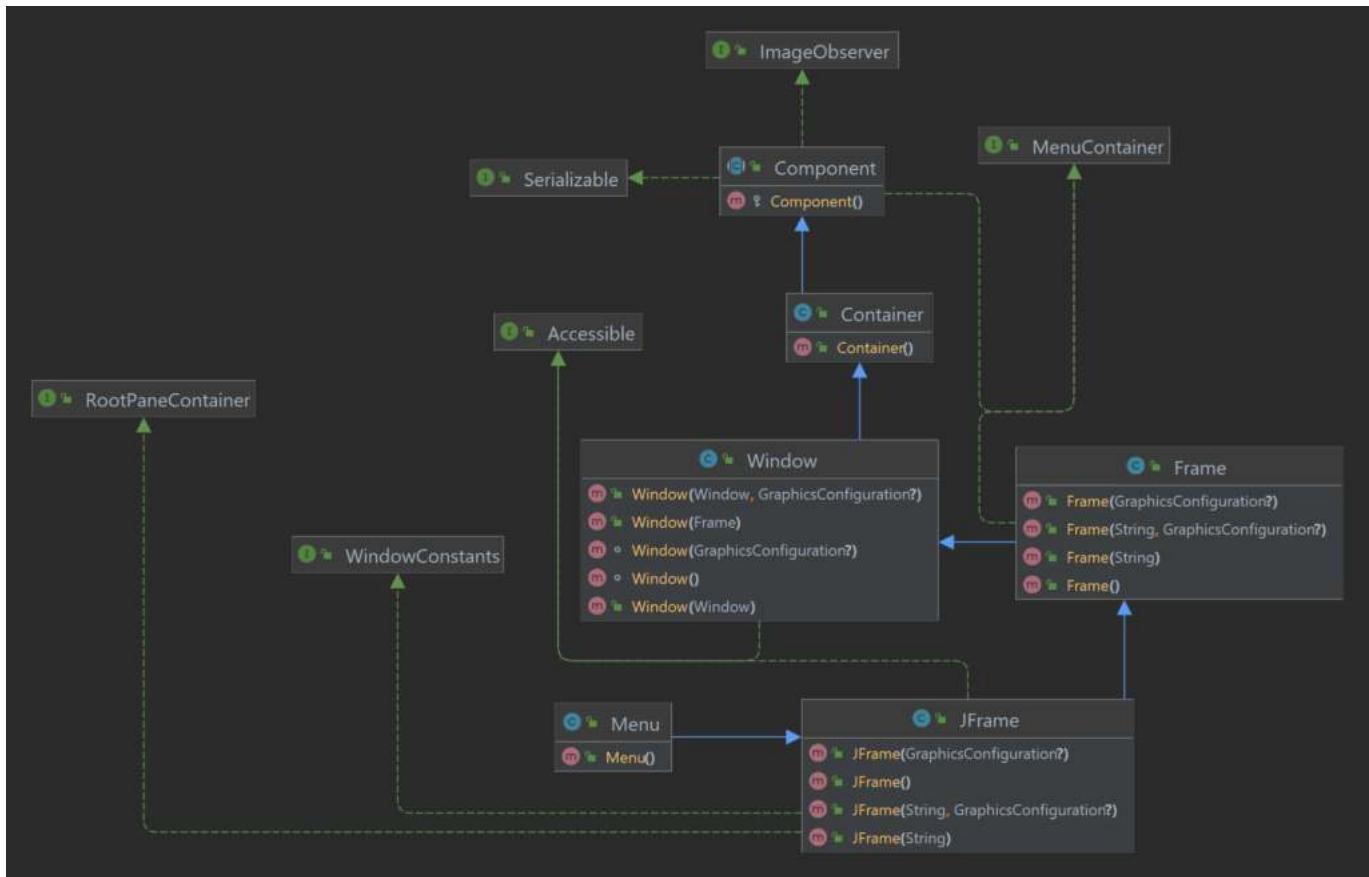




Clasele **MENU** și **MENU2** sunt responsabile pentru desenarea meniurilor (cel principal și cel secundar) din joc.

Meniul de primă clasă extinde `JFrame` și suprascrie metoda `paint` pentru a desena butoanele folosind obiecte `RoundRectangle2D`. Coordonatele fiecărui buton sunt stabilite și apoi utilizate pentru a desena textul și forma butonului. Fontul, culoarea și conturul obiectului grafic sunt, de asemenea, setate înainte de desenarea fiecărui buton.

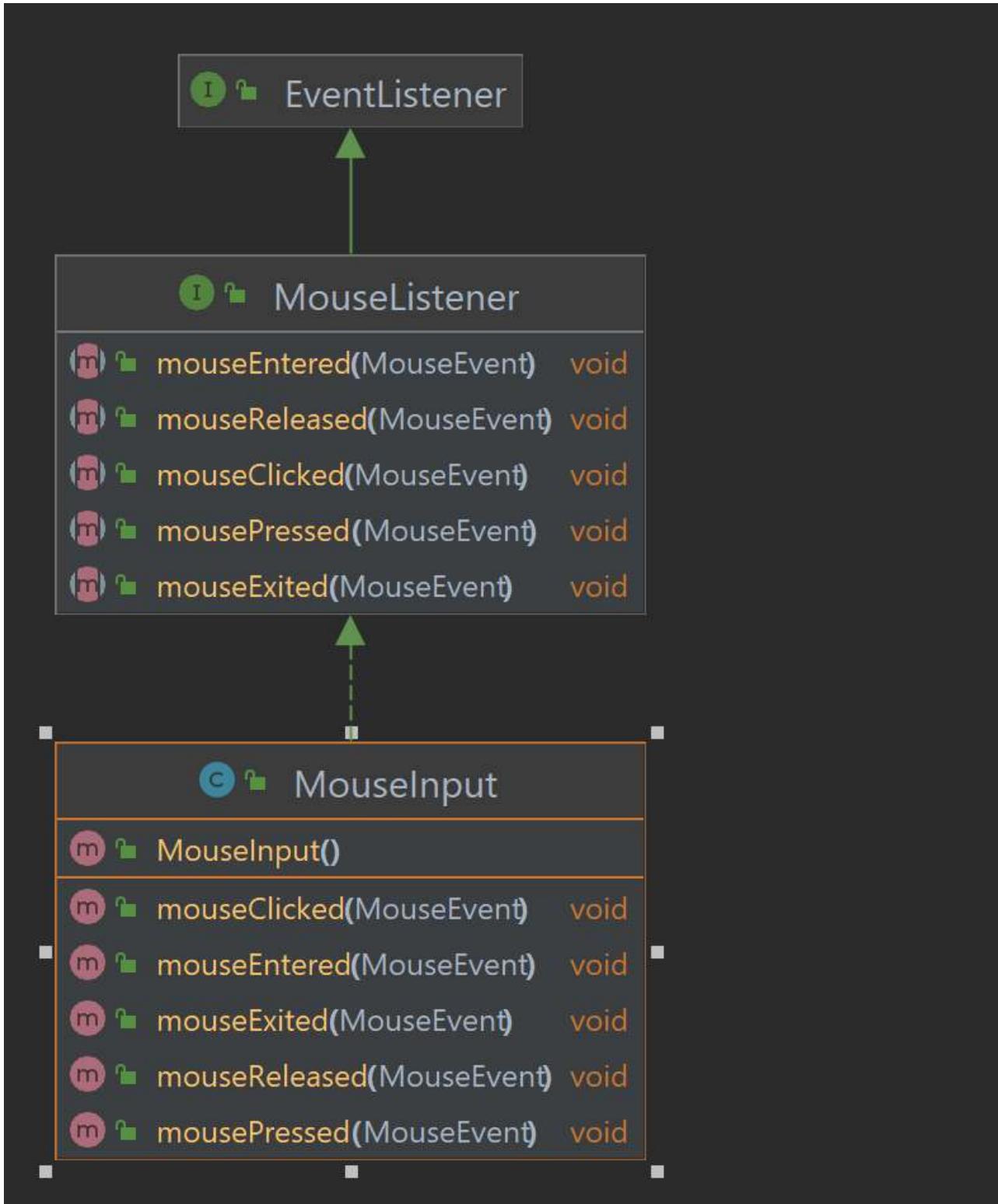
A doua clasă `Menu2` definește o metodă numită `draw` care preia un obiect `Graphics` și desenează butoanele folosind obiecte `RoundRectangle2D`. Similar cu prima clasă, coordonatele fiecărui buton sunt stabilite și utilizate pentru a desena textul și forma butonului. Fontul, culoarea și conturul obiectului grafic sunt, de asemenea, setate înainte de desenarea fiecărui buton.



Clasa **MouseListener** are metode care permit raspunderea evenimentelor mouse-ului, cum ar fi atunci când butonul mouse-ului este apăsat, apăsat, eliberat sau când mouse-ul intră sau ieșe dintr-o componentă.

Se suprascriu cele cinci metode ale interfeței MouseListener. Metoda mouseClicked() este apelată când se face clic pe butonul mouse-ului. Metodele mouseEntered() și mouseExited() sunt apelate atunci când mouse-ul intră sau, respectiv, ieșe dintr-o componentă. Metoda mousePressed() este apelată atunci când este apăsat un buton al mouse-ului. În cele din urmă, metoda mouseReleased() este apelată atunci când un buton al mouse-ului este eliberat.

Metoda mouseReleased() conține unele funcționalități suplimentare. Verifică coordonatele x și y ale click-ului mouse-ului folosind metodele getX() și getY() ale clasei MouseEvent. Apoi verifică dacă coordonatele sunt în anumite limite și, dacă da, schimbă starea unui obiect GamePanel în GAME sau ieșe din program folosind System.exit(1).



Clasa **GamePanel** extinde JPanel și implementează interfața Runnable. Clasa conține diverse proprietăți și metode pentru gestionarea mediului de joc și redarea graficii.

Clasa conține o enumerare numită STATE, care definește stările posibile ale jocului. Stările posibile sunt MENU, GAME și EXIT.

Clasa are mai multe proprietăți, inclusiv un obiect Clip numit sunet, două obiecte Menu numite menu și menu2, un obiect STATE numit State, un obiect Thread numit gameThread, un obiect Score numit scor, un obiect Life numit viață, un obiect Scafandru numit scafandru , un obiect Image numit imagine și un obiect Graphics numit graphics.

Clasa are un constructor care inițializează mediul de joc, adaugă un KeyListener, adaugă un MouseListener, setează dimensiunea preferată a panoului, creează un nou obiect Thread și apelează metoda generateEnvironment() pentru a inițializa mediu de joc.

Metoda generateEnvironment() inițializează mediul de joc prin generarea de inamici, crearea unui nou obiect Scafandru, crearea unui nou obiect Score, crearea unui nou obiect Life și generarea de scoici și pești.

Metoda paint() este responsabilă pentru redarea graficii jocului. Metoda apelează metoda draw(), care desenează fundalul, obiectul jucător, obiectul scor și obiectul life.

Metoda draw() are două condiții pentru diferitele stări ale jocului. Dacă starea este GAME, metoda desenează imaginile de fundal principale și secundare, obiectul jucător, obiectul scor și obiectul life. Dacă starea este MENU, metoda desenează fundalul meniului și noul meniu de joc. Dacă starea este EXIT, metoda desenează fundalul meniului și meniul de ieșire.

Clasa **GAME** creează o interfață grafică cu utilizatorul. Programul creează un JFrame și îl adaugă un GamePanel.

Constructorul clasei Game stabilește proprietățile JFrame-ului: titlul său, dacă poate fi redimensionat sau nu și cum ar trebui să se comporte când este închis. Apoi dimensionează JFrame pentru a se potrivi conținutului său și îl setează să fie vizibil. La final centrează JFrame pe ecran.

Clasa **BOMBA** se ocupă de gestionarea bombei și interacțiunile acesteia cu jucătorul. Variabilele numarBombe și MultimeBombe sunt utilizate pentru a ține evidența numărului de bombe create și a bombelor existente în joc. Variabila numarBombe este un

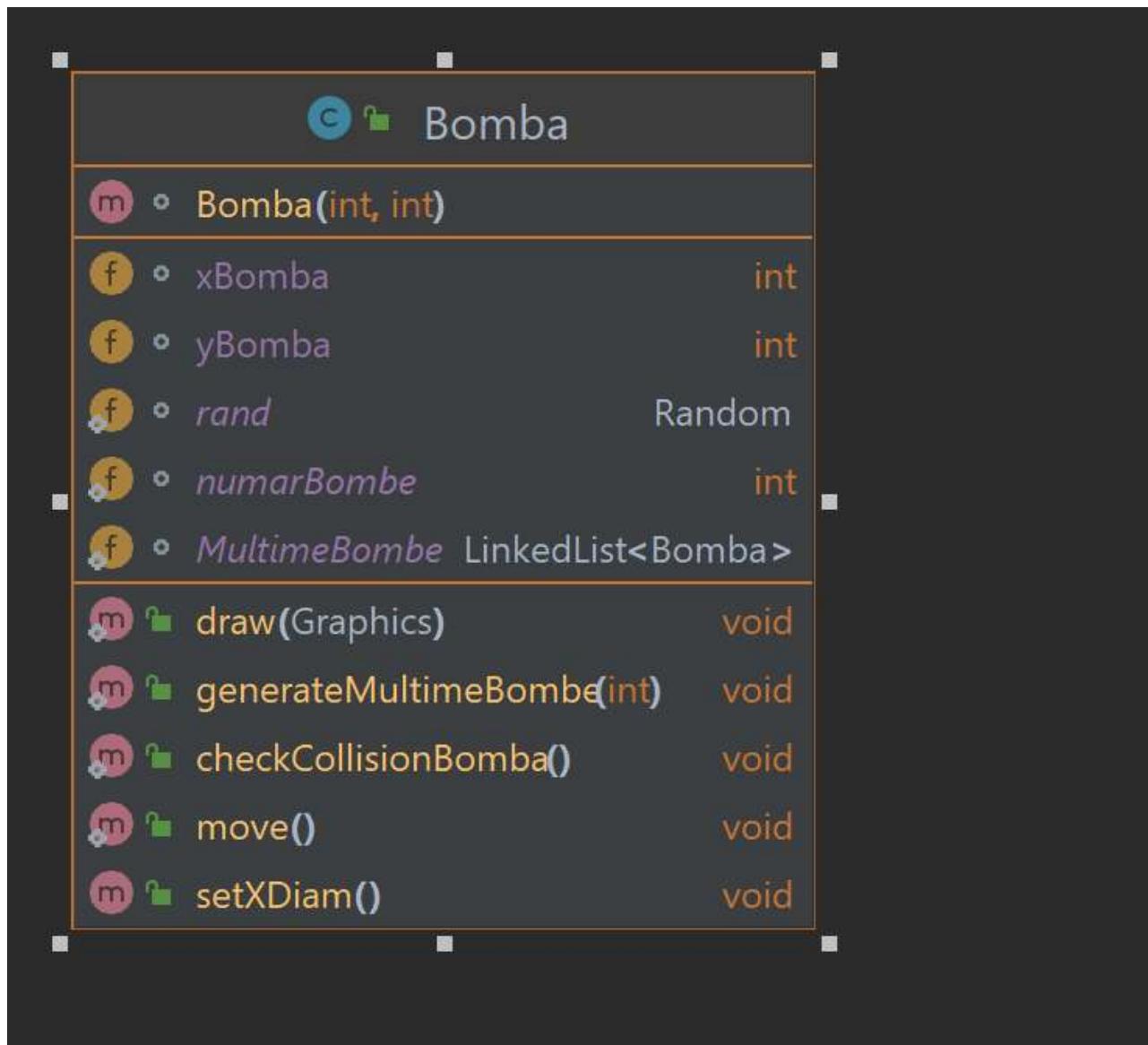
contor care reprezintă numărul de obiecte de tip bomba create, în timp ce MultimeBombe este o listă în care se stochează toate obiectele de tip bomba create.

Metoda generateMultimeBombe(int nr) este responsabilă de generarea bombei și adăugarea acesteia în mulțimea MultimeBombe. Această metodă primește un număr nr și creează bombe în interiorul ferestrei de joc, generând coordonatele lor aleatoriu.

Metoda draw(Graphics g) desenează bombele pe ecran utilizând obiectul Graphics. Pentru fiecare obiect de tip Bomba din mulțimea MultimeBombe, se desenează bomba pe coordonatele sale specifice.

Metoda move() gestionează mișcarea bombelor pe ecran prin actualizarea coordonatei x a fiecărei bombe. Astfel, coordonata x a fiecărei bombe este redusă cu o valoare specificată, ceea ce duce la deplasarea lor spre stânga pe ecran.

Metoda checkCollisionBomba() verifică coliziunile dintre jucător (scafandru) și bombe. Se calculează distanța între centrul fiecărei bombe și centrul scafandrului, iar apoi se compară această distanță cu suma valorilor razei pentru detectarea coliziunii. În cazul unei coliziuni, bomba este eliminată din mulțimea MultimeBombe, numărul de bombe scade, iar jucătorul pierde 5 vieți.



Clasa **INIMA** este responsabilă de gestionarea inimilor în joc și interacțiunile acestora cu jucătorul. Aceasta stochează coordonatele inimii, numărul total de inimi create și o listă în care sunt păstrate obiectele de tip Inima.

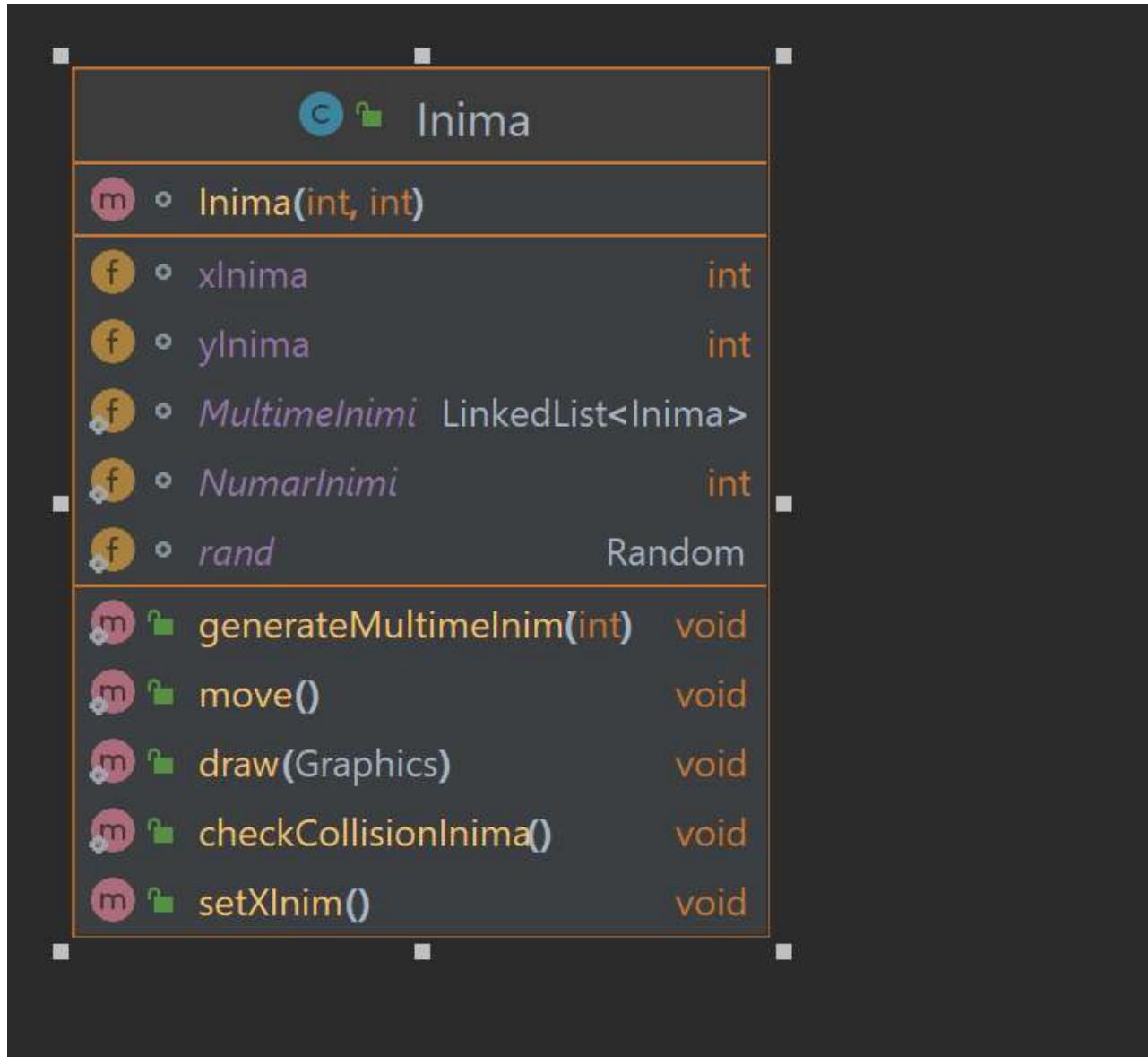
Metoda `generateMultimeInimi` generează inimi în interiorul ferestrei de joc. Aceasta primește un număr nr și adaugă obiecte de tip Inima cu coordonate aleatoare în lista `MultimeInimi`.

Metoda `draw` desenează inimile pe ecran. Se iterează prin fiecare obiect Inima din lista `MultimeInimi` și se utilizează obiectul `Graphics2D` pentru a desena inima pe coordonatele specificate.

Metoda move gestionează mișcarea inimilor pe ecran. Se actualizează coordonata x a fiecarei inimi prin apelul metodei setXInim. În plus, se verifică dacă vreo inimă a ieșit în afara ecranului la stânga. În acest caz, inima respectivă este eliminată din lista MultimeInimi, iar variabila NumarInimi este actualizată.

Metoda setXInim actualizează coordonata x a inimii, determinând astfel viteza de deplasare a inimii pe ecran.

Metoda checkCollisionInima verifică coliziunile dintre jucător (scafandru) și inimi. Se calculează centrul inimii și centrul scafandrului, iar apoi se calculează distanța dintre aceste puncte. Dacă distanța este mai mică sau egală cu suma valorilor razei, înseamnă că a avut loc o coliziune. În acest caz, inima este eliminată din lista MultimeInimi, variabila NumarInimi este actualizată, jucătorul primește 5 vieți, iar scorul este scăzut cu 30 de puncte.



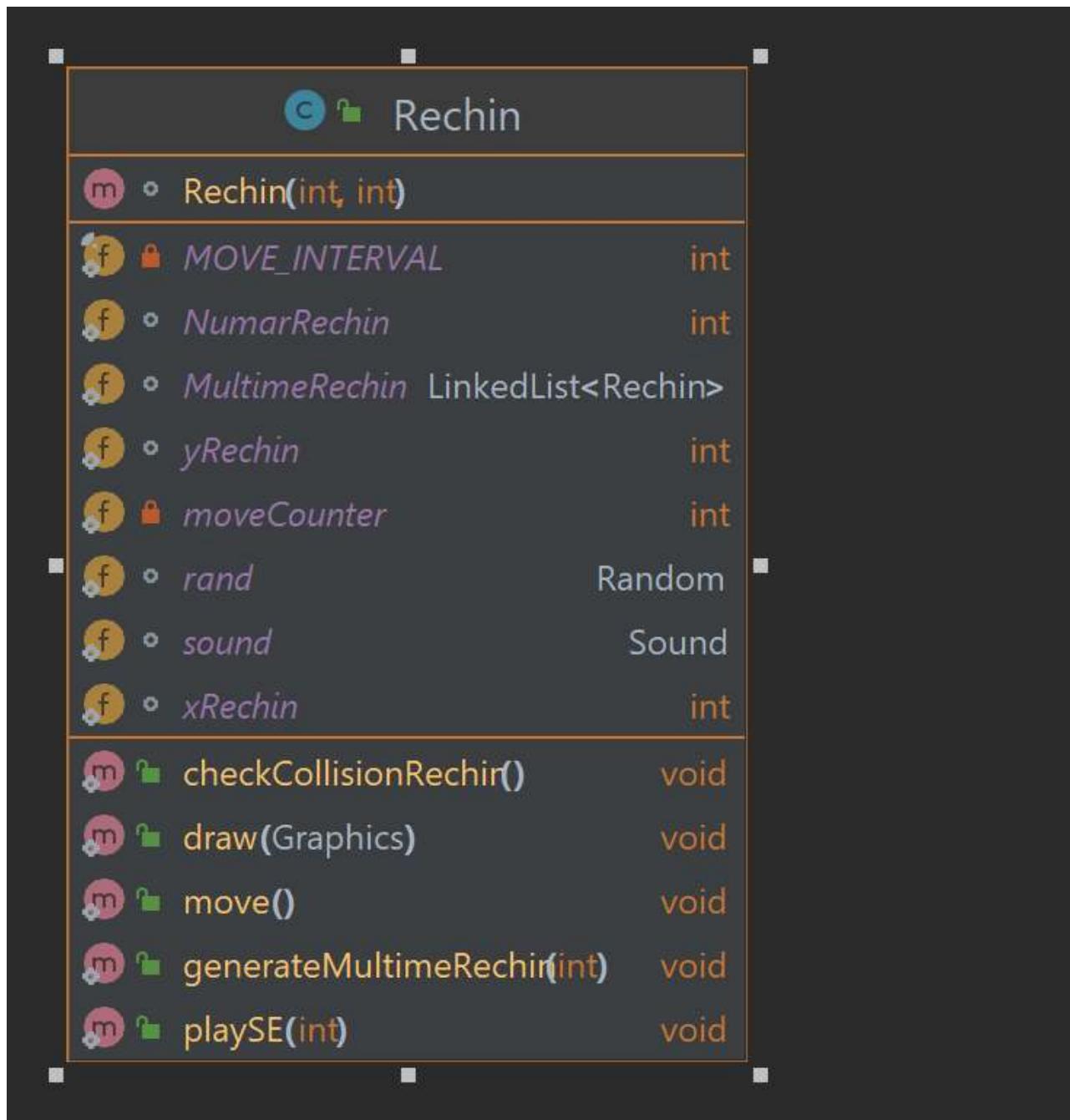
Clasa **Rechin** este responsabilă de gestionarea rechinului final în joc și interacțiunile acestuia cu jucătorul. Aceasta stochează coordonatele rechinului, numărul total de rechini create și o listă în care sunt păstrate obiectele de tip Rechin.

Metoda **generateMultimeRechin** generează rechini în interiorul ferestrei de joc. Aceasta primește un număr nr și adaugă obiecte de tip Rechin cu coordonate aleatoare în lista **MultimeRechin**.

Metoda **draw** desenează rechinii pe ecran. Se iterează prin fiecare obiect Rechin din lista **MultimeRechin** și se utilizează obiectul **Graphics2D** pentru a desena rechinul pe coordonatele specificate.

Metoda checkCollisionRechin verifică coliziunile dintre jucător (scafandru) și rechinul final. Se calculează centrul rechinului și centrul scafandrului, iar apoi se calculează distanța dintre aceste puncte. Dacă distanța este mai mică sau egală cu suma valorilor razei, înseamnă că a avut loc o coliziune. În acest caz, se redă un sunet specific, rechinul este eliminat din lista MultimeRechin, jucătorul pierde toate viețile și este afișat un mesaj de înfrângere. Jocul se încheie prin apelul metodei System.exit(0).

Metoda move() gestionează mișcarea rechinului pe ecran. Rechinul se mișcă la fiecare iterare. Se actualizează coordonata x a fiecărui rechin prin decrementarea valorii acesteia. În plus, se verifică dacă vreun rechin a ieșit în afara ecranului la stânga. În acest caz, jocul se încheie prin apelul metodei System.exit(0).



Clasa **MAIN**

Atributul instance reprezintă instanța singleton a clasei Main, utilizată pentru a permite accesul la aceeași instanță în întregul joc.

Atributele de tip BufferedImage reprezintă imaginile utilizate în joc. Acestea sunt încărcate din fișierele corespunzătoare utilizând clasa ImageIO.

Metoda constructor Main() este privată și aruncă o excepție de tip Invalid_IMAGE_Exception în cazul în care nu se pot încărca imaginile. În cadrul

constructorului, se creează o instanță a clasei Game și se încarcă imaginile din sprite sheet-uri și fișierele corespunzătoare. Prin utilizarea metodei getSubimage a clasei BufferedImage, se extrag porțiuni din imaginile încărcate pentru a obține sprite-urile specifice pentru personaje, obiecte și fundaluri.

Metoda getInstance() reprezintă o metodă singleton care returnează instanța clasei Main. Dacă instanța nu a fost încă creată, se creează prin apelul constructorului privat Main().

Main		
m 🔒 Main()		
f 📁 <i>inima</i>	BufferedImage	
f 📁 <i>sirena</i>	BufferedImage	
f 📁 <i>sfera</i>	BufferedImage	
f 📁 <i>scoica</i>	BufferedImage	
f 📁 <i>peste</i>	BufferedImage	
f 📁 <i>level4</i>	BufferedImage	
f 📁 <i>inamic</i>	BufferedImage	
f 📁 <i>level3</i>	BufferedImage	
f 📁 <i>level2</i>	BufferedImage	
f 📁 <i>background2</i>	BufferedImage	
f 📁 <i>scafandru</i>	BufferedImage	
f 🔒 <i>instance</i>	Main	
f 📁 <i>background</i>	BufferedImage	
f 📁 <i>bomba</i>	BufferedImage	
f 📁 <i>background_menu</i>	BufferedImage	
m 📁 main(String[])	void	
m 🔒 getInstance()	Main	

Tratarea exceptiilor:

CLASA DE EXCEPTII

Pachetul Exceptions : conține o excepție proprie pentru invalidarea imaginilor

Clasa Invalid_IMAGE_Exception este o clasă de excepții personalizată care extinde clasa Exception. Această clasă este utilizată pentru a genera o excepție atunci când imaginea este considerată invalidă.

Constructorul clasei Invalid_IMAGE_Exception este definit cu modul de acces public. Acesta folosește constructorul superclasei Exception pentru a seta mesajul excepției la "Imagine invalidă!".

Constructorul clasei Main este definit cu modul de acces private și aruncă Invalid_IMAGE_Exception. Acesta este conceput pentru a crea o instanță a clasei Game și pentru a încărca imaginile necesare pentru joc.

În constructorul clasei Main, imaginile sunt încărcate din fișiere folosind clasa ImageIO și metoda read. Calea fișierelor este specificată prin intermediul metodei System.getProperty("user.dir"), care obține directorul curent al aplicației. Dacă apar excepții în timpul încărcării imaginilor, acestea sunt prinse și este aruncată o excepție Invalid_IMAGE_Exception.

```
1 package Exceptions;
2
3     //clasa pentru generarea exceptiei proprii
4
5 usages
6
7 public class Invalid_IMAGE_Exception extends Exception
8 {
9     1 usage
10    public Invalid_IMAGE_Exception()
11    {
12        super("Imagine invalida!");
13    }
14 }
```

```
level3 = ImageIO.read(new File( pathname: System.getProperty("user.dir") + "\\Resources\\10x64\\background_menu.png"));
background_menu = ImageIO.read(new File( pathname: System.getProperty("user.dir") + "\\resources\\Backgrounds\\background_menu.png"));

// redau caracterele cu coordonatele fiecarui element din imagine
scafandru = spriteSheet.getSubimage( x: 377, y: 1, w: 170, h: 170);
sirena = spriteSheet.getSubimage( x: 103, y: 1, w: 170, h: 170);
sfera = spriteSheet.getSubimage( x: 549, y: 1, w: 100, h: 100);
scoica = spriteSheet.getSubimage( x: 1, y: 1, w: 100, h: 100);
peste = spriteSheet.getSubimage( x: 275, y: 1, w: 100, h: 100);
bomba=bomb.getSubimage( x: 0, y: 0, w: 50, h: 50);
inima=Heart.getSubimage( x: 1, y: 1, w: 40, h: 40);
inamic= shark.getSubimage( x: 1, y: 1, w: 241, h: 110);

} catch (Exception e) {
    e.printStackTrace();
    throw new Invalid_IMAGE_Exception();
}

}
```

```
public static void main(String[] args) throws Invalid_IMAGE_Exception
{
    Main.getInstance();
}
```

EXCEPTII DE INTRARE/IESIRE :

In restul codului am utilizat structuri de control try-catch ca sa prind si sa tratez exceptiile si ofer mesaje de eroare sau iau masuri corespunzatoare ca sa rezolv problemele respective.

Un exemplu la baza de date :

```
public static void saveIntoDatabaseFinalScore() {
    try {
        String insertQuery = "INSERT INTO scoreTable (ScoruriFinale) VALUES (?)";
        PreparedStatement insertStatement = connection.prepareStatement(insertQuery);

        //Parametrul 1 specifică pozitia parametrului in instructiunea SQL,
        // iar Score.score reprezinta valoarea scorului final pe care dorim sa o inseram
        insertStatement.setInt( parameterIndex: 1, Score.score);

        //Se afiseaza in consola mesajul "Saved"
        System.out.println("Saved!");
        insertStatement.executeUpdate();

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

Eroarea care poate aparea este o exceptie de tipul SQLEXCEPTION
Cauzele erorii pot fi de exemplu : conexiunea la baza de date, structura bazei de date, restrictii/valori invalide etc.

Pentru gestionarea acestei exceptii , codul afiseaza o urmarire a stivei (stack trace), utilizand e.printStackTrace(). Aceasta va afisa informatii detaliate despre exceptie in consola, inclusiv mesajul de eroare si linia de cod unde a fost aruncata. Masuri pentru a remedia : corectarea conexiunii la baza de date, verificarea si ajustarea structurii tabelei, asigurarea conformitatii datelor inserate etc..

Sound

Am adaugat:

- Sunet de fundal pe tot parcursul jocului
- Efecte sonore : - colectare perla
 - coliziunea cu inamic (sirena)
 - castigarea jocului
 - atac inamic final (rechin)

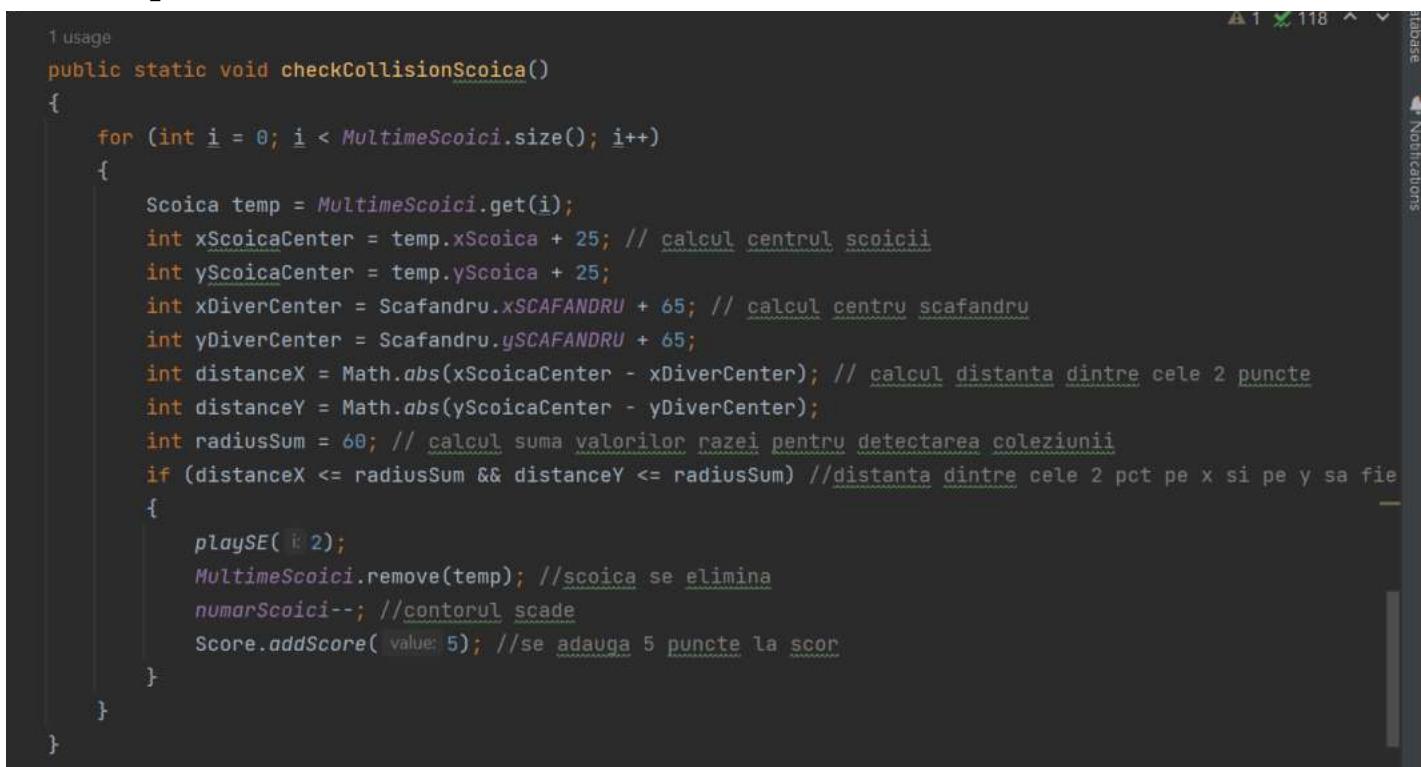
ALGORITMI

Descriere algoritmi coliziuni:

- **Coliziunea personajului cu obiectele pe care le colecteaza:**

Pentru fiecare element pe care eroul îl intersectează, am implementat o functie **checkCollision** de forma :

Un exemplu ar fi:



The screenshot shows a Java code editor with a dark theme. The code is a method named `checkCollisionScoica()` which iterates through a list of objects called `MultimeScoici`. For each object, it calculates the distance between the center of the object and the center of a diver named `Scafandru`. If the distance is less than or equal to a sum of radii, it plays a sound effect, removes the object from the list, decrements a counter, and adds 5 points to the score.

```
1 usage
public static void checkCollisionScoica()
{
    for (int i = 0; i < MultimeScoici.size(); i++)
    {
        Scoica temp = MultimeScoici.get(i);
        int xScoicaCenter = temp.xScoica + 25; // calcul centrul scoicii
        int yScoicaCenter = temp.yScoica + 25;
        int xDiverCenter = Scafandru.xSCAFANDRU + 65; // calcul centru scafandru
        int yDiverCenter = Scafandru.ySCAFANDRU + 65;
        int distanceX = Math.abs(xScoicaCenter - xDiverCenter); // calcul distanta dintre cele 2 puncte
        int distanceY = Math.abs(yScoicaCenter - yDiverCenter);
        int radiusSum = 60; // calcul suma valorilor razei pentru detectarea coliziunii
        if (distanceX <= radiusSum && distanceY <= radiusSum) //distanta dintre cele 2 pct pe x si pe y sa fie
        {
            playSE( i * 2);
            MultimeScoici.remove(temp); //scoica se elimina
            numarScoici--; //contorul scade
            Score.addScore( value: 5); //se adauga 5 puncte la scor
        }
    }
}
```

Am ales această metodă de coliziune, deoarece e mai eficientă pentru că utilizează un model simplu de coliziune circulară. În loc să verifice fiecare pixel în parte sau să folosească metode complexe de detecție a coliziunii, se calculează distanța între punctele centrale ale celor două entități și se compară cu suma razelor pentru a determina coliziunea. Acest lucru este mai rapid și mai eficient decât alte metode mai complexe, cum ar fi coliziunea bazată pe pixeli, în special în cazul jocurilor cu mulți actori sau obiecte care trebuie verificate într-un interval de timp scurt.

PASI:

-Un for parcurge fiecare element dintr-o mulțime de elemente

-Pentru fiecare element, calculez coordonatele centrului acestuia și ale scafandrului protagonist

-Calculez distanța pe axa x și pe axa y între cele două puncte centrale folosind funcția Math.abs, care returnează valoarea absolută a diferenței.

-Specific o valoare de suma a razelor (radiusSum). Această valoare reprezintă suma valorilor razei pentru a detecta coliziunea și determină cât de apropiate trebuie să fie cele două puncte pentru a fi considerate colizionate.

-Verific dacă distanța pe axa x este mai mică sau egală cu radiusSum și dacă distanța pe axa y este mai mică sau egală cu radiusSum. Această condiție verifică dacă cele două puncte se află la o distanță mai mică sau egală cu suma razelor, ceea ce indică o coliziune.

-În cazul în care se detectează o coliziune, se efectuează diferite acțiuni: efect sonor, elementul se elimină din multimea de elemente, contorul scade, se adaugă/scad puncte /viata

- **Coliziunea personajului cu inamici (sirena + rechin):**

```
1 usage
public static void checkCollisionSirena() {
    for (int i = 0; i < MultimeSirene.size(); i++) {
        Enemy temp = MultimeSirene.get(i);
        int xSirenaCenter = temp.xEnemy + 85; // calcul punct central sirena
        int ySirenaCenter = temp.yEnemy + 85;
        int xDiverCenter = Scafandru.xSCAFANDRU + 65; // calcul punct central scafandru
        int yDiverCenter = Scafandru.ySCAFANDRU + 65;
        int distanceX = Math.abs(xSirenaCenter - xDiverCenter); // calcul distanta dintre cele 2 puncte
        int distanceY = Math.abs(ySirenaCenter - yDiverCenter);
        int radiusSum = 100; //calcul suma valorilor razei pentru detectarea coliziunii
        if (distanceX <= radiusSum & distanceY <= radiusSum)
        {
            playSE( 3 );
            MultimeSirene.remove(temp); // sirena dispare
            NumarSirene--; // scade contorul
            Life.subtractLife( 1 ); // este scazuta o viata
        }
    }
}
```

```
1 usage
public static void checkCollisionRechin() //coliziune rechin cu personaj
{
    for (int i = 0; i < MultimeRechin.size(); i++) {
        Rechin temp = MultimeRechin.get(i);
        int xRechinCenter = temp.xRechin + 55; // calcul punct central rechin
        int yRechinCenter = temp.yRechin + 55;
        int xDiverCenter = Scafandru.xSCAFANDRU + 65; // calcul punct central scafandru
        int yDiverCenter = Scafandru.ySCAFANDRU + 65;
        int distanceX = Math.abs(xRechinCenter - xDiverCenter); // calcul distanta dintre cele 2 puncte
        int distanceY = Math.abs(yRechinCenter - yDiverCenter);
        int radiusSum = 100; //calcul suma valorilor razei pentru detectarea coliziunii
        if (distanceX <= radiusSum & distanceY <= radiusSum) {
            playSE( 5 );
            MultimeRechin.remove(temp);
            Life.Life = 0;
            JOptionPane.showMessageDialog( parentComponent: null, message: "Ai pierdut jocul! Ia-o de la capat!" );
            System.exit( status: 0 );
        }
    }
}
```

PASI:

-Un for parcurge fiecare element din multimea inamicilor

-Pentru fiecare element, calculez coordonatele centrului acestuia și ale scafandrului protagonist

-Calculez distanța pe axa x și pe axa y între cele două puncte centrale folosind funcția Math.abs, care returnează valoarea absolută a diferenței.

-Specific o valoare de suma a razelor (radiusSum). Această valoare reprezintă suma valorilor razei pentru a detecta coliziunea și determină cât de apropiate trebuie să fie cele două puncte pentru a fi considerate colizionate.

-Verific dacă distanța pe axa x este mai mică sau egală cu radiusSum și dacă distanța pe axa y este mai mică sau egală cu radiusSum. Această condiție verifică dacă cele două puncte se află la o distanță mai mică sau egală cu suma razelor, ceea ce indică o coliziune.

-În cazul în care se detectează o coliziune, se efectuează diferite acțiuni: efect sonor, elementul se elibera din multimea de elemente, contorul scade, se adaugă/scad puncte /viata

- **Coliziunea sferei mistice cu inamici (sirena + rechin):**

Pentru inamicii pe care sfera mistica îi intersectează, am implementat funcția [checkCollisionInamici](#):

```
1 usage
public static void checkCollisionInamici() {
    List<Bullet> bulletsToRemove = new ArrayList<>(); // Lista pentru a stoca sferele ce trebuie eliminate
    List<Enemy> enemiesToRemove = new ArrayList<>(); // Lista pentru a stoca inamicii ce trebuie eliminati
    List<Rechin> rechiniToRemove = new ArrayList<>(); // Lista pentru a stoca rechinii ce trebuie eliminati

    for (Bullet tempBullet : MultimeSfere) {
        Rectangle bulletBounds = new Rectangle(tempBullet.xBullet, tempBullet.yBullet, Main.sfera.getWidth(), Main.sfera.getHeight());

        for (Enemy tempEnemy : Enemy.MultimeSirene) {
            Rectangle enemyBounds = new Rectangle(tempEnemy.xEnemy, tempEnemy.yEnemy, Main.sirena.getWidth(), Main.sirena.getHeight());

            if (bulletBounds.intersects(enemyBounds)) {
                bulletsToRemove.add(tempBullet); // Adaugă sfera la lista de eliminat
                enemiesToRemove.add(tempEnemy); // Adaugă inamicul la lista de eliminat
                Score.addScore( value: 10); // Adaugă 10 puncte la scor
                break; // Iesire din bucla internă pentru a evita modificarea concurentă
            }
        }

        for (Rechin tempRechin : Rechin.MultimeRechin) {
            Rectangle rechinBounds = new Rectangle(tempRechin.xRechin, tempRechin.yRechin, Main.inamic.getWidth(), Main.inamic.getHeight());
        }
    }
}
```

```
for (Rechin tempRechin : Rechin.MultimeRechin) {
    Rectangle rechinBounds = new Rectangle(tempRechin.xRechin, tempRechin.yRechin, Main.inamic.getWidth(), Main.inamic.getHeight());

    if (bulletBounds.intersects(rechinBounds)) {

        Rechin.NumarRechin--;
        bulletsToRemove.add(tempBullet); // Adaugă sfera la lista de eliminat
        rechiniToRemove.add(tempRechin); // Adaugă rechinul la lista de eliminat
        playSE( id: 4);
        Score.addScore( value: 100); // Adaugă 100 puncte la scor
        break; // Iesire din bucla internă pentru a evita modificarea concurentă
    }
}

MultimeSfere.removeAll(bulletsToRemove); // Eliminare toate gloantele din lista de eliminat
Enemy.MultimeSirene.removeAll(enemiesToRemove); // Eliminare toți inamicii din lista de eliminat
Rechin.MultimeRechin.removeAll(rechiniToRemove); // Eliminare toți rechinii din lista de eliminat
maximumSize -= bulletsToRemove.size(); // Scazut numărul de gloante
Enemy.NumarSirene -= enemiesToRemove.size(); // Scazut numărul de sirene
}
```

Acest tip de coliziune bazat pe obiecte de tip Rectangle este eficient deoarece

permite o verificare rapidă și eficientă a coliziunii între două entități. Utilizarea dreptunghiurilor pentru a defini limitele permite o detecție simplă și rapidă a coliziunii, fără a fi nevoie să se compare fiecare pixel sau să se utilizeze metode complexe de detecție a coliziunii. Aceasta abordare permite și o implementare ușoară a eliminării entităților colizionate și actualizarea scorului.

PASI:

Creez trei liste separate pentru a stoca sferele ce trebuie eliminate (bulletsToRemove), inamicii ce trebuie eliminați (enemiesToRemove), și rechinii ce trebuie eliminați (rechiniToRemove)

Parcurg lista MultimeSfere care conține toate sferele de putere magica

Pentru fiecare sferă, creez un obiect de tip Rectangle care reprezintă limitele acesteia. Obiectul Rectangle este definit cu coordonatele x și y ale sferei și dimensiunile luate din imaginea sprite-ului (Main.sfera.getWidth() și Main.sfera.getHeight())

Se parcurg listele de inamici (Enemy.MultimeSirene) și rechini (Rechin.MultimeRechin)

Pentru fiecare inamic sau rechin, creez un obiect de tip Rectangle care reprezintă limitele acestuia

Verific dacă limitele sferei intersectează limitele inamicului sau rechinului folosind metoda intersects a obiectului Rectangle

În cazul în care se detectează o coliziune:

- Sfera și inamicul/ rechinul sunt adăugate în liste separate (bulletsToRemove/enemiesToRemove/rechiniToRemove) pentru a fi eliminați ulterior. Cresc scorul în funcție de puncte (cu 10/100)
- După ce am verificat toate coliziunile, se elimină toate sferele, inamicii și rechinii din listele respective folosind metoda removeAll

Se actualizează variabilele de numărare a sferelelor și a inamicilor (maximumSize și Enemy.NumarSirene) prin scăderea dimensiunii listelor de

eliminat (bulletsToRemove ,enemiesToRemove)

- **Coliziunea personajului cu harta (ecranul)**

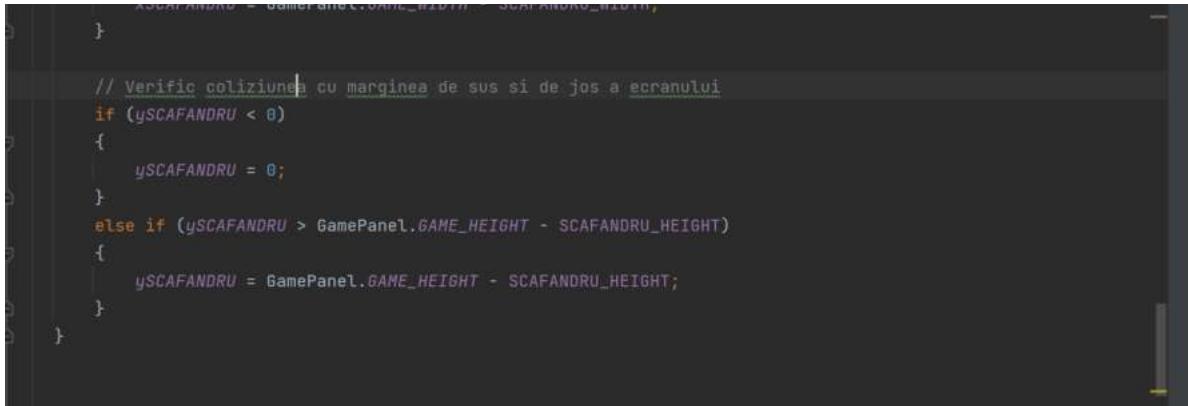
The screenshot shows a code editor with two panels. The top panel displays a list of variable usages for `xVelocity`, `yVelocity`, `speed`, `ySCAFANDRU`, `xSCAFANDRU`, and `xSCAFANDRU`. The bottom panel shows the corresponding Java code for the `move()` method, which includes collision detection logic for the player's position.

```
2 usages
int xVelocity;      //viteza pe OX
2 usages
int yVelocity; //viteza pe OY
4 usages
int speed = 5; //viteza
17 usages
public static int ySCAFANDRU; //pozitia pe OY a scafandrului
17 usages
public static int xSCAFANDRU; //pozitia pe OX a scafandrului
2 usages
int y = ySCAFANDRU;
no usages
int x = xSCAFANDRU;
/* Scafandru(int x, int y) //setez pozitia scafandrului
{
    xSCAFANDRU = x;
    ySCAFANDRU = y;
}

1 usage
public void move()
{
    xSCAFANDRU += xVelocity;
    ySCAFANDRU += yVelocity;

    // Verific coliziunea cu marginea din stanga si dreapta a ecranului
    if (xSCAFANDRU < 0)
    {
        xSCAFANDRU = 0;
    }
    else if (xSCAFANDRU > GamePanel.GAME_WIDTH - SCAFANDRU_WIDTH)
    {
        xSCAFANDRU = GamePanel.GAME_WIDTH - SCAFANDRU_WIDTH;
    }

    // Verific coliziunea cu marginea de sus si de jos a ecranului
    if (ySCAFANDRU < 0)
    {
        ySCAFANDRU = 0;
    }
    else if (ySCAFANDRU > GamePanel.GAME_HEIGHT - SCAFANDRU_HEIGHT)
    {
        ySCAFANDRU = GamePanel.GAME_HEIGHT - SCAFANDRU_HEIGHT;
    }
}
```



```
        }

        // Verific coliziunea cu marginea de sus si de jos a ecranului
        if (ySCAFANDRU < 0)
        {
            ySCAFANDRU = 0;
        }
        else if (ySCAFANDRU > GamePanel.GAME_HEIGHT - SCAFANDRU_HEIGHT)
        {
            ySCAFANDRU = GamePanel.GAME_HEIGHT - SCAFANDRU_HEIGHT;
        }
    }
```

Actualizez pozitia personajului principal pe baza vitezei in directiile x si y.
XScafandru si YScafandru sunt coordonatele curente ale personajului
XVelocity si YVelocity sunt vitezele curente ale personajului in directiile x si y

Înainte de a actualiza poziția personajului, se verifică coliziunea cu marginile ecranului. În cazul în care personajul depășește marginea din stânga sau dreapta a ecranului, adică $xSCAFANDRU < 0$ sau $xSCAFANDRU > GamePanel.GAME_WIDTH - SCAFANDRU_WIDTH$, se iau măsuri pentru a preveni această coliziune.

Dacă personajul depășește marginea din stânga, se setează $xSCAFANDRU$ (poziția personajului pe axa x) la valoarea minimă posibilă, adică 0. Astfel, personajul nu va putea ieși în afara ecranului pe partea stângă.

În cazul în care personajul depășește marginea din dreapta, se ajustează $xSCAFANDRU$ la valoarea corespunzătoare pentru a-l ține în interiorul ecranului. Valoarea este calculată prin scăderea lățimii personajului ($SCAFANDRU_WIDTH$) din lățimea totală a jocului ($GamePanel.GAME_WIDTH$). Astfel, personajul nu va putea ieși în afara ecranului pe partea dreaptă.

Similar, se verifică și coliziunea personajului cu marginea de sus și de jos a ecranului. Dacă personajul depășește marginea de sus, se setează $yScafandru$ (poziția personajului pe axa y) la 0, astfel împiedicându-l să iasă în afara ecranului pe partea de sus. În cazul în care personajul depășește marginea de jos, $yScafandru$ este ajustat pentru a-l menține în interiorul ecranului. Valoarea este calculată similar cu coliziunea pe axa x, prin scăderea înălțimii

personajului (SCAFANDRU_HEIGHT) din înălțimea totală a jocului (GamePanel.GAME_HEIGHT).

Astfel, prin aceste verificări de coliziune cu marginile ecranului, funcția move() asigură că personajul rămâne în interiorul limitelor vizibile și nu poate ieși în afara ecranului. Aceasta contribuie la o experiență de joc coerentă și previne situațiile în care personajul ar putea dispărea sau ar deveni incontrolabil.

Descriere deplasarea inamicilor:

Algoritmul de deplasare este implementat într-o buclă care se execută de mai multe ori pe secundă, astfel încât inamicii se vor mișca într-un mod continuu și fluid pe ecran în timpul jocului.

SIRENE:

```
public static void move() //misiunea sirenelor
{
    Iterator<Enemy> iterator = MultimeSirene.iterator();
    while (iterator.hasNext()) {
        Enemy enemy = iterator.next();
        enemy.setXEnemy();
        if (enemy.xEnemy < -44) {
            iterator.remove();
            Enemy.NumarSirene--;
        }
    }
}
```

-Se inițializează un iterator pentru a parcurge lista de sirene (MultimeSirene)

-Se parcurge lista folosind un buclă while și iteratorul

-Pentru fiecare sirenă, se apelează metoda setXEnemy() pentru a actualiza poziția pe axa x a sirenului

-Dacă sirena a depășit o anumită valoare pe axa x (-44), înseamnă că s-a deplasat complet în afara ecranului

-Sirena este eliminată din lista utilizând metoda iterator.remove()

-Numărul total de sirene (Enemy.NumarSirene) este decrementat

RECHIN:

```
1 usage
public static void move() {
    moveCounter++;
    if (moveCounter >= MOVE_INTERVAL) {
        moveCounter = 0;
        Iterator<Rechin> iterator = MultimeRechin.iterator();
        while (iterator.hasNext()) {
            Rechin rechin = iterator.next();
            rechin.xRechin--;

            if (rechin.xRechin + 241 < 0) {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Ai pierdut jocul! Ia-o de la cap!");
                System.exit( status: 0);
            }
        }
    }
}
```

-Se utilizează o variabilă moveCounter pentru a urmări numărul de iterații

-Se stabilește un interval de mișcare (MOVE_INTERVAL) în iterații

-În fiecare iterație, se incrementează moveCounter

-Dacă moveCounter depășește sau este egal cu MOVE_INTERVAL, se realizează mișcarea rechinului

- Se inițializează un iterator pentru a parcurge lista de rechini (MultimeRechin)
- Se parcurge lista folosind un buclă while și iteratorul
- Pentru fiecare rechin, se decrementează coordonata x (rechin.xRechin) pentru a-l deplasa spre stânga
- Dacă coordonata x a rechinului plus lungimea acestuia (241 în cazul dat) este mai mică decât 0, înseamnă că rechinul s-a deplasat complet în afara ecranului și jucătorul a pierdut jocul
- Se afișează un mesaj ca s-a pierdut jocul
- Se încheie programul utilizând System.exit(0)

Descriere algoritm gestiune obiecte:

Obiectele colectate de scafandru sunt : scoicile care îi aduc 5p la scor și inimile care îi aduc 5 vieți, dar scad 30 de p din scor. De restul obiectelor jucătorul trebuie să se ferească, deoarece sirenele îi scad 10p , iar rechinul îl omoară.

(Algoritmii sunt prezentati mai sus)

SABLOANE DE PROIECTARE:

SINGLETON - in main

```
import Exceptions.Invalid_IMAGE_Exception;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;

public class Main {
    private static Main instance; //instanta singleton
```

```
//Singleton
public static Main getInstance() throws Invalid_IMAGE_Exception
{
    if(instance==null)
    {
        instance=new Main();
    }
    return instance;
}

public static void main(String[] args) throws Invalid_IMAGE_Exception
{
    Main.getInstance();
}
```

Singleton ul se asigura ca , clasa main are o singura instantă și oferă un punct global de acces la aceasta instantă.

Am declarat variabila statică privată instance de tip main - variabila va retine instantă singleton a clasei. E privată ca să se evite accesul din exterior și pastrăm controlul asupra creării instantei singleton.

Constructorul clasei main e privat ca să nu poată fi accesat din exterior. Asta previne crearea de noi instante ale clasei în afara de clasa în sine.

Am definit metoda publica statica getInstance care returneaza instanta clasei Main. Verific daca e null, daca da inseamna ca nu a fost creata inca o instantă si creez una noua prin apelul constructorului privat. Daca nu e null inseamna ca exista deja o instantă si e returnată.

Metoda main e punctul de intrare in aplicatie deci aici apelez metoda ca sa obtin instantă clasei main.

Un bonus este economia pentru ca resursele (imagini etc..) se incarca doar o singura data in memoria aplicatiei, astfel evit incarcarea duplicata a resurselor.

STATE- in GamePanel

```
GamePanel.java × Menu2.java × MouseInput.java × Scafandru.java × Invalid_IMAGE_Exception.java × Peste.java ×
import javax.swing.*;

/**
 * CLASA CARE ASIGURA TOATA ACTIUNEA JOCULUI
 */
public class GamePanel extends JPanel implements Runnable {

    20 usages
    public enum STATE {
        //STATE PATTERN
        MENU, GAME, SETTINGS, EXIT, BACK, SUNET
    }
    //Starile posibile ale jocului care pot fi folosite pentru a controla fluxul jocului
}
```

```
20 usages
public static STATE State = STATE.MENU; //variabila in care tin evidenta starii curente a jocului
6 usages
```

```
    public void draw(Graphics g) {  
  
        if (State == STATE.GAME) {  
            GamePanel.createDatabase();  
            GamePanel.createDatabaseFinalScore();  
            Image img2 = Main.background2;  
            switch (level) { /*desenez fiecare background diferit in functie de nivel*/  
  
                case 1: {  
                    Image img1 = Main.background;  
                    background_draw(img1, img2, g);  
                    break;  
                }  
                case 2: {  
                    Image img21 = Main.level2;  
                    background_draw(img21, img2, g);  
                    break;  
                }  
            }  
        }  
    }  
}
```

```
1 usage  
public void move() { /*asigur miscarea tuturor elementelor*/  
    if (State == STATE.GAME) {  
        scafandru.move();  
        Bullet.move();  
        Enemy.move();  
        Scoica.move();  
        Peste.move();  
        if (level >= 2) {  
            Bomba.move();  
        }  
        if (level >= 3) {  
            Inima.move();  
            Rechin.move();  
        }  
    }  
}
```

Am folosit sablonul STATE ca sa asigur o gestionare mai eficienta a starilor si actiunilor jocului. Am implementat functionalitati specifice pentru fiecare stare (in clasele GamePanel si MouseInput).

Se defineste un enum STATE care reprezinta starile posibile ale jocului. Am folosit enum ca sa fie mai eficient de definit si gestionat starile, ma asigur ca sunt utilizate doar valori valide.

Se utilizeaza o variabila State ca sa se tina evidenta starii curente a jocului. In functie de valoarea variabilei aleg sa desenez anumite elemente sau sa efectuez diverse actiuni in metodele draw si move.

In draw: se deneaza diverse imagini de fundal pentru fiecare nivel, obiecte etc.

Interactiunea cu utilizatorul e mai usoara prin intermediul starilor.

Utilizatorul interactioneaza cu jocul in moduri diferite in functie de starea curenta.

Se indeplineste proprietatea de modularitate. Sablonul ajuta sa separ logic starile jocului si actiunile asociate acestora. Jocul e mai modular si usor de intretinut, deoarece fiecare stare poate sa aiba propriile reguli si actiuni. Pot dezvolta si extinde ulterior jocul

In clasa **MOUSEINPUT**:

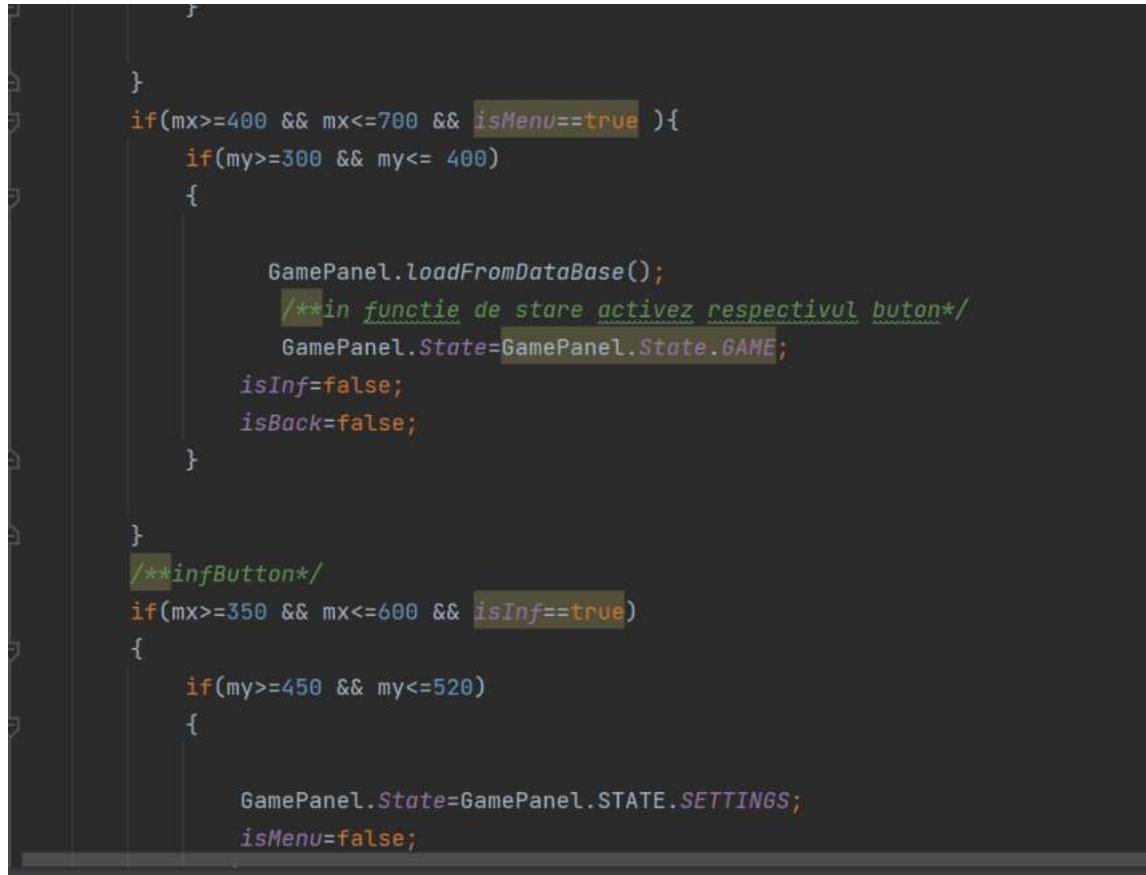
```
java.java × Menu2.java × MouseInput.java × Scafandru.java × Invalid_IMAGE_Exception.java × Peste.j
import java.awt.event.*;

3 usages
public class MouseInput implements MouseListener {

    9 usages
    static boolean isMenu = true;
    7 usages
    static boolean isInf=true;
    6 usages
    static boolean isBack=true;

    @Override
    public void mouseClicked(MouseEvent e) //cand se face click
    {
    }
    @Override
    public void mouseEntered(MouseEvent e) //intra intr o componentă
    {
    }

    @Override
```



```
        }
        if(mx>=400 && mx<=700 && isMenu==true ){
            if(my>=300 && my<= 400)
            {

                GamePanel.loadFromDataBase();
                /*in functie de stare activez respectivul buton*/
                GamePanel.State=GamePanel.State.GAME;
                isInf=false;
                isBack=false;
            }

        }
        /*infButton*/
        if(mx>=350 && mx<=600 && isInf==true)
        {
            if(my>=450 && my<=520)
            {

                GamePanel.State=GamePanel.STATE.SETTINGS;
                isMenu=false;
            }
        }
    }
}
```

Am implementat interfata MouseListener si am suprascris diverse metode. In metoda mouseReleased am folosit sablonul STATE ca sa controlez fluxul actiunilor bazate pe starea curenta a jocului ca sa gestionez mai usor actiunile si comportamentul butoanelor. Contribuie la o structura mai usor de intelese si intretinut . Codul e flexibil si extensibil.

Am definit cateva variabile de stare care indica daca anumite instructiuni sunt disponibile de starea curenta a jocului apoi am implementat logica pentru actiunile de Mouse in functie de coordonatele mouse ului si starea curenta a jocului.

Diagrame

Diagrama finala:

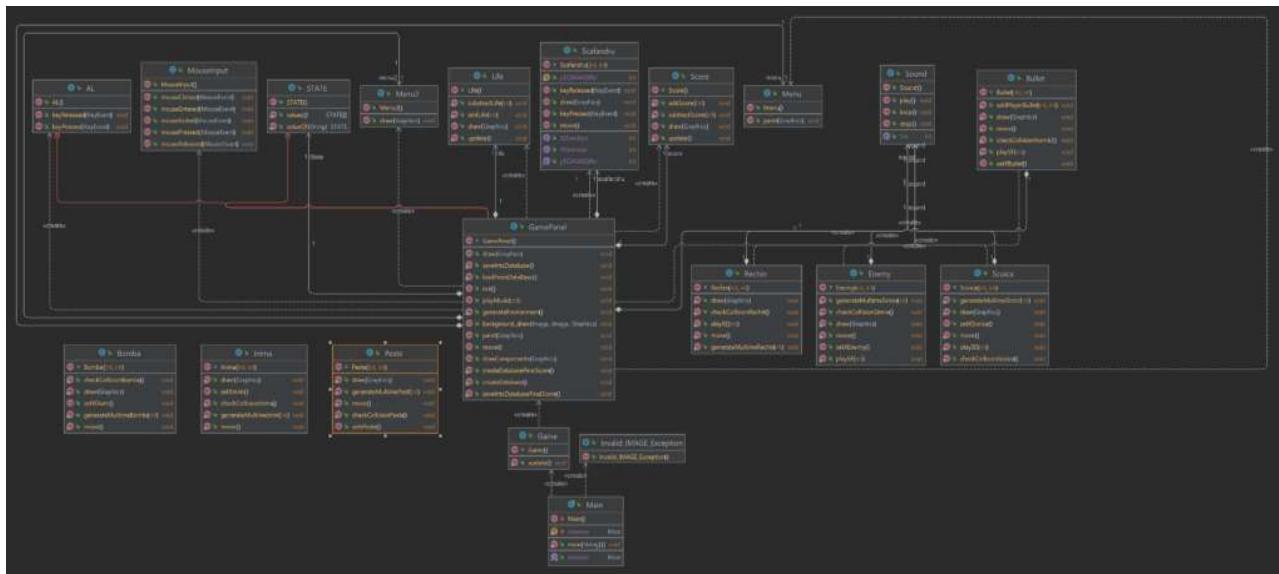


Diagrama Pachet Exceptii:



Diagrama Singleton in MAIN

Prin implementarea sablonului Singleton, se asigură că există o singură instanță a clasei Main în cadrul jocului.

Main		
m	>Main()	
f	scafandru	BufferedImage
f	inamic	BufferedImage
f	level4	BufferedImage
f	bomba	BufferedImage
f	peste	BufferedImage
f	sirena	BufferedImage
f	level2	BufferedImage
f	sfera	BufferedImage
f	background_menu	BufferedImage
f	background2	BufferedImage
f	scoica	BufferedImage
f	inima	BufferedImage
f	instance	Main
f	level3	BufferedImage
f	background	BufferedImage
m	main(String[])	void
m	getInstance()	Main

BAZA DE DATE

Baza de date pe care am implementat-o include funcții pentru crearea, salvarea și încărcarea datelor, precum și pentru crearea și salvarea scorurilor finale.

DB Browser for SQLite - D:\1207B_ButuAlexandra_etapa2\The treasure from the depths\db\tiobe.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 Pragmas Execute SQL

Table: playerTable

	id	Nivel	scafandrx	scafandruy	Score	Life
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	865	10	-100	14
2	2	1	615	245	-190	18
3	3	1	0	0	0	10
4	4	1	0	0	0	10
5	5	1	0	0	0	10
6	6	1	0	0	0	10
7	7	1	0	0	0	10
8	8	2	535	155	20	10
9	9	2	535	155	20	10
10	10	2	535	155	20	10
11	11	2	535	155	20	10
12	12	2	535	155	20	10
13	13	2	535	155	20	10
14	14	2	535	155	20	10
15	15	1	0	0	0	10
16	16	1	0	0	0	10
17	17	1	0	0	0	10

1 - 18 of 32 Go to: 2

Edit Database Cell

Mode: Text Import Export Set as NULL

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

DB Browser for SQLite - D:\1207B_ButuAlexandra_etapa2\The treasure from the depths\db\tiobe.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 Pragmas Execute SQL

Table: scoreTable

	id	ScoruriFinale
	Filter	Filter
1	1	95
2	2	80
3	3	95
4	4	40
5	5	55
6	6	145
7	7	115
8	8	35
9	9	110
10	10	85
11	11	75
12	12	110
13	13	330
14	14	130
15	15	180
16	16	190
17	17	115

1 - 18 of 31 Go to: 2

Edit Database Cell

Mode: Text Import Export Set as NULL

Type of data currently in cell: Text / Numeric 1 char(s) Apply

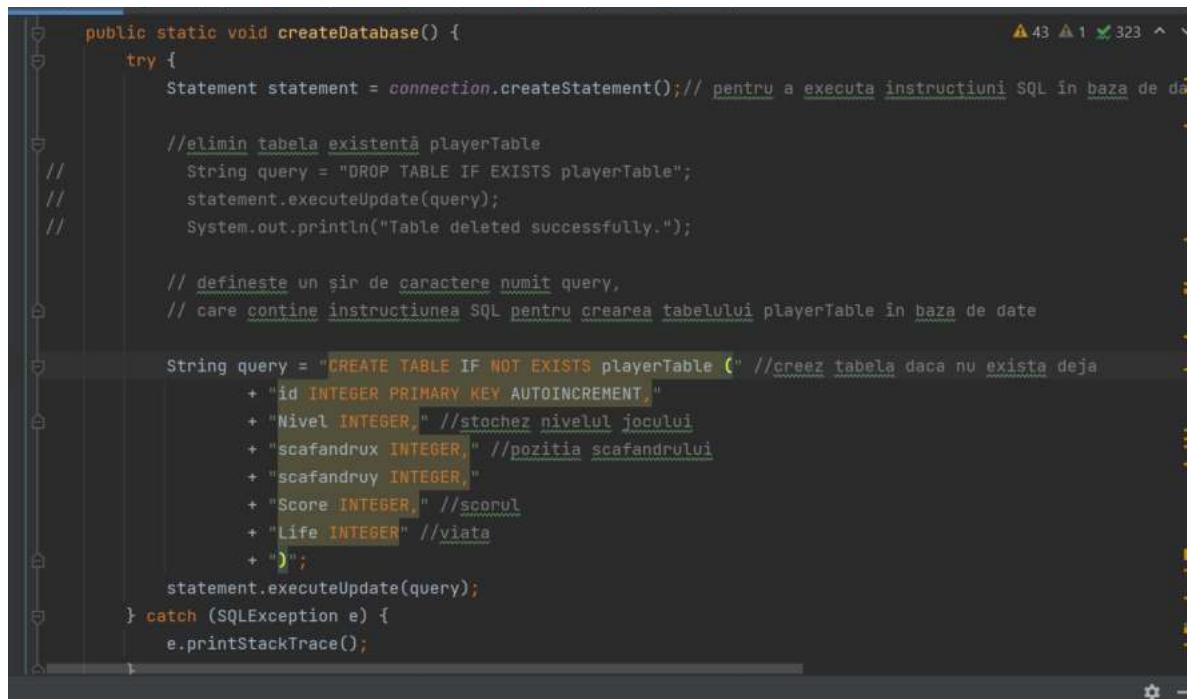
Remote

Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

Functia createDatabase asigură că tabela playerTable există în baza de date, iar structura sa este definită pentru a stoca informațiile relevante despre joc, precum nivelul, poziția scafandrului, scorul și viața.



```
public static void createDatabase() {
    try {
        Statement statement = connection.createStatement(); // pentru a executa instructiuni SQL in baza de date

        //elimin tabela existentă playerTable
        String query = "DROP TABLE IF EXISTS playerTable";
        statement.executeUpdate(query);
        System.out.println("Table deleted successfully.");

        // defineste un sir de caractere numit query,
        // care contine instructiunea SQL pentru crearea tabelului playerTable in baza de date

        String query = "CREATE TABLE IF NOT EXISTS playerTable ( //creez tabela daca nu exista deja
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            Nivel INTEGER, //stocchez nivelul jocului
            scafandruX INTEGER, //pozitia scafandrului
            scafandruY INTEGER,
            Score INTEGER, //scorul
            Life INTEGER //viata
        )";
        statement.executeUpdate(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Functia saveIntoDatabase pregătește și execută o instrucțiune SQL pentru a insera datele curente ale jocului (nivelul, poziția scafandrului, scorul și viața) în tabela playerTable a bazei de date.

```
public static void saveIntoDatabase() {
    try {
        // permit inlocuirea valorilor ??? cu parametri reali
        String insertQuery = "INSERT INTO playerTable (Nivel, scafandruX, scafandruY, Score, Life) VALUES (?, ?, ?, ?, ?)";
        //pregatesc instructiunea SQL pentru inserare
        PreparedStatement insertStatement = connection.prepareStatement(insertQuery);

        //Se specifica indexul parametrului (1-5) si valoarea pe care sa o inlocuiasca in instructiunea SQL.
        // Valorile sunt preluate din level, Scafandru.xSCAFANDRU, Scafandru.ySCAFANDRU, Score.score , Life.Life
        insertStatement.setInt( parameterIndex: 1, level);
        insertStatement.setInt( parameterIndex: 2, Scafandru.xSCAFANDRU);
        insertStatement.setInt( parameterIndex: 3, Scafandru.ySCAFANDRU);
        insertStatement.setInt( parameterIndex: 4, Score.score);
        insertStatement.setInt( parameterIndex: 5, Life.Life);

        //se afiseaza mesajul "Saved" in consola
        System.out.println("Saved");
        //execut instructiunea SQL si efectuez inserarea datelor in tabela
        insertStatement.executeUpdate();

        // eliberez resursele si inchei conexiunea cu baza de date
        insertStatement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Functia loadFromDataBase() execută o interogare SQL pentru a selecta ultimul rând din tabela playerTable și încarcă valorile acestui rând în variabilele și obiectele corespunzătoare din joc: nivelul, poziția scafandrului, scorul și viața.

```
1 usage
public static void loadFromDataBase() {
    try (Statement statement = connection.createStatement()) {

        //selectez toate coloanele din tabela playerTable, ordonez rezultatele după coloana id
        // în ordine descrescătoare și limitez rezultatele la un singur rând
        ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM playerTable ORDER BY id DESC LIMIT 1");

        if (resultSet.next())//Dacă există un rând disponibil
            {//obtin valorile corespunzătoare din coloanele "Nivel", "scafandrux", "scafandruy", "Score", "Life"
                // Aceste valori sunt stocate în variabilele și obiectele corespunzătoare din joc
                level = resultSet.getInt( columnLabel: "Nivel");
                Scafandru.xSCAFANDRU = resultSet.getInt( columnLabel: "scafandrux");
                Scafandru.ySCAFANDRU = resultSet.getInt( columnLabel: "scafandruy");
                Score.score = resultSet.getInt( columnLabel: "Score");
                Life.Life = resultSet.getInt( columnLabel: "Life");
                System.out.println("Incarcat");//afisez mesajul "Incarcat" în consolă
            }
        resultSet.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

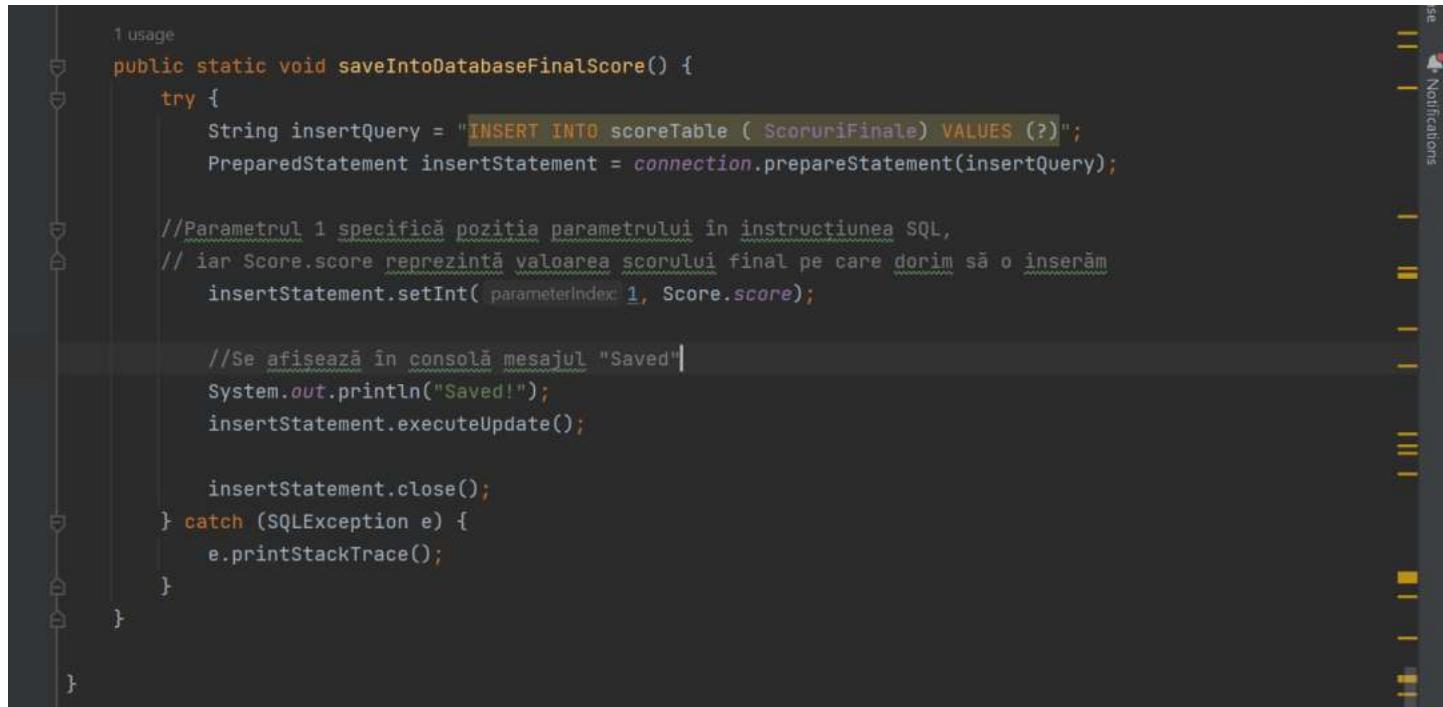
Functia createDatabaseFinalScore verifică dacă tabela "scoreTable" există în baza de date și, în caz contrar, o creează și introduce scorul final .

```
1 usage
public static void createDatabaseFinalScore() {
    try {
        Statement statement = connection.createStatement();

        // String query = "DROP TABLE IF EXISTS playerTable";
        // statement.executeUpdate(query);
        // System.out.println("Table deleted successfully.");
        //

        // tabela "scoreTable" are o coloană "id" de tip INTEGER cu cheie primară
        // și incrementare automată, și o coloană "ScoruriFinale" de tip INTEGER
        String query = "CREATE TABLE IF NOT EXISTS scoreTable (" +
                       + "id INTEGER PRIMARY KEY AUTOINCREMENT," +
                       + "ScoruriFinale" +
                       + ")";
        statement.executeUpdate(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Functia saveIntoDatabaseFinalScore pregătește și execută o instrucțiune SQL de inserare a scorului final în tabela "scoreTable" a bazei de date. Valorile sunt preluate din variabilele corespunzătoare, iar rezultatul este afișat în consolă.



```
1 usage
public static void saveIntoDatabaseFinalScore() {
    try {
        String insertQuery = "INSERT INTO scoreTable (ScoruriFinale) VALUES (?)";
        PreparedStatement insertStatement = connection.prepareStatement(insertQuery);

        //Parametrul 1 specifică pozitia parametrului in instructiunea SQL,
        //iar Score.score reprezinta valoarea scorului final pe care dorim sa o inseram
        insertStatement.setInt(parameterIndex 1, Score.score);

        //Se afiseaza in consola mesajul "Saved"
        System.out.println("Saved!");
        insertStatement.executeUpdate();

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

Bibliografie

[1] 2D Game Development From Zero To Hero - Daniele Penazzo

Collision Detection and Reaction - pag: 122

Design Patterns - pag: 239

Developing Game Mechanics - pag: 301

[2] Core Java Volume I—Fundamentals (11th Edition) by Cay S. Horstmann

[3] Developing Games in Java by David Brackeen

Chapter 4. Sound Effects and Music

Chapter 11. Collision Detection

[4] <https://www.artstation.com/artwork/JLPY0>

Poza principala joc

[5]

<https://craftpix.net/product/underwater-world-game-kit/?num=1&count=1&sq=mermaid&pos=0>

Inspiratie elemente subacvatice

[6] <https://craftpix.net/freebies/free-underwater-world-2d-game-objects/>

Inspiratie elemente subacvatice

[7]

<https://craftpix.net/freebies/free-animated-explosion-sprite-pack/?num=1&count=46&sq=explosion&pos=5>

Inspiratie sfera mistica

[8] <https://ro.pinterest.com/search/pins/?q=game%20buttons%20ui%20design&rs=typed>
Inspiratie butoane/meniu

[9]

<https://ro.pinterest.com/search/pins/?q=game%20scuba%20diver%20character&rs=typed>

// Inspiratie personaj principal

[10]

https://www.codeandweb.com/free-sprite-sheet-packer?fbclid=IwAR0XTc3IrhYK6KAAs7hgVZ18bjMJA0rJTr-Uu74Np0qjPV0D0aB9E4B_oZos

//spritesheet

[11] <https://mixkit.co/free-sound-effects/game/?page=2>

//efecte sonore