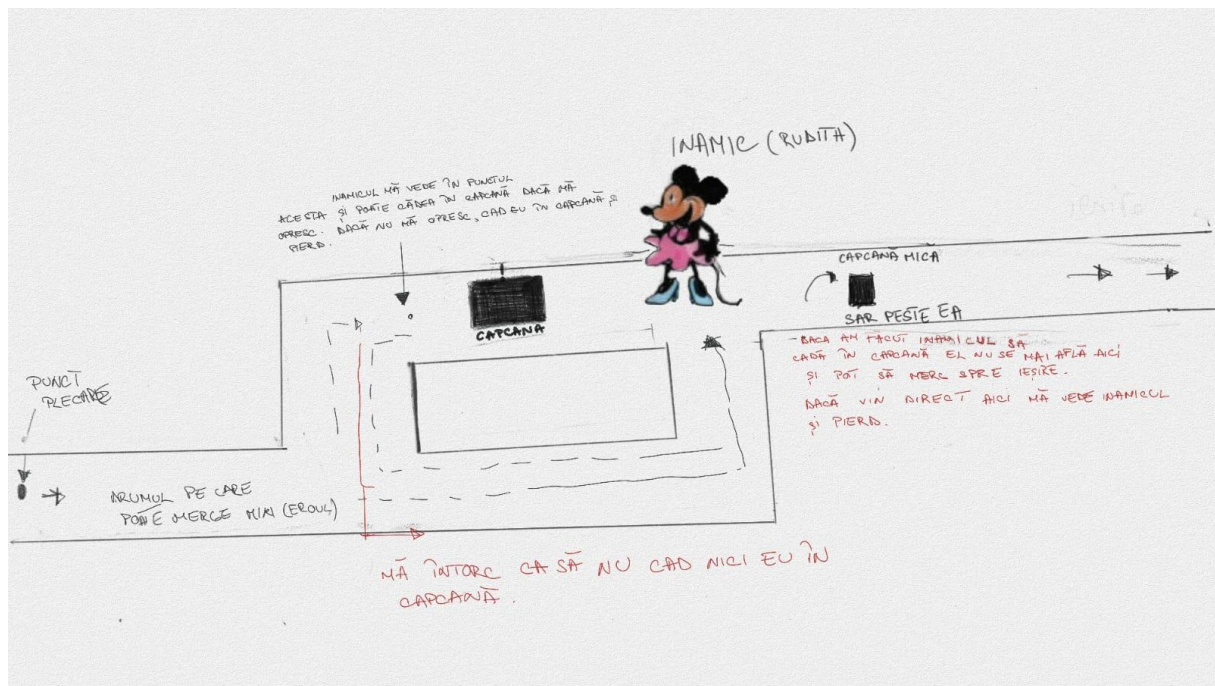


Șoarecu'
Buzan Alexandra – Maria
1211A

1. Proiectarea contextului și descriere detaliată

Jocul este situat într-o mină de nestemate unde locuiește șorecele Miki, un șoarece respectat în societate, alături de soția sa, Mini, și cei șapte șoricei ai săi. Miki trăiește o viață învăluită în mister, având afaceri ascunse cu nestemate furate din mină, dar cu imaginea unui soț și tată muncitor și cinstit, care muncește din greu să își susțină familia. După o zi grea de muncă, Miki reușește să fure prin metodele sale vechi mai multe nestemate, atât pentru că voia să se îmbogățească, dar și pentru a-i face un dar soției sale, Mini, cu ocazia aniversării a 2 ani de la nunta șoricească. În mod neobișnuit, Miki este supus unui control de rutină înainte de a părăsi mina, iar astfel fosta sa iubită, Rudith, descoperă nestematele furate asupra lui Miki și îl întemnițează. Problema este agravată de faptul că Miki i-a promis soției sale că nu va lucra alături de fosta lui iubită, care acum îl urăște pentru că a părăsit-o și pentru că a scos-o din afacerile lui. Acasă, cei șapte șoricei o ajută pe mama lor să pregătească o cină festivă pentru aniversarea la care vor fi prezente toate rudele celor doi soți. Din păcate, Miki nu poate ajunge la aniversare, iar cina decurge foarte prost din cauza absenței sale. Rudith apare neinvitată la aniversare și anunță personal deținerea lui Miki, mod prin care soția află și că acesta nu a rupt niciodată legătura cu marea ei rivală din trecut, dar și că toate darurile primite de la el au fost de fapt furate. Furioși, părinții lui Mini o obligă pe aceasta să îl părăsească pe Miki pentru un șoarece cinstit și bogat, care să nu le umilească fiica. Devastat și cu orgoliul rănit, Miki trebuie să evadeze pentru a-și recupera familia și onoarea, dar și pentru a se întâlni cu partenerii săi de jaf care îl așteaptă și se bazează pe el. Astfel, sunt puse în joc familia, reputația și afacerile șoarecelui nostru.





Personajul principal trebuie să evadeze din temnița minerilor după ce s-a aflat de traficul său cu nestemate și să ajungă în orașul șoarecilor . Miki trebuie să rezolve diferite labirinturi astfel încât să atragă la timp inamicul în ea. Dacă este prins sau dacă pică el în capcană , acesta pierde o viață din cele 3. La nivelurile inferioare nu poate face decât jump simplu , apoi la nivelul 2 poate face jump dublu cu care poate sări peste capcanele mari , iar la nivelul 3 și inamicul va putea sări peste capcanele mici.

Tipul jocului : survival.

2. Proiectarea sistemului și descriere detaliată

Controale:

W-Miki merge în sus

A-Miki merge în stanga

S-Miki merge în jos

D-Miki merge in dreapta

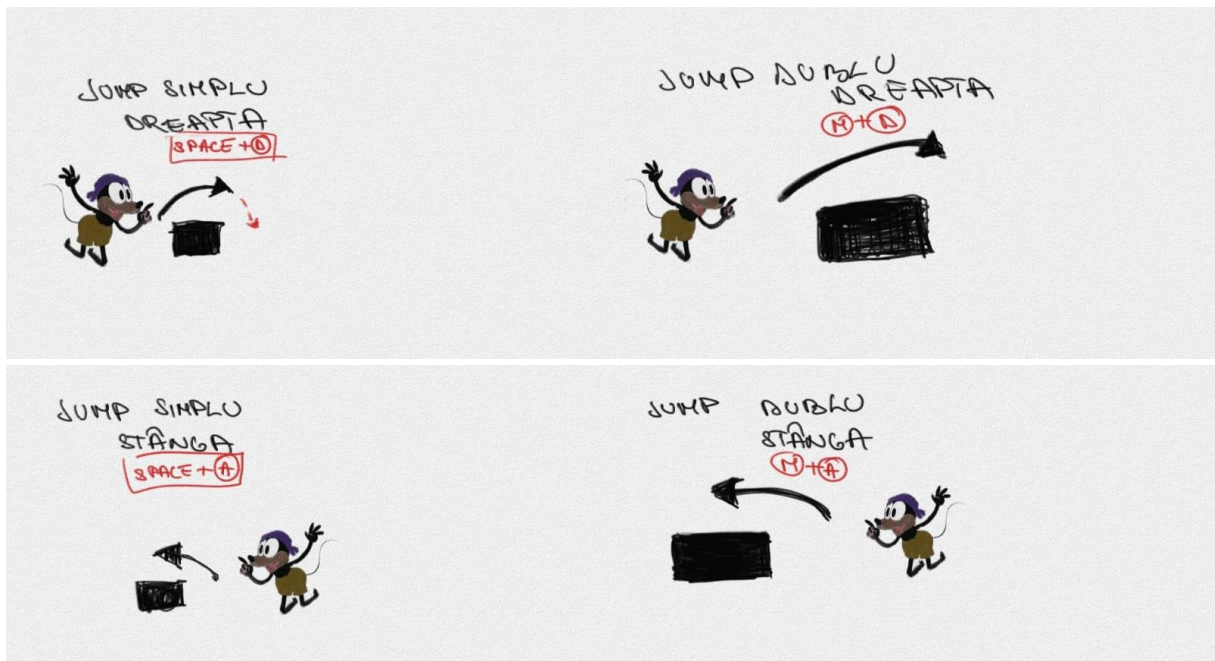
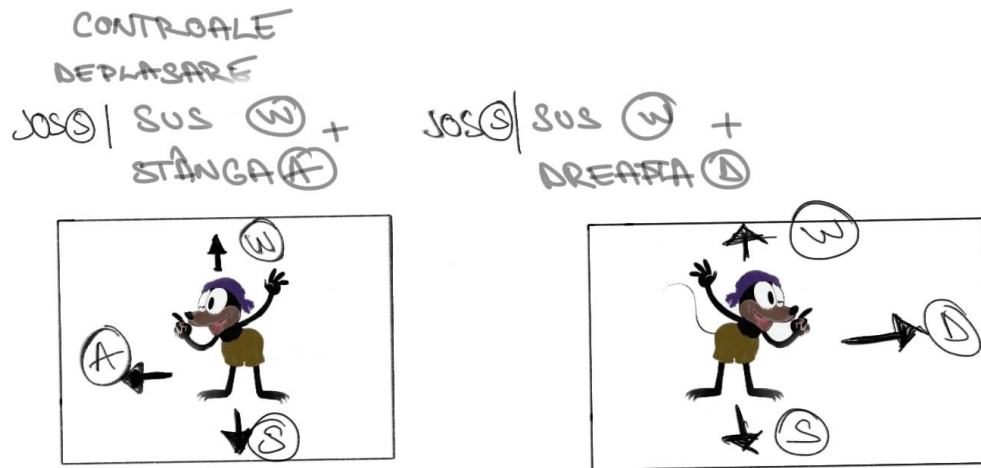
Space – sprint

Gameplay:

Miki merge prin labirint evitând capcanele și inamicii ; de asemenea încearcă să-și atragă inamicii în capcane.

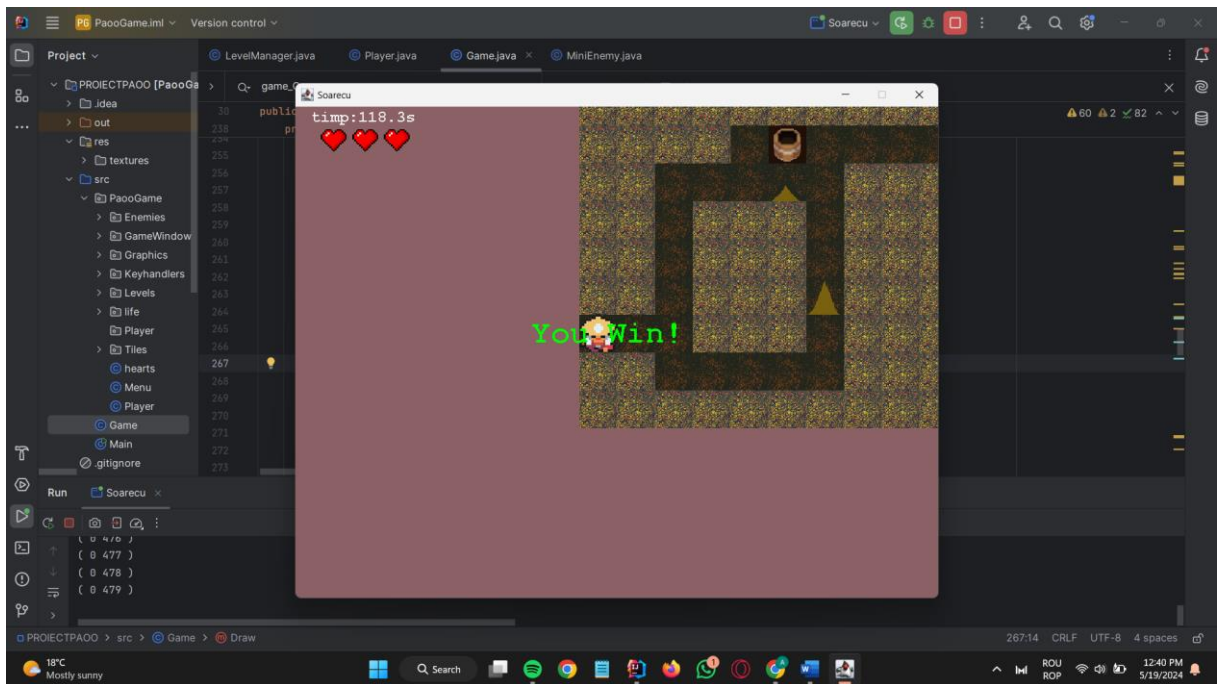
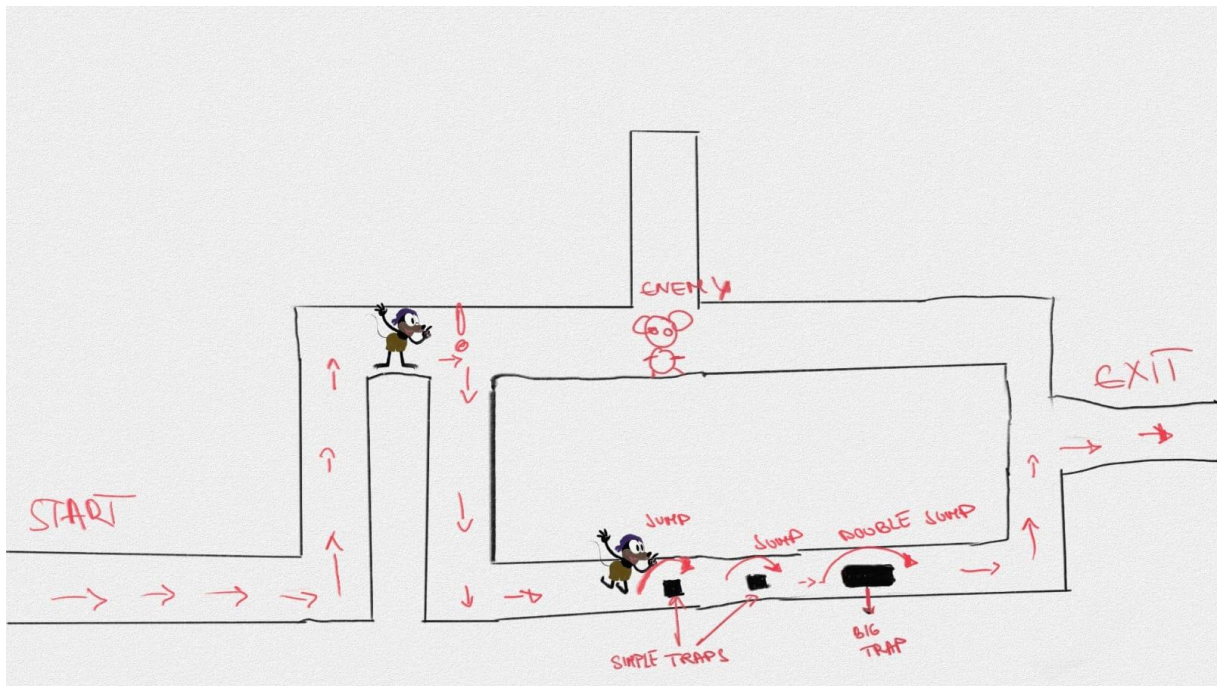
Scopul lui Miki este să evadeze din labirint pentru a se întâlni cu ceilalți hoți.

Ultraspeed oferă imunitate la capcane și viteză mai mare, dar accelerează scăderea timpului, iar jocul trebuie terminat înainte ca timer-ul să arate 0 secunde.



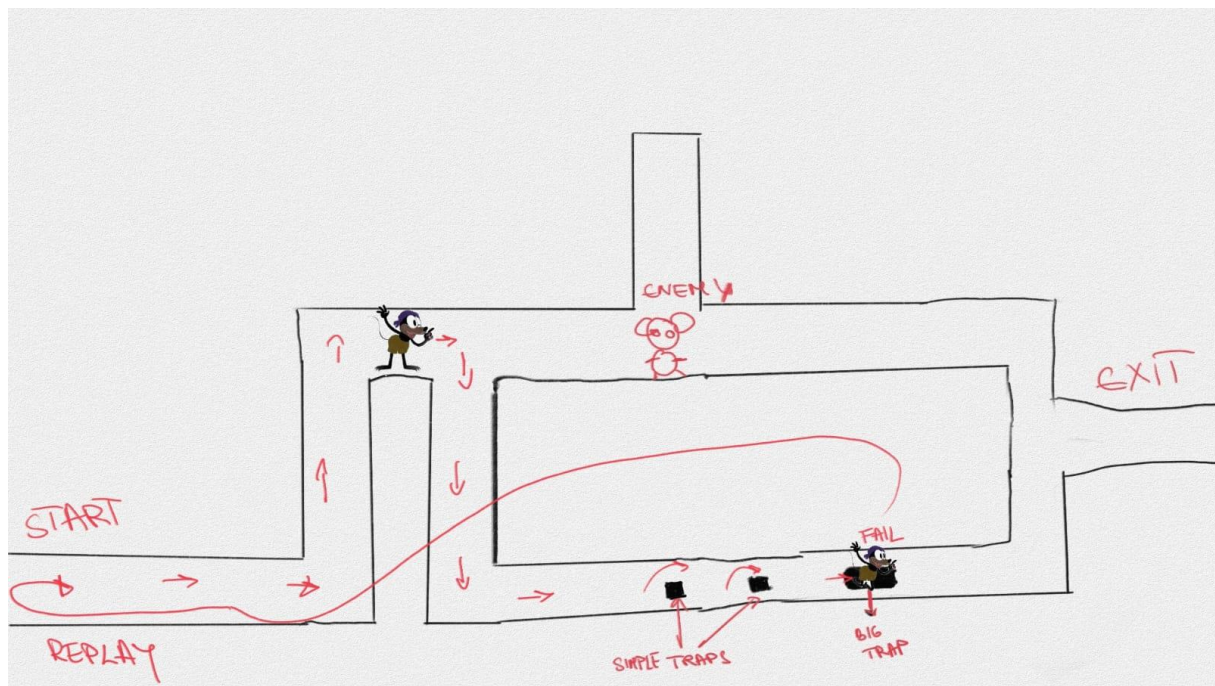
Basic gameplay, excluzând partea de jump dublu, am renunțat la ea deoarece jocul are vedere topdown și nu avea sens. Am adăugat Sprint în locul său.

Succes



Eșec

Dacă viața player-ului ajunge 0 sau timer-ul indică 0 secunde , atunci jocul trebuie reluat de la început. Viața personajului scade când intră în coliziune cu capcanele și cu inamicii.



3.Proiectarea conținutului și descriere detaliată

Caractere :

1. Miki – personajul controlat de jucător

-la nivelul 1 acesta nu poate ataca și doar aleargă și se folosește de capcane în favoarea sa

-la nivelul 2 acesta poate ataca provocând damage (cu fiecare atac) o viață inamicului și îl încetinește pentru 2 secunde

-la nivelul 3 poate face jump dublu

-își păstrează și toate proprietățile anterioare

-acesta are 3 vieți

2. Rudith – inamic prezent la fiecare nivel

-la fiecare nivel acesta stă pe loc până când îl vede pe Miki , apoi merge spre el să îl prindă

-nu atacă și este ușor de atras în capcane

-moare când intră în coliziune cu capcana

-ia player-ului 75 hp (adică tot)



3. Mini – inamic prezent începând cu nivelul 2

-ia 50 hp când intră în coliziune cu player-ul

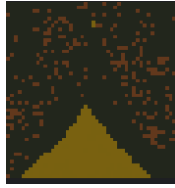
-nu ia damage de la capcane

-patrulează constant pe anumite zone din joc

Sprite-uri ce vor fi utilizate



Tile capcană



DB Browser for SQLite - C:\Users\Alexandra\OneDrive\Desktop\1211A_BuzanAlexandra_et2\PROIECTPAOI\gamebun2.db

File Edit View Tools Help

Database Structure Browse Data Edit Pragma Execute SQL

Table: GameStateBunFinal

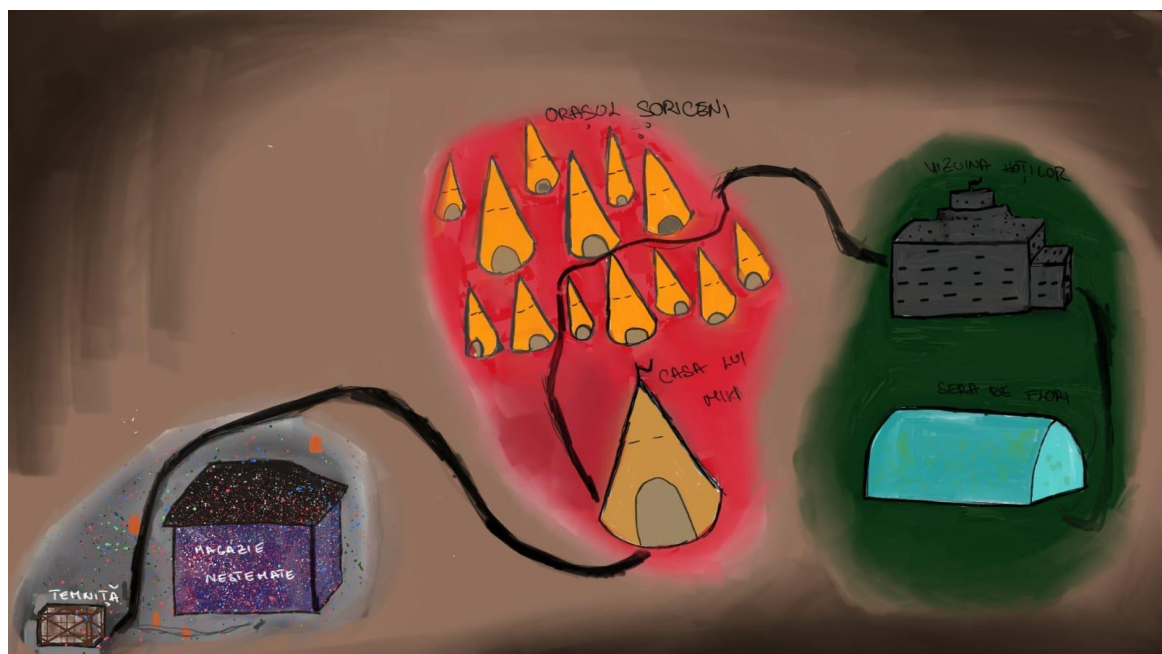
| ID | ACTUAL_HEALTH | LEVEL | MIKI_POZX | MIKI_POZY | EN_POZX | EN_POZY | TIME | MINI_POZX | MINI_POZY |
|----|---------------|-------|-----------|-----------|---------|---------|----------------------|-----------|-----------|
| 1 | 1 | 26 | 2 | 479 | 143 | 621 | 38 6.233333333333319 | 195 | 492 |

Go to: 1

SQL Log Plot DB Schema Remote UTF-8

Ultimele poziții ale personajelor, nivelul, timpul și hp-ul player-ului sunt salvate într-o bază de date în momentul în care personajul se mișcă. Astfel, după ce închid jocul, îl reiau exact de unde am rămas.

4.Proiectarea nivelurilor și descriere detaliată



Jocul are 3 niveluri

Miki trebuie să evadeze din temniță . La acest nivel este un singur inamic : Rudith . Rudith va îl va păzi , dar playerul o va atrage într-o capcană și astfel va evada. După ce evadează din temniță Miki jefuiește Magazia cu Nestemate pentru a-i mulțumi pe hoți , dar și pentru a încerca să-și repare viața. Doar țepușe de lemn.

START

SAR PESTE TERUȘA MICĂ
(INAMICUL DE NIVEL 1 NU POATE SARII)

INAMIC NIVEL 1 (RUBINĂ)
- NU POATE ATACA, TREBUIE ATACAT ÎNTR-O TERUȘĂ

INAMICUL MĂ VEDĂ
ȘI RĂMÎNE DUPĂ MINE → MĂ GRĂBESC SPRE O TERUȘĂ

MĂ GRĂBESC (CU STAU)

AICI MOARE INAMICUL

IEȘIRE

TERUȘA MARE DE LEMN

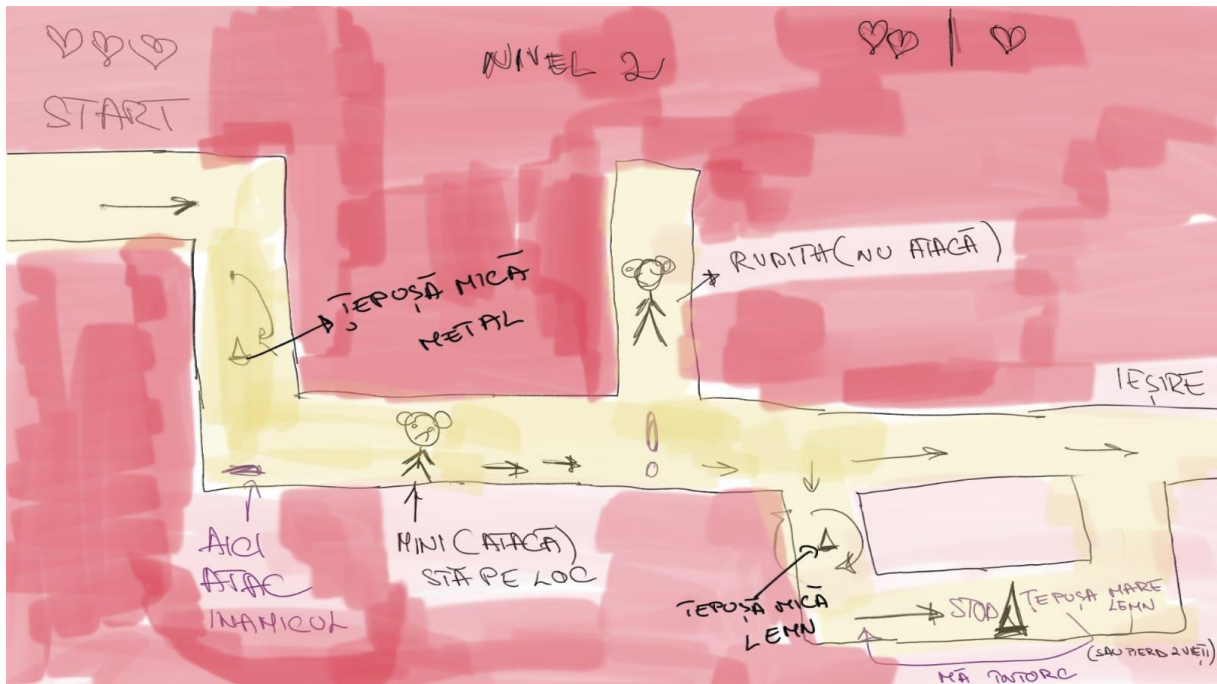
ACEASTĂ CALE (S) - 2 VIEȚI

LA NIVELUL 1 MIKI NU POATE FACE UNDA DUBLU



Nivelul 2 :

Miki trebuie să treacă prin orașul Șoriceni unde îl mituiește pe șerif , apoi se îndreaptă spre casa lui . Mini nu vrea să îl ierte pe Miki , iar aici devine al doilea inamic. Aici playerul poate ataca inamicii. Aici apar țepușele de metal.

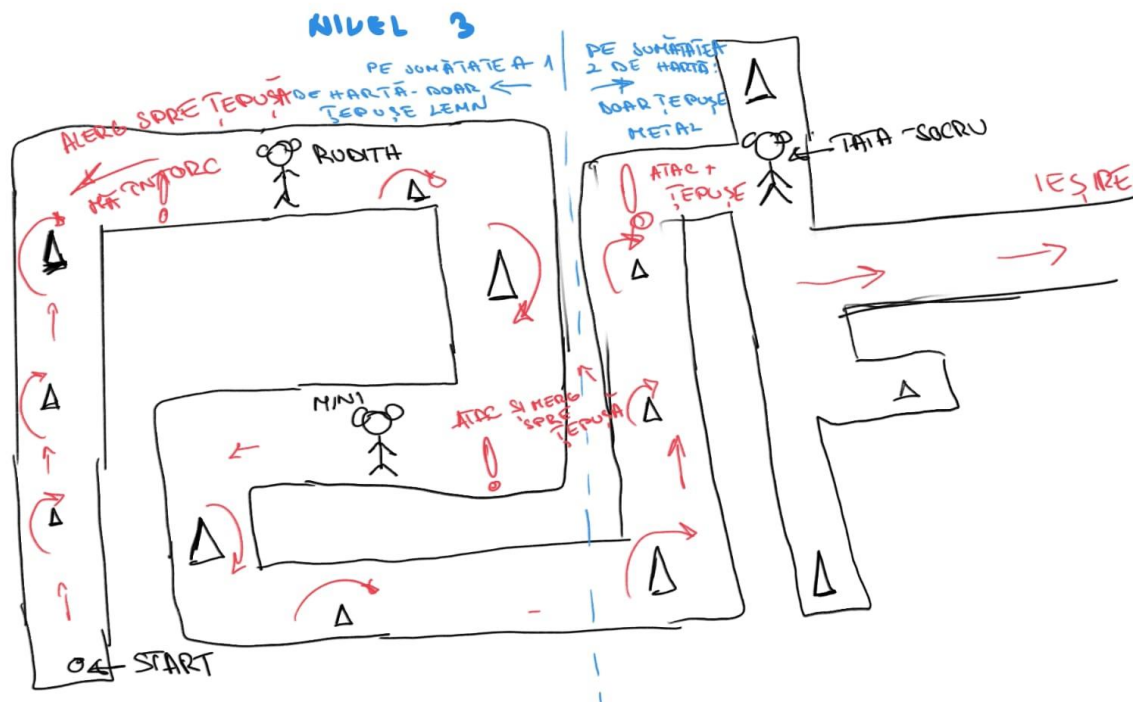


La acest nivel am 2 inamici : Rudith și Mini. Mini pleacă din dreapta hărții atașate , merge jos pe culoar , apoi reia această mișcare. Rudith mă detectează doar dacă vin de sus și mă urmărește până la prima țepușă.

Nivelul 3 :

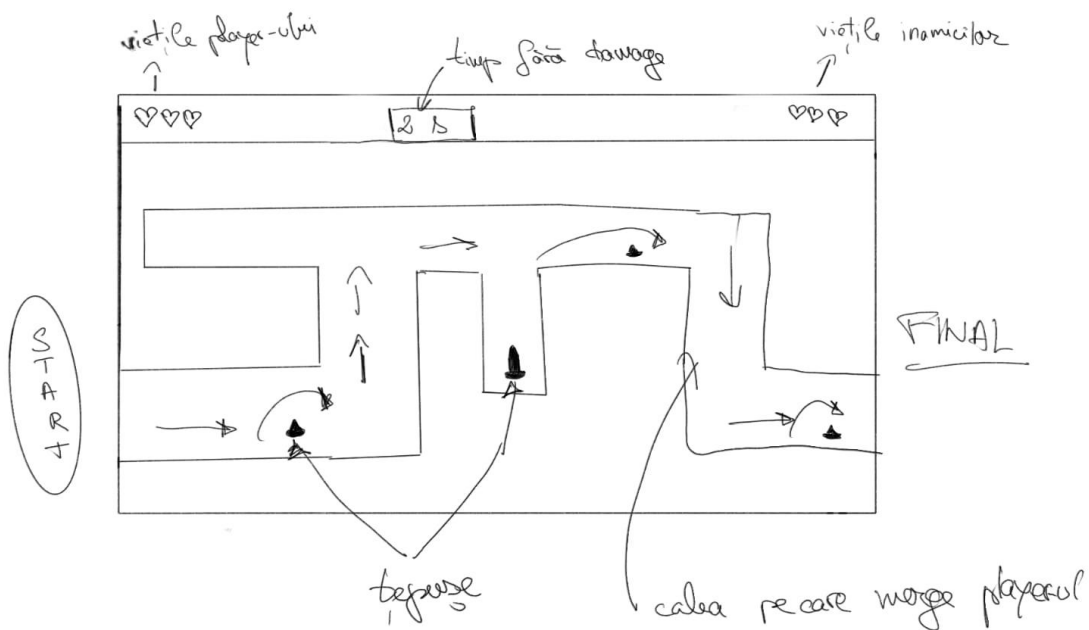
Miki trebuie să se întâlnească cu hoții și să mai încerce împăcarea cu familia , acum că și-a reparat imaginea în fața legii. Astfel , prima dată merge la vizuina hoților , unde este descoperit de Tata-Socru , tatăl lui Mini care este ofensat de modul în care Miki continuă să nu aibă regrete. După ce player-ul îl învinge pe final boss , Miki se împacă cu fosta lui iubită , Rudith și îi cumpără un buchet de flori.

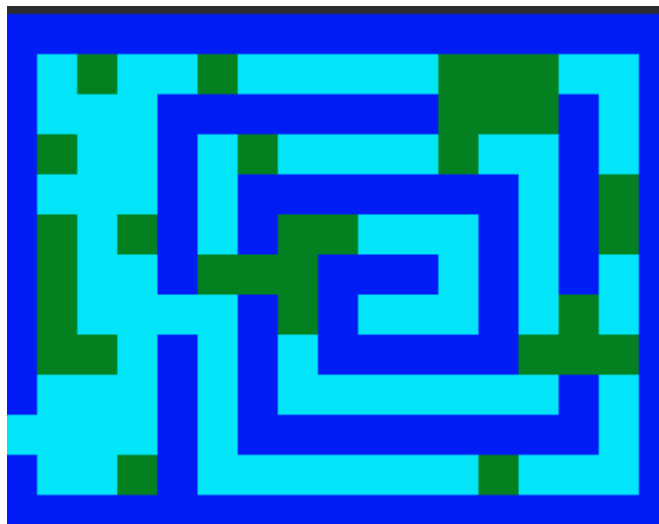




5.Proiectarea interfeței și descriere detaliată

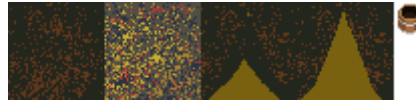
Idee generală a interfeței la fiecare nivel, harta diferă , după cum este desenată la proiectarea nivelelor.



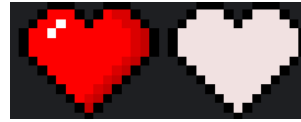


6.Sprite-uri:

Pentru hartă



Pentru hp



Pentru player



7.Bibliografie

<https://itch.io/game-assets/tag-48x48>

<https://refactoring.guru/design-patterns/observer>

<https://refactoring.guru/design-patterns/strategy>

Descrierea scheletului proiectului

Diagramă UML

-separat

Descriere clase proiect

Main

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

GameWindow

După cum discutăm mai sus, metoda run() a clasei Game creează o instanță a clasei GameWindow. Această clasă este responsabilă cu fereastra în care vor fi desenate obiectele pe un canvas (se poate observa că avem o metoda GetCanvas care întoarce canvasul). De asemenea, avem un obiect JFrame care permite desenarea de butoane, controale, textbox-uri, etc, dar poate conține și un canvas în care pot fi desenate diverse obiecte folosind texturi. Dacă de exemplu dorim să avem un meniu pentru joc, atunci sunt necesare mai multe obiecte JFrame, unul pentru meniu în care sunt desenate butoanele, etc și unul pentru acțiunea jocului, iar în funcție de interacțiunea utilizatorului acestea vor fi schimbate între ele.

Tile

Această clasă reține informații despre tile-urile (dalele) din joc. În clasa Game există o instanță a clasei Tile, deoarece clasa Tile conține un vector cu obiecte tot de tipul Tile, așadar în acest vector vor fi stocate toate „dalele/tile-urile”, din joc în așa fel încât acest vector poate fi parcurs și pentru fiecare element se apelează metoda Update() și apoi Draw(). De asemenea, clasa Tile mai reține și câte o instanță pentru fiecare sub-tip de tile (GrassTile, MountainTile, TreeTile, SoilTile și WaterTile). În funcție de hartă, acel vector de tile-uri va conține tile-uri de acest tip. GrassTile, MountainTile, TreeTile, SoilTile și WaterTile Toate aceste clase extind clasa Tile, preluând comportamentul acelei clase sau putând să-l suprascrie folosind @Override. Constructorul acestora primește ca parametru un ID prin care se va putea identifica acel tile și apelează constructorul clasei de bază, adică al clasei Tile, care primește ca parametru un membru static al clasei Assets. Membrul static folosit este ales în funcție de tipul clasei respective, de exemplu pentru GrassTile se va folosi membrul static grass.

Assets

Conține câte un membru static de tip BufferedImage pentru fiecare tile/dală, chiar și pentru jucători/inamici. În acești membri sunt stocate imaginile, adică texturile acestora care vor fi desenate prin apelarea metodelor Draw() din fiecare clasa tile apelate din metoda Draw() din clasa Game. În acest exemplu este doar o imagine care conține toate texturile folosite, așadar pentru a putea instanța aceste obiecte statice trebuie să decupăm fiecare textură din acea imagine. Pentru a face asta se folosește o instanță a clasei SpriteSheet în metoda Init(). Metoda folosită din SpriteSheet este crop(x, y).

SpriteSheet Constructorul acestei clase primește imaginea, iar clasa conține 2 membri constanți (final în Java) pentru înălțimea, respectiv lățimea texturilor (trebuie să aibă toate aceleași dimensiuni). Metoda `crop(x, y)` primește doi parametri, `x` specifică pe ce coloană se află textura pe care dorim să o decupăm în imaginea cu toate texturile, iar `y` specifică linia. Pentru a afla efectiv poziția în imagine se determină 15 pixelii de unde încep texturile de interes prin înmulțirea liniei/coloanei cu înălțimea, respectiv lățimea texturilor. Astfel se obține colțul din stânga sus al texturii, iar pentru a afla celelalte colțuri se folosesc înălțimea, respectiv lățimea.

ImageLoader

Înainte de a extrage fiecare textură, imaginea cu toate texturile trebuie să fie citită din memorie, iar pentru asta se folosește clasa `ImageLoader` cu metoda statică `LoadImage(path)` care primește ca parametru calea către imagine în memoria calculatorului.

LevelManager

`LevelManager` este o clasă fundamentală pentru joc, responsabilă de gestionarea nivelurilor și interacțiunilor cu hărțile. Stochează harta sub formă de matrice și gestionează sănătatea jucătorului. Metodele sale includ încărcarea hărților, detectarea capcanelor și verificarea coliziunilor. Matricea pentru hartă este construită folosindu-ne de programul `Aseprite` unde mă folosesc de pixelul `red` pentru a obține un `id` corespunzând fiecărui `tile`. Variabilele membru sunt :

`map`: o matrice bidimensională care stochează informații despre tipurile de dale din hartă.

`health`: un întreg care indică starea de sănătate a jucătorului.

Constructor: Constructor implicit pentru inițializarea clasei.

Metoda `GetMap(BufferedImage image)`: Este o metodă statică ce primește o imagine și convertește aceasta într-o hartă de dale. Parcurge fiecare pixel al imaginii, determinând tipul de dala pe baza culorii pixelului. Returnează harta generată. Metoda `GetDamageTraps(int x, int y)`: Verifică dacă jucătorul a intrat într-o capcană care îi cauzează daune. Reduce viața jucătorului în funcție de tipul capcanei. Metoda `getSpriteIndex(int x, int y)`: Returnează indicele dalei de la o anumită poziție pe hartă. Metoda `Checker(int x, int y)`: Verifică dacă un anumit punct de pe hartă este ocupat de o dala care împiedică deplasarea.

Level1

`Level1` este clasa care gestionează inițializarea și desenarea primului nivel al jocului. Ea încarcă imaginea nivelului dintr-un fișier specific și folosește `LevelManager` pentru a desena harta nivelului. Clasa servește drept interfață între joc și nivelul său specific.

Variabile Membru:

image: o referință către imaginea nivelului.

map: o matrice bidimensională care stochează informații despre tipurile de dale din nivel.

Constructor:

Constructorul implicit inițializează variabila map la valoarea null.

Metoda InitLvl1():

Încarcă imaginea nivelului dintr-un fișier specificat în calea relativă /textures/mapaversiunenoua.png folosind clasa ImageLoader.

Metoda DrawLevel(Graphics g):

Desenează nivelul utilizând metoda DrawMap() din LevelManager, pasând imaginea nivelului și contextul grafic Graphics.

Metoda GetMap():

Returnează harta nivelului.

KeyHandler

KeyHandler este o clasă care implementează interfața KeyListener pentru a gestiona evenimentele de tastatură. Ea urmărește starea tastelor W, A, S, D, SPACE și M pentru a indica direcția de mișcare a jucătorului și pentru a activa săriturile. Clasa oferă metode pentru a verifica starea acestor taste.

Variabile de stare: Clasa are șase variabile private de stare (sus, jos, stanga, dreapta, jump, jumpdublu), care rețin starea tastelor asociate acțiunilor de sus, jos, stânga, dreapta, salt și dublu salt. Aceste variabile sunt setate la true atunci când tasta corespunzătoare este apăsată și la false atunci când tasta este eliberată.

Constructorul: Constructorul este lăsat gol pentru moment. Acesta poate fi utilizat pentru inițializarea variabilelor sau alte operațiuni de pregătire.

Metode pentru gestionarea evenimentelor de tastatură:

keyPressed(KeyEvent e): Această metodă este apelată atunci când o tastă este apăsată. În această metodă, se verifică codul tastelor și se setează variabilele de stare corespunzător.

keyReleased(KeyEvent e): Această metodă este apelată atunci când o tastă este eliberată. Similar cu metoda keyPressed, aceasta verifică codul tastelor și actualizează variabilele de stare corespunzător.

keyTyped(KeyEvent e): Această metodă este apelată atunci când o tastă este tastată, dar nu este folosită în acest exemplu.

Metode pentru interogarea stării tastelor:

Pentru fiecare acțiune (sus, jos, stanga, dreapta, jump, jumpdublu), există o metodă publică care întoarce starea curentă a acelei acțiuni.

Enemy

Enemy este o clasă care extinde LevelManager și definește caracteristicile și comportamentul inamicilor din joc. Aceasta ține evidența poziției, sănătății și tipului de inamic. Clasa oferă metode pentru a desena inamicul pe ecran.

Variabile de stare:

enPozX și enPozY: Acestea sunt coordonatele X și Y ale inamicului pe ecran.

health: Variabilă pentru viața inamicului.

id: Un identificator pentru tipul de inamic. Deși nu este folosit în constructorul furnizat, probabil este destinat să distingă între diferitele tipuri de inamici.

EnemyType: Pare să fie o variabilă pentru tipul de inamic, dar nu este folosită în codul furnizat.

Variabile pentru dimensiunile inamicului:

EnemyWidth și EnemyHeight: Dimensiunile inamicului în pixeli. Acestea sunt setate la o valoare fixă de 48x48 pixeli.

Imaginea inamicului:

img: Este o imagine BufferedImage care reprezintă aspectul inamicului. În constructor, aceasta este setată în funcție de id. În codul furnizat, doar tipul de inamic cu id 1 are o imagine asociată, din Assets.

Constructor:

Constructorul primește coordonatele x și y ale inamicului și un id pentru a specifica tipul inamicului. Acesta inițializează variabilele de stare corespunzător și setează imaginea inamicului în funcție de id.

Metoda DrawEnemy(Graphics g):

Această metodă desenează inamicul pe ecran folosind contextul grafic g. Aceasta plasează imaginea inamicului la coordonatele enPozX și enPozY, cu dimensiunile definite de EnemyWidth și EnemyHeight.

RudithEnemy

RudithEnemy este o clasă care reprezintă un tip specific de inamic în joc. Aceasta definește dimensiunile și poziția inamicului, precum și imaginea asociată cu acesta. Clasa oferă metode pentru a desena inamicul pe ecran și pentru a actualiza imaginea inamicului în funcție de acțiunile acestuia.

Variabile de stare:

rudithPozX și rudithPozY: Acestea sunt coordonatele X și Y ale inamicului Rudith pe ecran.

rudithW și rudithH: Dimensiunile în pixeli ale inamicului Rudith. Acestea sunt setate la o valoare fixă de 48x48 pixeli.

imgRow și imgCol: Variabile pentru a specifica rândul și coloana imaginii inamicului dintr-o anumită imagine de ansamblu. Acestea pot fi utilizate pentru a afișa diverse animații sau stări ale inamicului.

img: Imaginea BufferedImage care reprezintă aspectul inamicului Rudith. În constructor, aceasta este setată inițial la prima imagine dintr-o anumită matrice de imagini (Assets.rudith[0][0]).

Constructor:

Constructorul primește coordonatele x și y ale inamicului Rudith și inițializează variabilele de stare corespunzător. De asemenea, setează imaginea inamicului la prima imagine dintr-o anumită matrice de imagini.

Metoda DrawEnemy(Graphics g):

Această metodă desenează inamicul Rudith pe ecran folosind contextul grafic g. Plasează imaginea inamicului la coordonatele rudithPozX și rudithPozY, cu dimensiunile definite de rudithW și rudithH.

Metode pentru actualizarea și randarea inamicului:

Update(int row, int col): Această metodă actualizează imaginea inamicului la o anumită poziție dintr-o matrice de imagini. Acest lucru poate fi util pentru a schimba animațiile sau stările inamicului.

Render(Graphics g): Această metodă randează inamicul pe ecran prin apelarea metodei DrawEnemy(Graphics g).

Player

Importurile sunt utilizate pentru a aduce alte clase și resurse în acest fișier. De exemplu, sunt importate clasele KeyHandler și LevelManager, precum și imagini și alte resurse grafice necesare pentru joc.

Această clasă definește comportamentul și aspectul jucătorului în cadrul jocului, gestionându-i mișcările, viața și interacțiunile cu alte entități din joc.

Atributele clasei:

mikiWidth, mikiHeight: Dimensiunile jucătorului.

mikiPozX, mikiPozY: Poziția jucătorului pe axele X și Y.

imgRow, imgCol: Indicii rândului și coloanei pentru imaginea jucătorului.

health: Viața jucătorului.

img: Imaginea jucătorului.

direction: Direcția în care se mișcă jucătorul.

miki: O referință statică la instanța jucătorului, utilizată pentru a implementa un singleton.

Constructorul clasei:

Constructorul este privat, ceea ce înseamnă că nu poate fi accesat din exteriorul clasei. Acest lucru este făcut pentru a asigura că obiectele de tip jucător sunt create folosind metoda createPlayer, care implementează un singleton.

Metoda createPlayer:

Implementează un singleton pentru a asigura că există întotdeauna doar o singură instanță a jucătorului.

Metodele clasei:

DrawPlayer(Graphics g): Desenează jucătorul pe ecran.

setImgRow, setImgCol, getImgRow, getImgCol: Metode pentru a seta și obține rândul și coloana imaginii jucătorului.

Update: Actualizează imaginea jucătorului în funcție de rândul și coloana specificate.

MovementMiki: Gestionează mișcarea jucătorului în funcție de tastele apășate.

Metodele moveup, movedown, moveleft, moveright: Implementează mișcările jucătorului în diferite direcții.

DamageEnemy: Metodă care verifică dacă jucătorul a fost lovit de un inamic și aplică daunele corespunzătoare.

Render: Metodă care desenează jucătorul pe ecran.

Comentarii în cod:

Codul este însoțit de unele comentarii care explică logică din spatele anumitor porțiuni de cod și decizii de proiectare.

BarrelTile

În ansamblu, clasa BarrelTile definește comportamentul și aspectul unei dale de butoi în cadrul jocului. Ea indică că această dala este solidă și trebuie tratată ca obstacol în logică și fizică. Utilizarea imaginii din Assets.barrel îi conferă aspectul vizual corespunzător în joc.

Extinderea clasei de bază Tile:

Clasa BarrelTile extinde clasa Tile , ceea ce înseamnă că va moșteni anumite comportamente și caracteristici definite în clasa de bază Tile.

Constructorul clasei:

Constructorul primește un ID și apelează constructorul clasei de bază (super) cu ID-ul și imaginea asociată. În acest caz, imaginea asociată este Assets.barrel, care este încărcată din altă clasă (probabil din pachetul PaoGame.Graphics) și reprezintă aspectul vizual al dalei.

Metoda IsSolid:

Această metodă este suprascrisă (@Override) din clasa de bază.

Returnează o valoare booleană care indică dacă dala este solidă sau nu. În acest caz, metoda întoarce true, ceea ce înseamnă că dala de butoi este solidă și nu poate fi traversată de către alte entități din joc.

bigwoodspikeTile

Clasa bigwoodspikeTile definește comportamentul și aspectul unei dale cu un mare cuțit de lemn în cadrul jocului. Ea indică că această dala este solidă și trebuie tratată ca un obstacol în logică și fizică.

groundTile

Clasa groundTile definește comportamentul și aspectul unei dale de sol în cadrul jocului. Ea indică că această dala este solidă și trebuie tratată ca un obstacol în logică și fizică.

Constructorul clasei:

Constructorul primește un ID și apelează constructorul clasei de bază (super) cu ID-ul și imaginea asociată. În acest caz, imaginea asociată este Assets.ground, care este încărcată din altă clasă (probabil din pachetul PaoGame.Graphics) și reprezintă aspectul vizual al dalei.

Metoda IsSolid:

Această metodă este suprascrisă (@Override) din clasa de bază.

Returnează o valoare booleană care indică dacă dala este solidă sau nu. În acest caz, metoda întoarce true, ceea ce înseamnă că dala de sol este solidă și nu poate fi traversată de către alte entități din joc.

roadTile și bigwoodspikeTile – analog

Rudith

Extinderea clasei de bază Tile:

Clasa Rudith extinde clasa Tile, ceea ce sugerează că este o variantă specializată a unei dale generice. Aceasta înseamnă că va moșteni anumite comportamente și caracteristici definite în clasa de bază Tile.

Atributele clasei:

rudithH, rudithW: Înălțimea și lățimea imaginii pentru dala Rudith.

Constructorul clasei:

Constructorul primește un ID și coordonatele x și y ale imaginii din Assets.rudith. Apoi, el apelează constructorul clasei de bază (super) cu imaginea asociată și ID-ul. Imaginile sunt obținute dintr-o matrice Assets.rudith[x][y] și reprezintă aspectul vizual al dalei.

Game

Aceasta este cea mai importantă clasă din cod. Aici au loc majoritatea inițializărilor, sunt create metodele pentru salvarea jocului într-o bază de date, sunt implementate clase pentru desenarea imaginilor, timpul, scrierea textului pentru Game Over sau Win etc.

Tot aici sunt apelate toate metodele de deplasare ale inamicilor și ale player-ului.

Design Pattern-urile abordate în codul meu sunt Observer, Strategy și Singleton.

Am folosit în clasa Player design pattern-ul Singleton făcându-mi constructorul private și scriind o nouă metodă publică în care verific dacă am creat deja un Player . Apelez noua metodă în Game pentru a crea un nou Player.

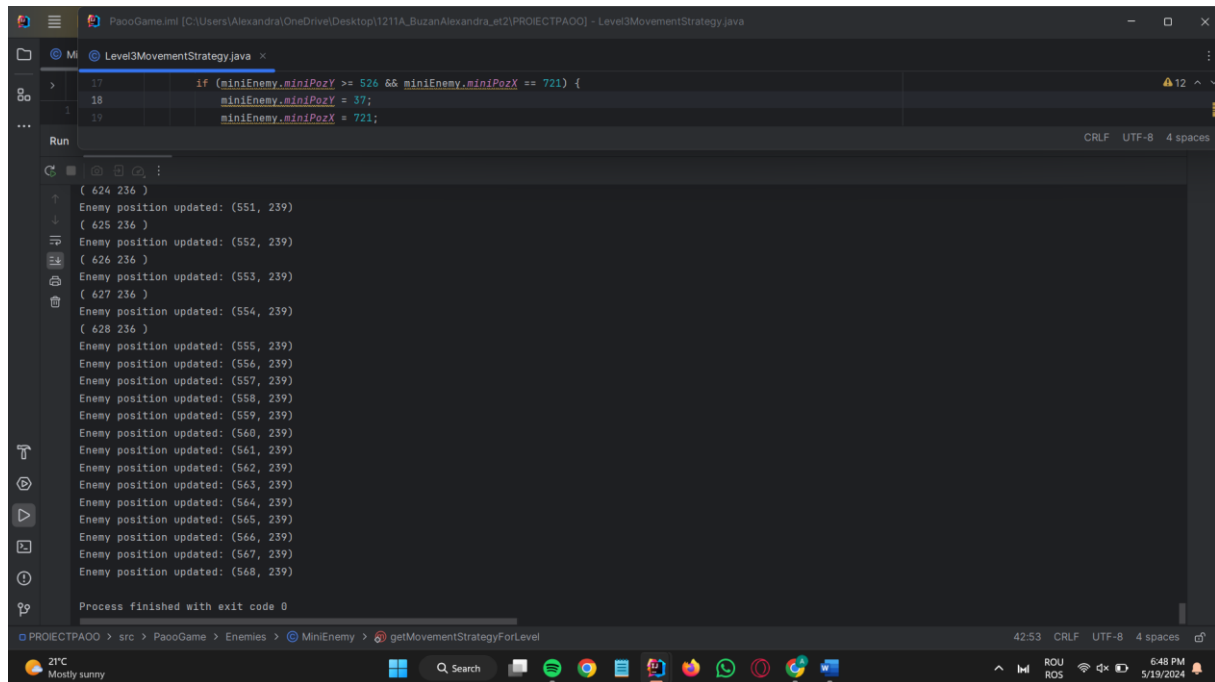
În clasa Ruddith am folosit modelul Observer care notifică ori de câte ori se modifică poziția inamicului de acest tip.

Subject : Obiectul care deține starea și notifică observatorii despre schimbările sale.

Observer : Obiectele care doresc să fie notificate despre schimbările subiectului.

Interfața Subject declară metode pentru adăugarea, eliminarea și notificarea observatorilor. Interfața Observer declară metoda update care va fi apelată de subiect pentru a notifica observatorul. Clasa Ruddith implementează interfața Subject și menține

o listă de observatori. Atunci când starea sa se schimbă, notifică toți observatorii prin apelarea metodei update. Clasa PlayerObserver implementează interfața Observer și definește comportamentul care trebuie executat atunci când este notificată.



```
Level3MovementStrategy.java
17     if (miniEnemy.miniPozY >= 526 && miniEnemy.miniPozX == 721) {
18         miniEnemy.miniPozY = 37;
19         miniEnemy.miniPozX = 721;
    }

Run
( 624 236 )
Enemy position updated: (551, 239)
( 625 236 )
Enemy position updated: (552, 239)
( 626 236 )
Enemy position updated: (553, 239)
( 627 236 )
Enemy position updated: (554, 239)
( 628 236 )
Enemy position updated: (555, 239)
Enemy position updated: (556, 239)
Enemy position updated: (557, 239)
Enemy position updated: (558, 239)
Enemy position updated: (559, 239)
Enemy position updated: (560, 239)
Enemy position updated: (561, 239)
Enemy position updated: (562, 239)
Enemy position updated: (563, 239)
Enemy position updated: (564, 239)
Enemy position updated: (565, 239)
Enemy position updated: (566, 239)
Enemy position updated: (567, 239)
Enemy position updated: (568, 239)
Process finished with exit code 0
```

Am folosit în clasa MiniEnemy design pattern-ul Strategy pentru a controla mișcarea acestui inamic. Pentru fiecare nivel am o nouă strategie de mișcare și am 2 clase pentru asta care implementează interfața metoda move() din interfața Movement Strategy . Am adăugat o strategie și în constructorul clasei MiniEnemy.

Pachete :

Enemies – aici am toate clasele și interfețele folosite pentru șabloanele de proiectare Strategy și Observer .

Game Window : pachet din scheletul dat în care am clasa ce îmi returnează canvas-ul și cu care pot să-mi creez imaginea jocului .

Graphics : pachet din scheletul dat în care am clasele și metodele ce se ocupă de imagini(încărcare, cropare etc).

Keyhandlers : am keyhandler-ul care verifică dacă am apăsat tastele W,A,S,D,Space.

Levels : aici am metodele ce se ocupă de setarea fiecarui nivel și realizarea hărții.

Life : aici am constructorul și metodele pentru inimioarele ce indică hp-ul player-ului.

Player : aici mă ocup de mișcarea și animațiile player-ului .

Explicarea algoritmilor de coliziuni

```
private boolean CheckEnemyCollision() 2 usages
{
    int sideLength=48;
    int mikiRight = mikiPozX + sideLength;
    int mikiBottom = mikiPozY + sideLength;

    int miniRight = enPozX + sideLength;
    int miniBottom = enPozY + sideLength;

    // stanga-dreapta
    if (mikiPozX >= miniRight || enPozX >= mikiRight) {
        return false;
    }

    // sus-jos
    if (mikiPozY >= miniBottom || enPozY >= mikiBottom) {
        return false;
    }

    return true;
}
```

Algoritmul pentru coliziunea player-inamic : am făcut analogia cu două pătrate care nu se suprapun niciodată, profitând de faptul că toate personajele au dimensiunile 48x48. Astfel, dacă nu se suprapun, nu există coliziuni. Am folosit și faptul că, coordonatele (x,y) indică, colțul de stânga sus al unui pătrat imaginar cu latura 48.

```
27
28 public void moveright(KeyHandler keyH) throws InvalidPlayerPositionException { 1 usage
29     if (direction.equals("right") && !Checker(x: mikiPozX + mikiWidth - 1, y: mikiPozY + mikiHeight - 15) && !Checker(x: mikiPozX + mikiWidth - 1, y: mikiPozY + 15) && miki
30     {
31         if ((actual_level == 2 && mikiPozX >= 768 && mikiPozY >= 382) || (actual_level == 1 && mikiPozX >= 765 && mikiPozY >= 426)) {
32             throw new InvalidPlayerPositionException();
33         }
34         mikiPozX += speed;
35         imgRow = 3;
36         System.out.println(" " + mikiPozX + " " + mikiPozY + " ");
37         imgCol = (imgCol + 1) % 4;
38         Update(imgRow, imgCol);
39         direction = "right";
40         SpikeDamageRight();
41         isRunning = false;
42     }
43 }
44 }
45 }
```

Pentru coliziunile cu pereții hărții m-am folosit tot de faptul că dimensiunile tile-urilor sunt de 48x48, dar am lăsat o distanță mai mare pentru flexibilitatea player-ului pe drum.

Am folosit același raționament și pentru coliziunile cu capcanele.

Descriere algoritmi deplasare inamici :

```
1
2 package PaoGame.Enemies;
3
4 import static PaoGame.Levels.LevelManager.actual_health;
5
6 public class Level3MovementStrategy implements MovementStrategy { 3 usages
7     @Override 1 usage
8     public void move(MiniEnemy miniEnemy) {
9         if (miniEnemy.miniPozY < 527 && miniEnemy.miniPozX == 721) {
10             miniEnemy.miniPozY++;
11         }
12         if (miniEnemy.miniPozY == 527 && miniEnemy.miniPozX >= 241) {
13             miniEnemy.miniPozX--;
14         }
15         if (miniEnemy.CheckPlayerEnemyCollision())
16             actual_health -= 50;
17         if (miniEnemy.miniPozY >= 526 && miniEnemy.miniPozX == 721) {
18             miniEnemy.miniPozY = 37;
19             miniEnemy.miniPozX = 721;
20         }
21     }
22 }
```

Inamicul care apare începând cu nivelul 2 se deplasează pe două direcții constant, schimbându-și coordonatele conform codului atașat . După ce ajunge la finalul destinației sale, își reia deplasarea de la început. Am folosit șablonul Strategy pentru această deplasare, având o interfață MovementStrategy implementată de două clase ce definesc strategii de deplasare pentru fiecare nivel la care apare personajul.

Pentru inamicul ce apare începând cu nivelul 1

```
53
54 public static void StartEnemyFollowing(int x, int y1, int y2) { 2 usages
55     if (mikiPozX == x && (mikiPozY >= y1 && mikiPozY <= y2)) {
56         startFollowing = true;
57     }
58 }
```

Am implementat o metodă care transmite inamicului că trebuie să mă urmărească în momentul în care player-ul ajunge pe axa xoy pe punctul x și un punct y cuprins pe lățimea drumului.

```

public void EnemyFollowingLvl1(int xr, int yr) { 1 usage
    StartEnemyFollowing( x: 478, y1: 221, y2: 240);
    if (startFollowing) {
        move_down( x: 239);

        if (enPozY == 239) {
            if (actual_health > 0 && enPozX != 568) {
                enPozX += 1;
                notifyObservers();
                if (enPozX == mikiPozX - 2 || enPozY == mikiPozY + 33) {
                    actual_health = 0;
                    startFollowing = false;
                }
            }
        }

        if (enPozX == 568 && enPozY == 239) {
            enemy1death = true;
            startFollowing = false;
        }
    }
}

```

Inamicul mă detectează doar dintr-o anumită poziție, mă își începe deplasarea pe coordonatele specificate și moare când intră în coliziune cu capcanele.