TRAIN RUSH
RAILS TO GLORY

Student: Vasilca Rareș-Mihai
Grupa: 1212B

STORY:

Într-o zi, în timp ce pășea prin străzile înguste ale orașului, Erald, un constructor priceput și curajos, aude chemarea disperată a consiliului orașului. Aceștia îl roagă să rezolve problema izolării orașului, construind o cale ferată care să lege Ardelia de orașele învecinate, deschizând astfel orașului posibilități nelimitate de comerț și dezvoltare.

Erald acceptă cu inima plină de determinare și hotărâre. Fără să ezite, își pregătește echipamentul și pornește pe drumul său plin de provocări. Prima etapă a călătoriei îl poartă prin pădurea densă, unde furtunile de zăpadă și creaturile sălbatice îi pun la încercare abilitățile și curajul.

Ajuns peste acest obstacol, Erald continuă să se lupte cu condițiile aspre ale unei zone înghețate, traversând munții înzăpeziți și viscolele puternice care îi pun la încercare rezistența și determinarea.

Ultima parte a călătoriei îl conduce prin aridul deșert, unde furtunile de nisip și lipsa apei îl pun în fața unor noi provocări. Cu toate acestea, cu determinare și hotărâre, Erald reușește să încheie construcția căii ferate, deschizându-i orașului Ardelia ușile spre o nouă eră de prosperitate și conectivitate.

Astfel, Erald devine eroul neînfricat al orașului, iar calea ferată pe care a construit-o, cunoscută acum sub numele de "Train Rush: Rails to Glory", devine legendară pentru călătorii și aventurieri din întreaga lume.

1. Proiectarea contextului și descriere detaliată:

Train Rush: Rails to Glory este un joc de aventură și strategie cu elemente de crafting și supraviețuire. Jucătorul se află în pielea unui constructor de cale ferată care trebuie să construiască șine pentru a asigura că trenul poate ajunge în siguranță la destinație. Timpul este crucial, iar jucătorul trebuie să gestioneze resursele și să evite inamicii pentru a finaliza traseul.

Poveste incipientă:

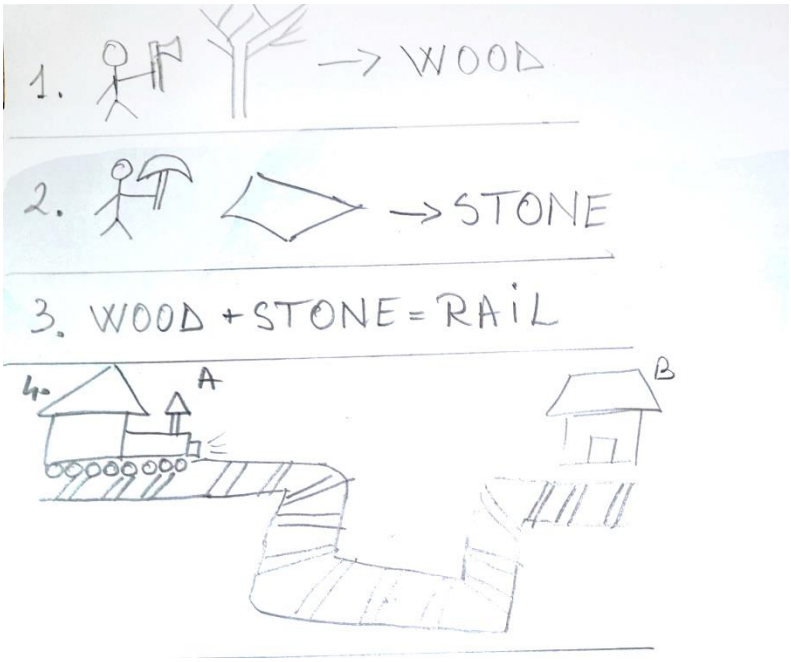
Într-o lume în care căile ferate sunt viața și sângele comunicațiilor, un constructor priceput este chemat să asigure conectivitatea între orașe. Cu resurse limitate și presiunea timpului crescând constant, constructorul trebuie să se angajeze într-o călătorie periculoasă și plină de provocări pentru a finaliza calea ferată și a aduce gloria în orașele pe care le conectează.

2. Proiectarea sistemului și descriere detaliată:

Gameplay:

- Jucătorul începe în punctul A.
- Trebuie să colecteze resurse (lemn și piatră) din mediul înconjurător pentru a putea construi șine pentru tren.
- Pe măsură ce trenul avansează, jucătorul trebuie să construiască șine în fața lui pentru a-l menține pe cale și a evita deraierea.
- Inamicii pot apărea pe parcurs și pot ataca jucătorul sau pot fura resursele sale, punând în pericol progresul său.
- Jucătorul trebuie să gestioneze resursele și să prioritizeze construcția șinelor în funcție de teren și de presiunea timpului.

- Jucătorul trebuie să ajungă cu trenul la punctul B înainte de a rămâne fără timp sau de a derai trenul.



- Fiecare nivel este considerat completat cu succes atunci când trenul ajunge la destinație fără să deraiere.

3. Proiectarea conținutului și descriere detaliată:

Resurse:

- Lemn: utilizat pentru construirea șinelor.
- Piatră: utilizată pentru construirea șinelor și a unor structuri de apărare împotriva inamicilor.

Inamici:

- Bandiți: pot ataca jucătorul sau pot fura resursele sale.
- Creaturi sălbatice: pot bloca drumul sau pot ataca jucătorul.

4. Proiectarea nivelurilor și descriere detaliată

ivelul 1: The forest

- Teren accidentat și bogat în resurse de lemn si piatra.
- Dificultate relativ ușoară, jucătorul este introdus în lumea jocului.

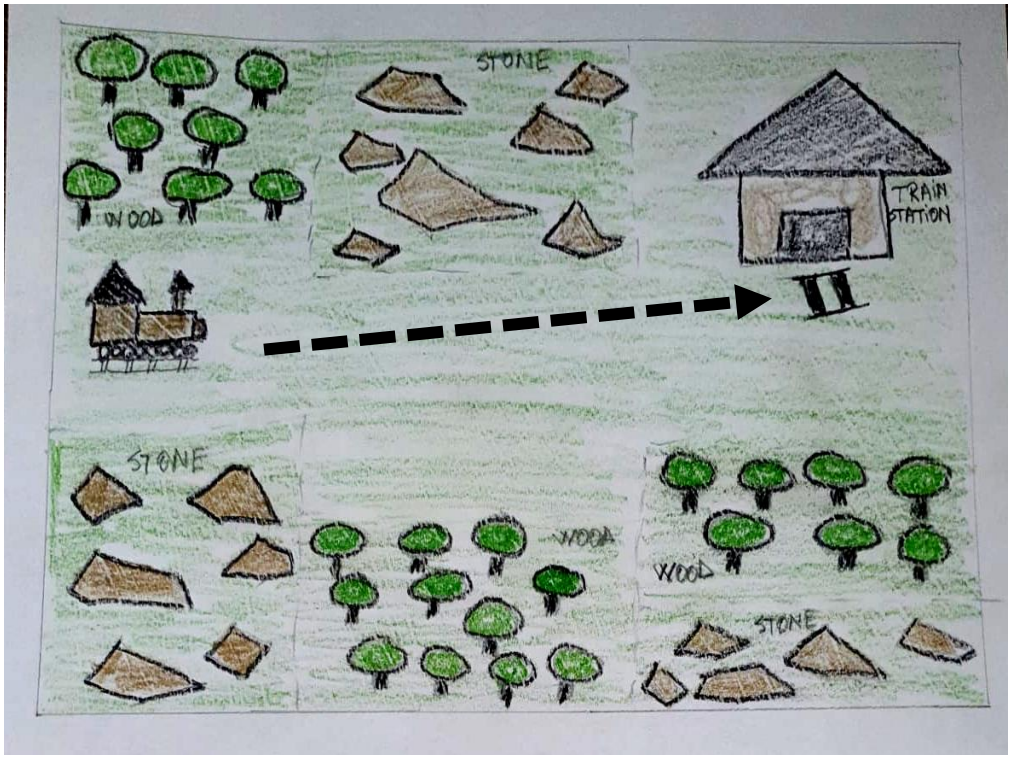
Nivelul 2: Iceland

- Teren înghețat și dificil de traversat.
- Furtuni de zăpadă ce încurcă vizibilitatea pe parcursul nivelului.

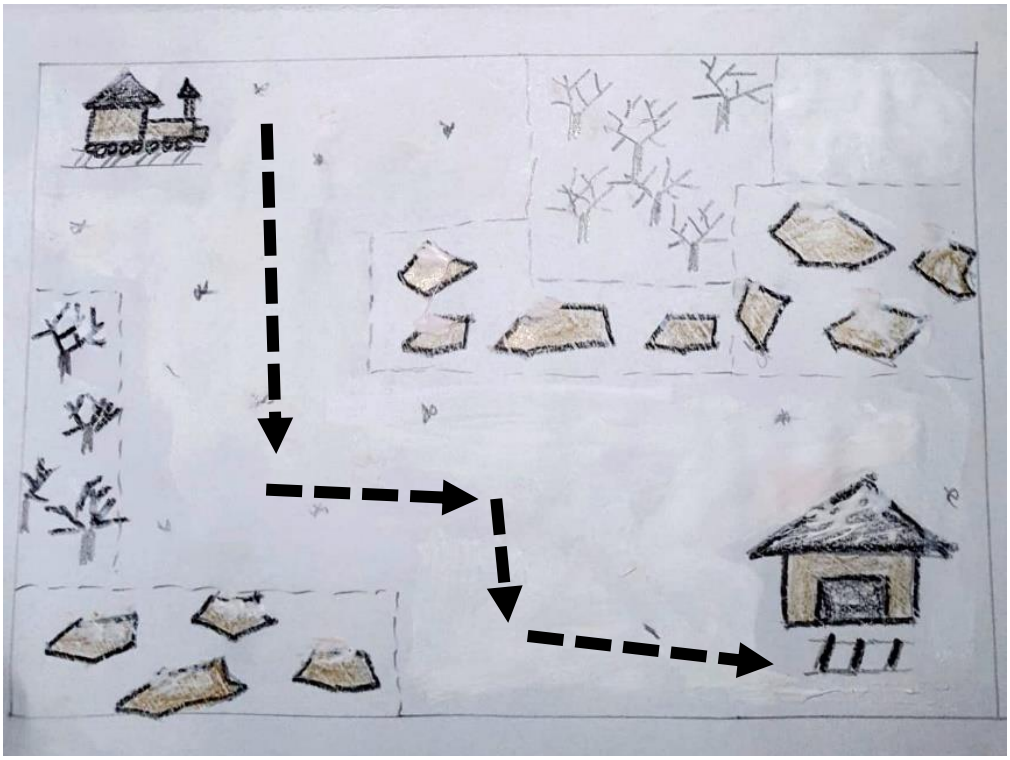
Nivelul 3: Desert

- Teren arid și fierbinte, cu resurse limitate.
- Vizibilitate redusă din pricina furtunilor de nisip
- Resursele sunt din ce în ce mai limitate, jucătorul trebuie să-și facă calculele corecte astfel încât lemnul și piatra să-i ajungă până la finalul călătoriei.
- Punctul B este înconjurat de copaci uscați și zone de piatră. Jucătorul trebuie să-și facă loc să treacă cu șinele printre acestea.

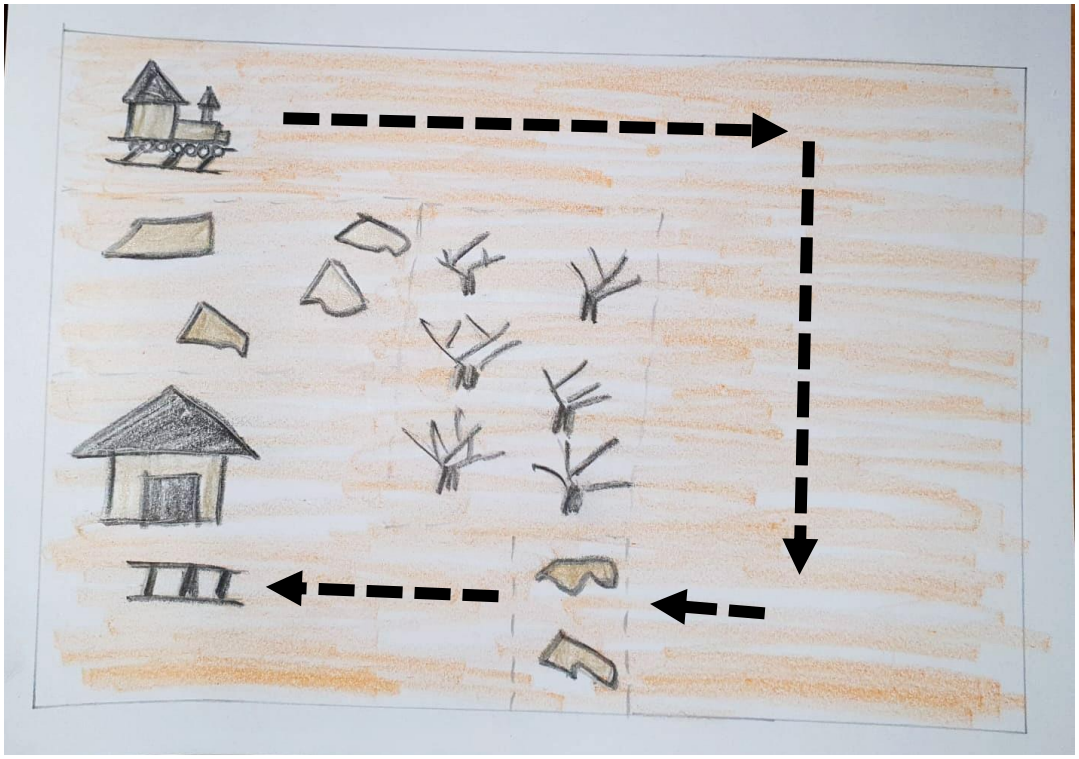
Nivel 1. The forest



Nivel 2. Iceland



Nivel 3. Desert



5. Proiectarea interfeței cu utilizatorul și descriere detaliată

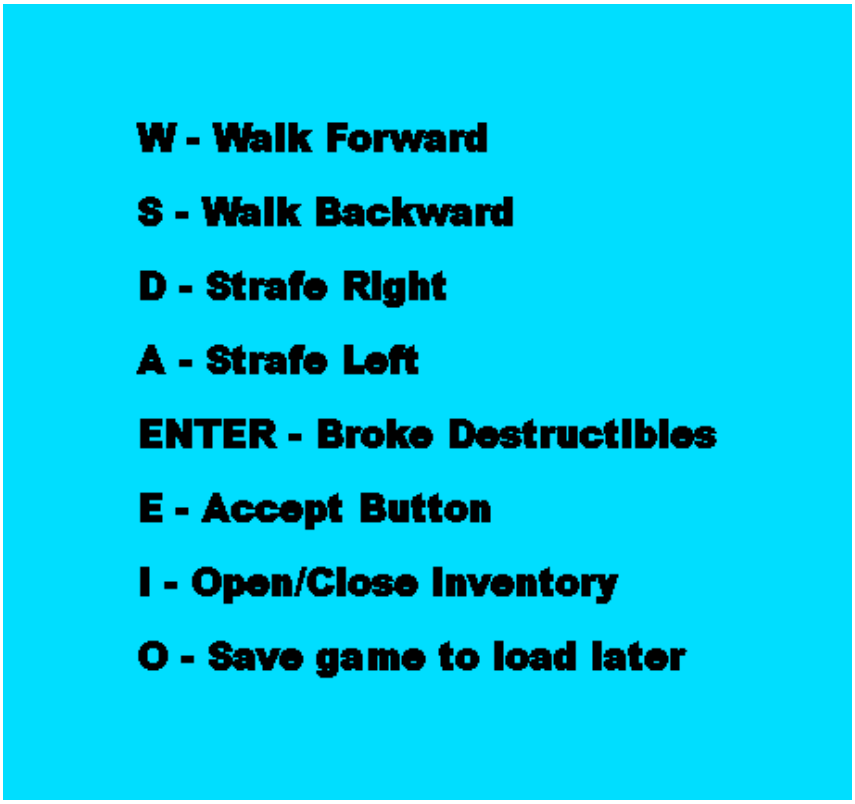
Interfața utilizatorului:

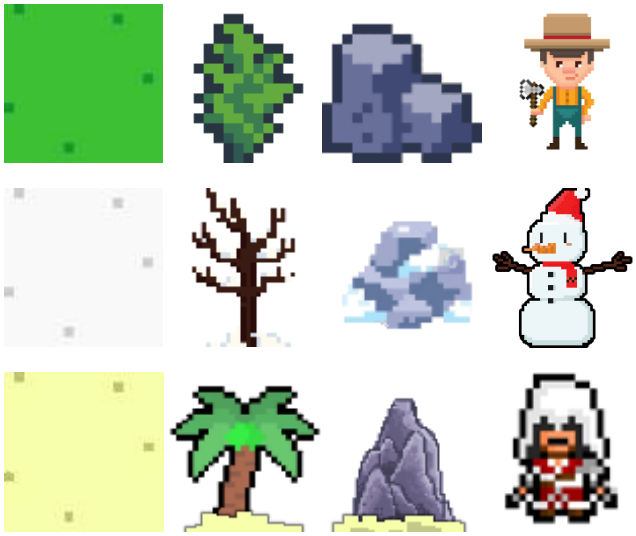
- Harta nivelului cu indicatorul trenului și resursele disponibile.
- Bară de timp și de progres pentru a ține evidența timpului rămas și a progresului realizat.
- Buton pentru colectarea resurselor și construirea șinelor.
- Buton pentru a se apăra împotriva inamicilor sau a le ataca.

Interacțiuni:

- Jucătorul poate interacționa cu resursele de pe hartă pentru a le colecta.
- Poate construi șine la un workbench.
- Poate ataca sau se poate apăra împotriva inamicilor prin intermediul unei interfețe intuitive.

Controale:

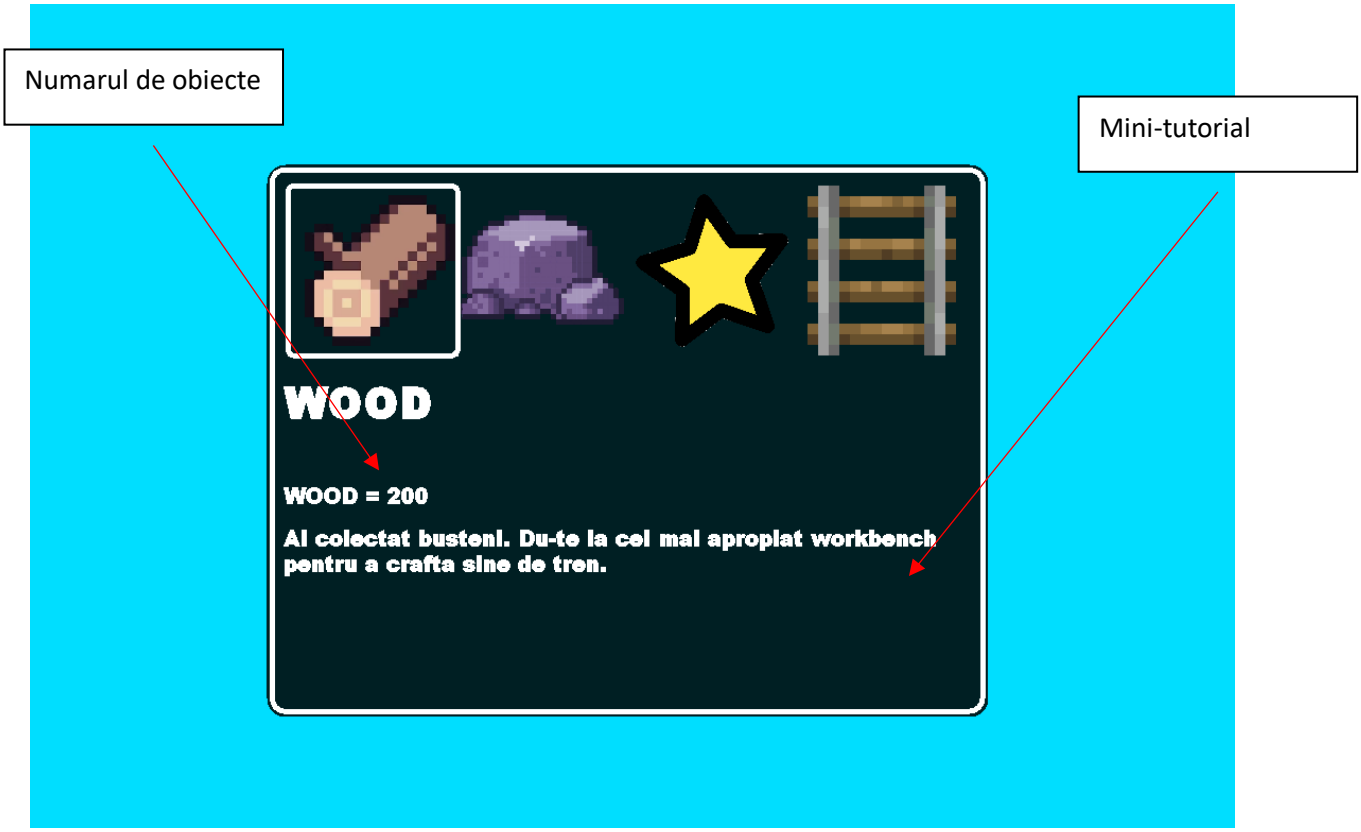




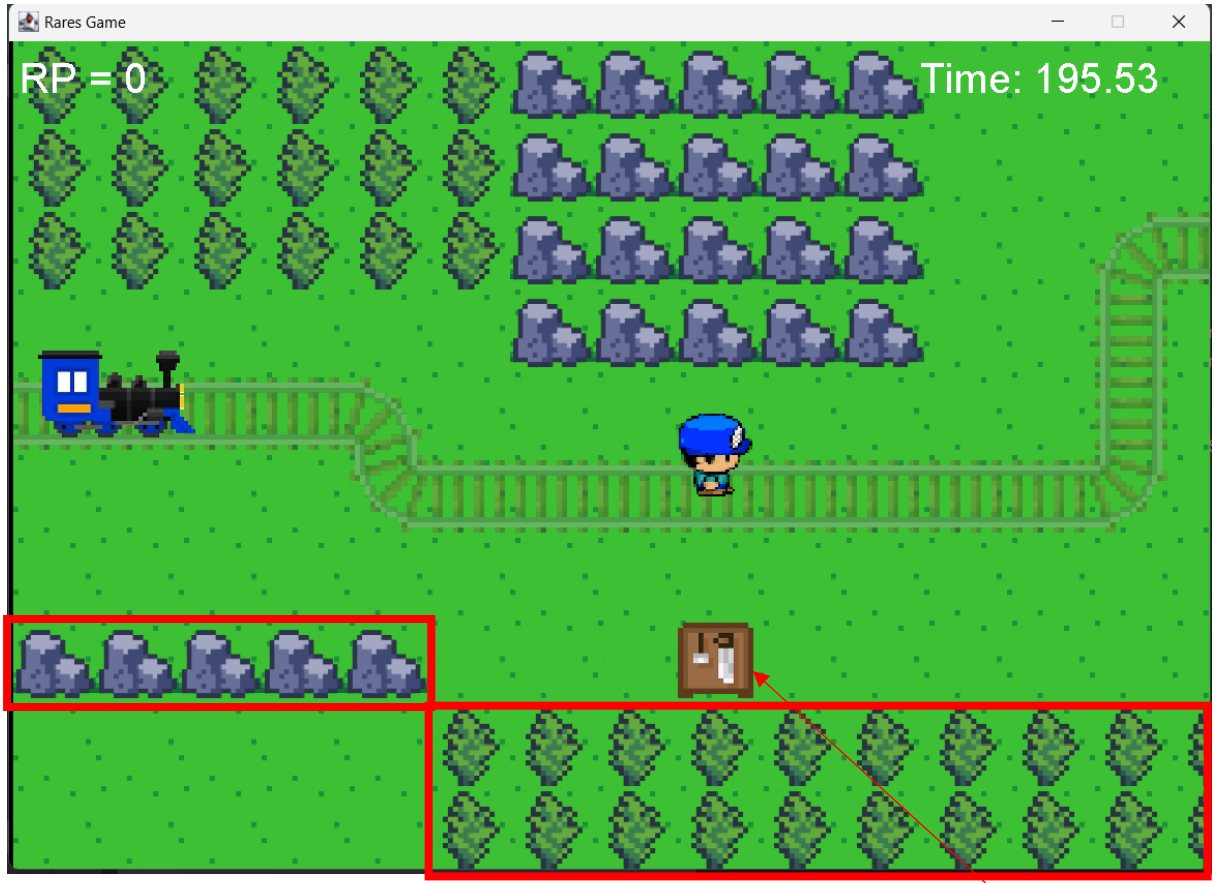
tiles

Erald ~ protagonistul jocului

Trenul

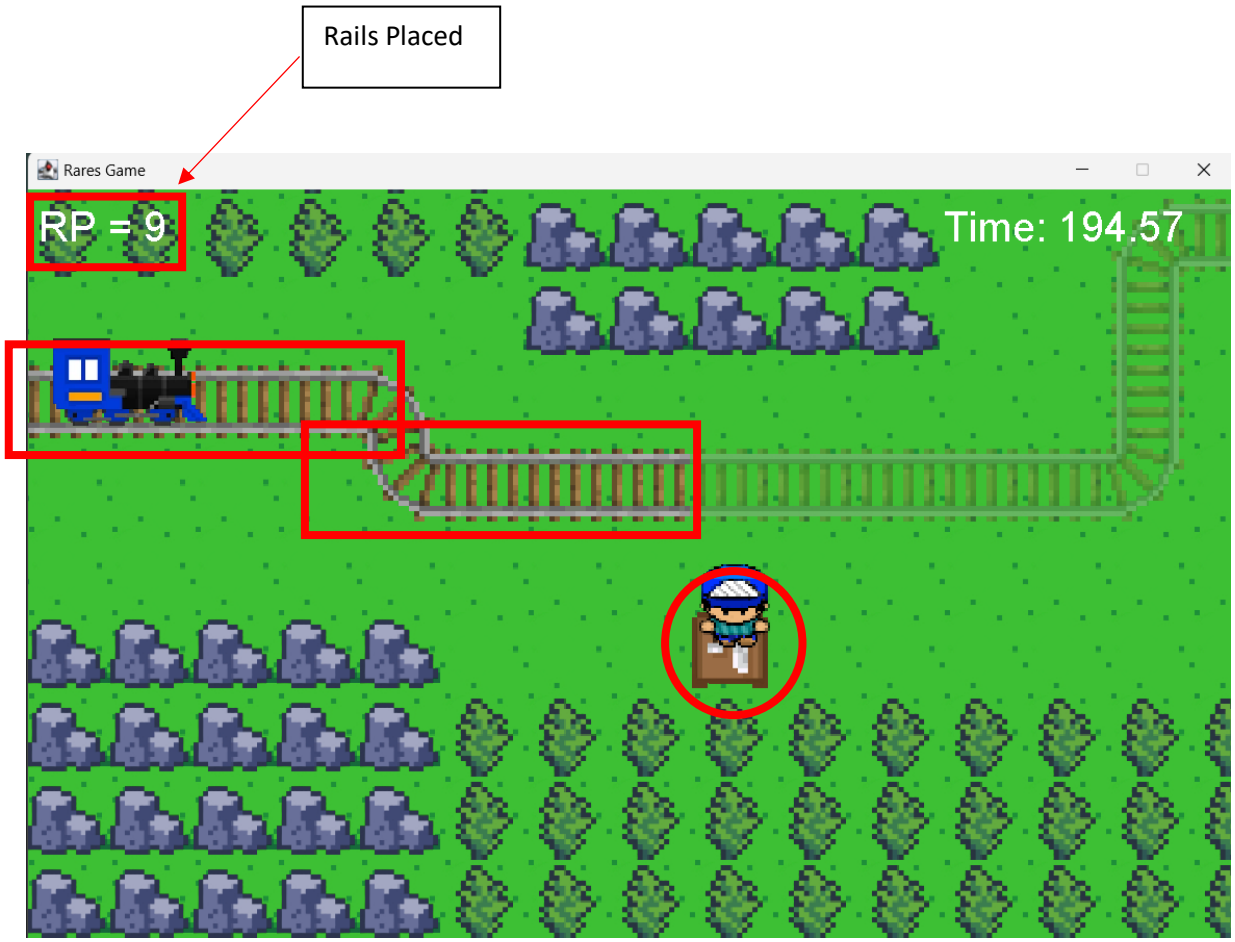


Inventar



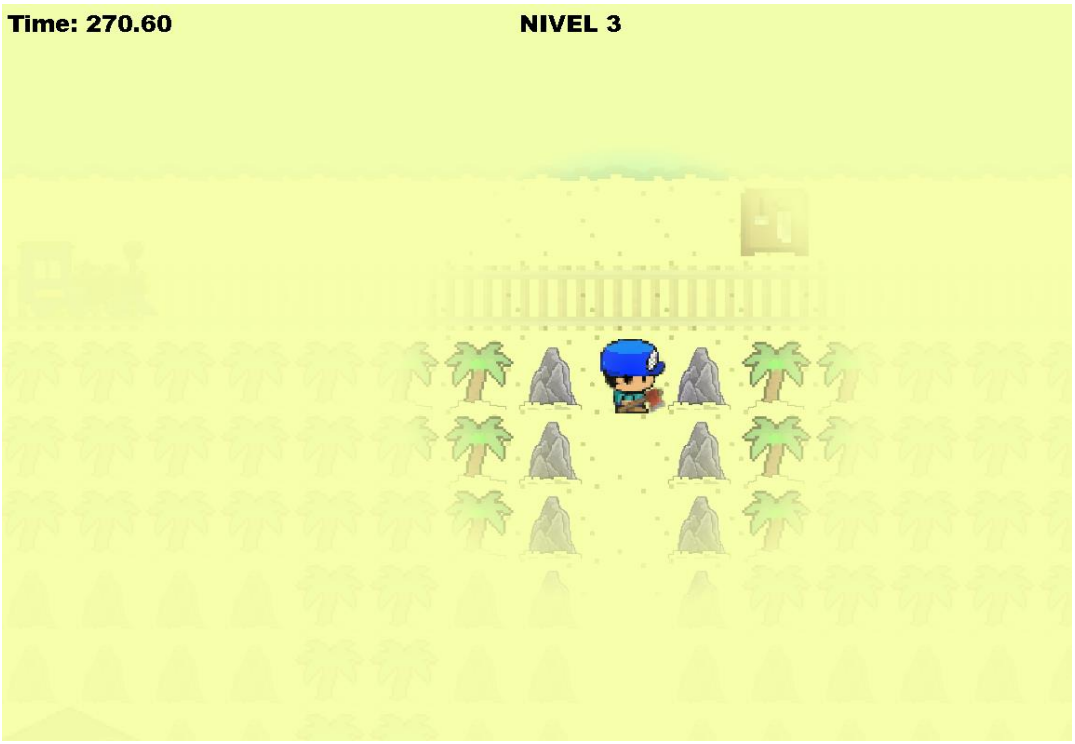
workbench

Unii copaci sau bucăți de piatră pot fi sparte. Acestea vor genera materiale. Odată colectate materialele, pot fi duse spre workbench unde se vor crafta șine de tren pe care va avansa trenul.


Rails Placed
Rares Game
RP = 9
Time: 194.57


Time: 31.60

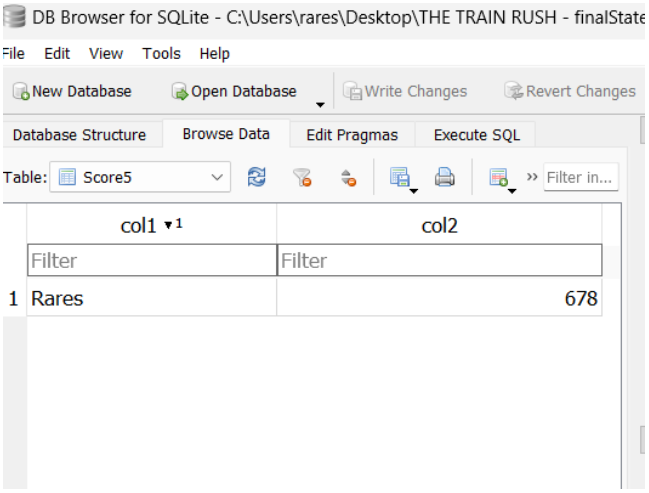
Daca pe harta, gasesti o stea, o colectezi si o duci la npc, acesta iti va oferi un rail.



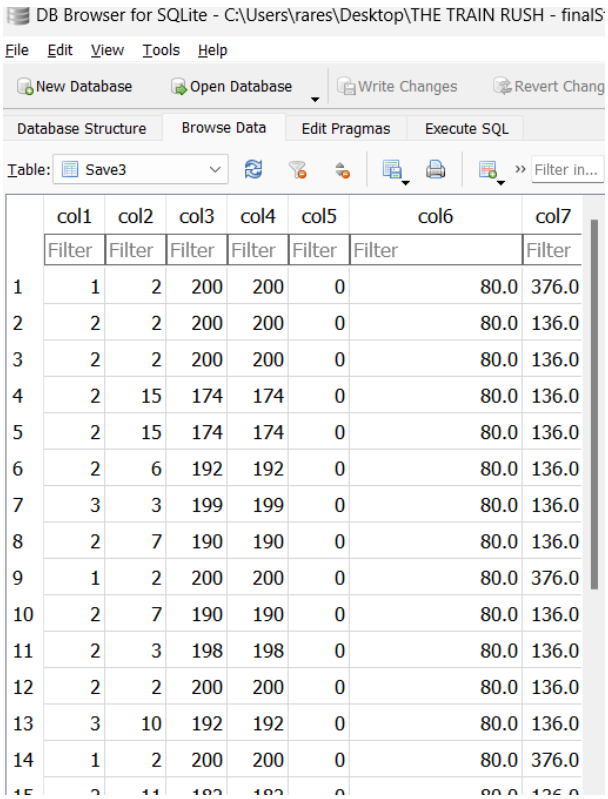
Pe parcursul nivelelor 2 si 3, pot incepe furtuni de zapada/nisip care ingreuneaza vederea

Baze de date

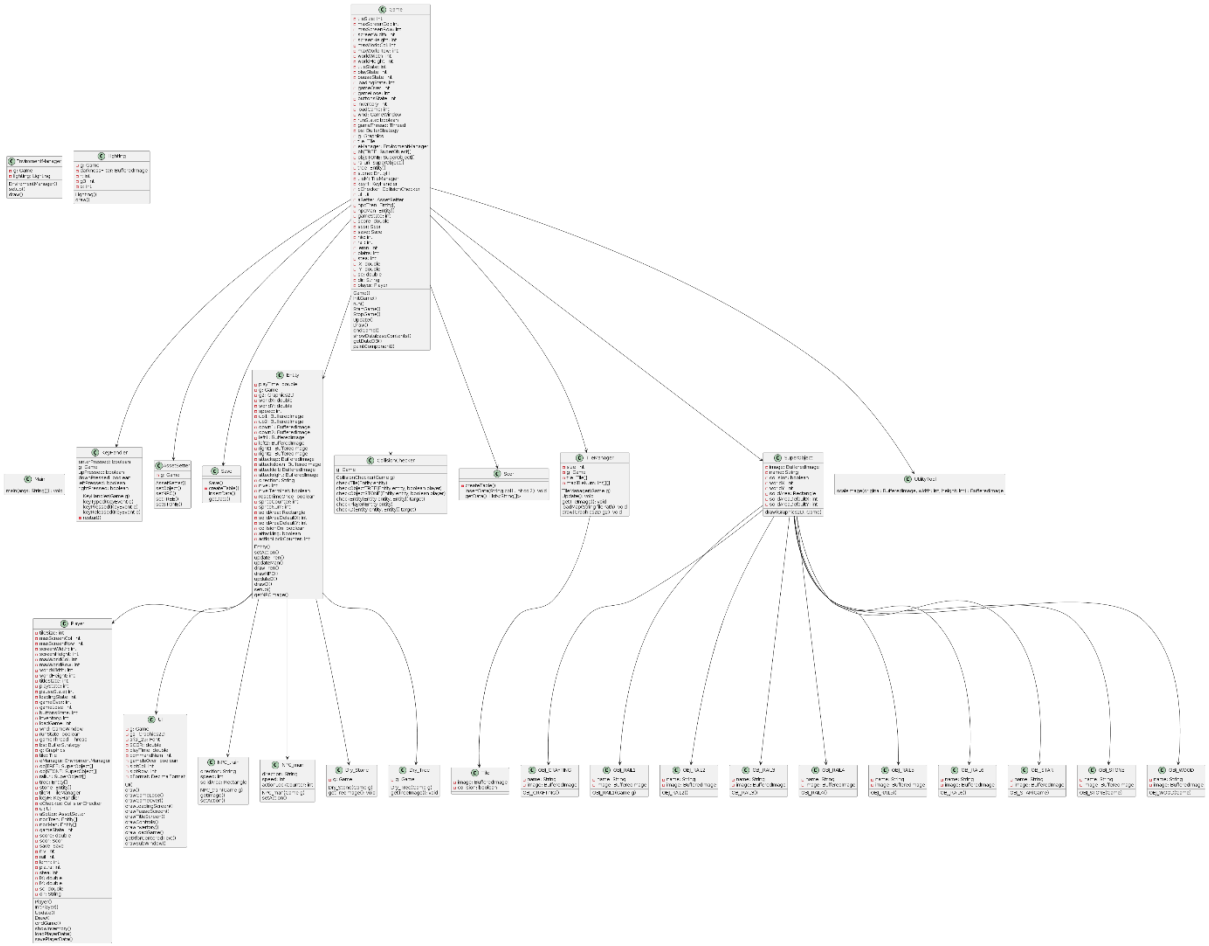
Obiectivul jocului este de a termina cele 3 nivele într-un timp cât mai scurt. La finalul jocului, jucătorul își poate scrie numele într-o casetă text. Numele acestuia și scorul său vor fi salvate într-o bază de date. De fiecare dată când cineva va termina jocul, baza de date va fi afișată pentru ca scorurile să poată fi comparate.



De fiecare dată când un jucător salvează starea jocului (apasând tasta „O” de la tastatură, nivelul, scorul său si alte date importante vor fi salvate în baza de date). La redeschiderea jocului, jucătorul are posibilitatea de a da „load game”, lucru ce îl va plasa la începutul nivelului la care se afla atunci când a salvat jocul.



	col1	col2	col3	col4	col5	col6	col7
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2	200	200	0	80.0	376.0
2	2	2	200	200	0	80.0	136.0
3	2	2	200	200	0	80.0	136.0
4	2	15	174	174	0	80.0	136.0
5	2	15	174	174	0	80.0	136.0
6	2	6	192	192	0	80.0	136.0
7	3	3	199	199	0	80.0	136.0
8	2	7	190	190	0	80.0	136.0
9	1	2	200	200	0	80.0	376.0
10	2	7	190	190	0	80.0	136.0
11	2	3	198	198	0	80.0	136.0
12	2	2	200	200	0	80.0	136.0
13	3	10	192	192	0	80.0	136.0
14	1	2	200	200	0	80.0	376.0



+atasez diagrama pdf in proiect

Game.java:

public Game() -> constructor; se formează spațiul de lucru, inițializează variabilele și setările inițiale ale jocului, adaugă ascultătorul de taste și configurează fereastra jocului.

void InitGame() -> apelarea funcțiilor pentru a seta obiectele, NPC-urile, copacii și pietrele din joc; pornește firul de execuție al jocului și setează starea jocului la titleState.

public void run() -> buclă de joc care rulează un număr specific de cadre pe secundă (frames per second - FPS); actualizează starea jocului și redesează ecranul.

public synchronized void StartGame() -> metoda este utilizată pentru a începe jocul, inițializând firul de execuție al jocului dacă acesta nu rulează deja.

public synchronized void StopGame() -> metoda este utilizată pentru a încheia jocul, oprind firul de execuție al jocului și așteptând să se termine.

private void Update() -> metoda este responsabilă pentru actualizarea stării jocului, inclusiv actualizarea jucătorului, mediului și NPC-urilor; de asemenea, scade scorul în timp.

private void Draw() -> metoda este responsabilă pentru desenarea scenei jocului folosind strategia de buffer.

public void endGame() -> metoda pentru încheierea jocului; afișează un câmp text pentru a introduce numele jucătorului și un buton pentru a salva scorul în baza de date.

public void showDatabaseContents() -> metoda afișează conținutul bazei de date într-un tabel nou într-o fereastră separată.

public void getDateDB() -> metoda obține datele salvate din baza de date și le afișează; de asemenea, actualizează variabilele interne ale jocului cu aceste date.

public void paintComponent(Graphics g) -> metoda pentru a desena componenta pe care este atașată; redesează elementele jocului pe baza stării actuale a jocului.

KeyHandler.java:

public KeyHandler(Game g) -> constructor; inițializează un nou handler de tastatură și primește o referință către obiectul jocului.

void keyTyped(KeyEvent e) -> metoda apelată când se tastează o literă

void keyPressed(KeyEvent e) -> metoda apelată când se apasă o tastă; actualizează variabilele corespunzătoare pentru tastatură.

void keyReleased(KeyEvent e) -> metoda apelată când se eliberează o tastă; actualizează variabilele corespunzătoare pentru tastatură.

Entity.java:

public Entity(Game g) -> constructor; inițializează o nouă entitate și primește o referință către obiectul jocului.

public void setAction() -> metoda pentru a seta acțiunea entității (nu este implementată în această versiune).

public void update() -> metoda responsabilă pentru actualizarea poziției și a stării entității în fiecare cadru de joc.

public void draw(Graphics2D g2) -> metoda pentru a desena entitatea pe ecran utilizând contextul grafic dat.

public void updateD() -> metoda alternativă pentru actualizarea entității (nu este implementată în această versiune).

public void drawD(Graphics2D g2) -> metoda alternativă pentru desenarea entității (nu este implementată în această versiune).

public BufferedImage setup(String imagePath, int width, int height) -> metoda pentru încărcarea și scalarea imaginilor entității.

NPC_train.java:

public NPC_train(Game g) -> constructor; inițializează un nou NPC de tip tren și primește o referință către obiectul jocului.

public void getImage() -> metoda pentru a încărca imaginile pentru animația trenului.

public void setAction() -> metoda pentru a seta acțiunile pe baza timpului de joc pentru NPC-ul tren.

AssetSetter.java:

public AssetSetter(Game g) -> constructor; inițializează un nou setter de asset-uri și primește o referință către obiectul jocului.

public void setObject() -> metoda pentru a seta obiectele în joc.

public void setNPC() -> metoda pentru a seta NPC-urile în joc.

public void setTree() -> metoda pentru a seta arborii în joc.

Player.java:

public Player(Game a, KeyHandler keyH) -> constructor; inițializează un nou jucător și primește referințe către obiectul jocului și handlerul de taste.

public void setDefaultValues() -> metoda pentru a seta valorile implicite ale jucătorului.

public void getPlayerImage() -> metoda pentru a încărca imaginile pentru animația jucătorului.

public void getPlayerAttackImage() -> metoda pentru a încărca imaginile pentru animația atacului jucătorului.

public void Update() -> metoda responsabilă pentru actualizarea poziției și a stării jucătorului în fiecare cadru de joc.

private void attaking() -> metoda pentru gestionarea animației atacului jucătorului.

private void interactNPC(int i) -> metoda pentru interacțiunea cu NPC-urile din joc (nu este implementată în această versiune).

private void interactD(int i) -> metoda pentru interacțiunea cu obiectele de tip D(obiecte care se pot distruge) din joc (nu este implementată în această versiune).

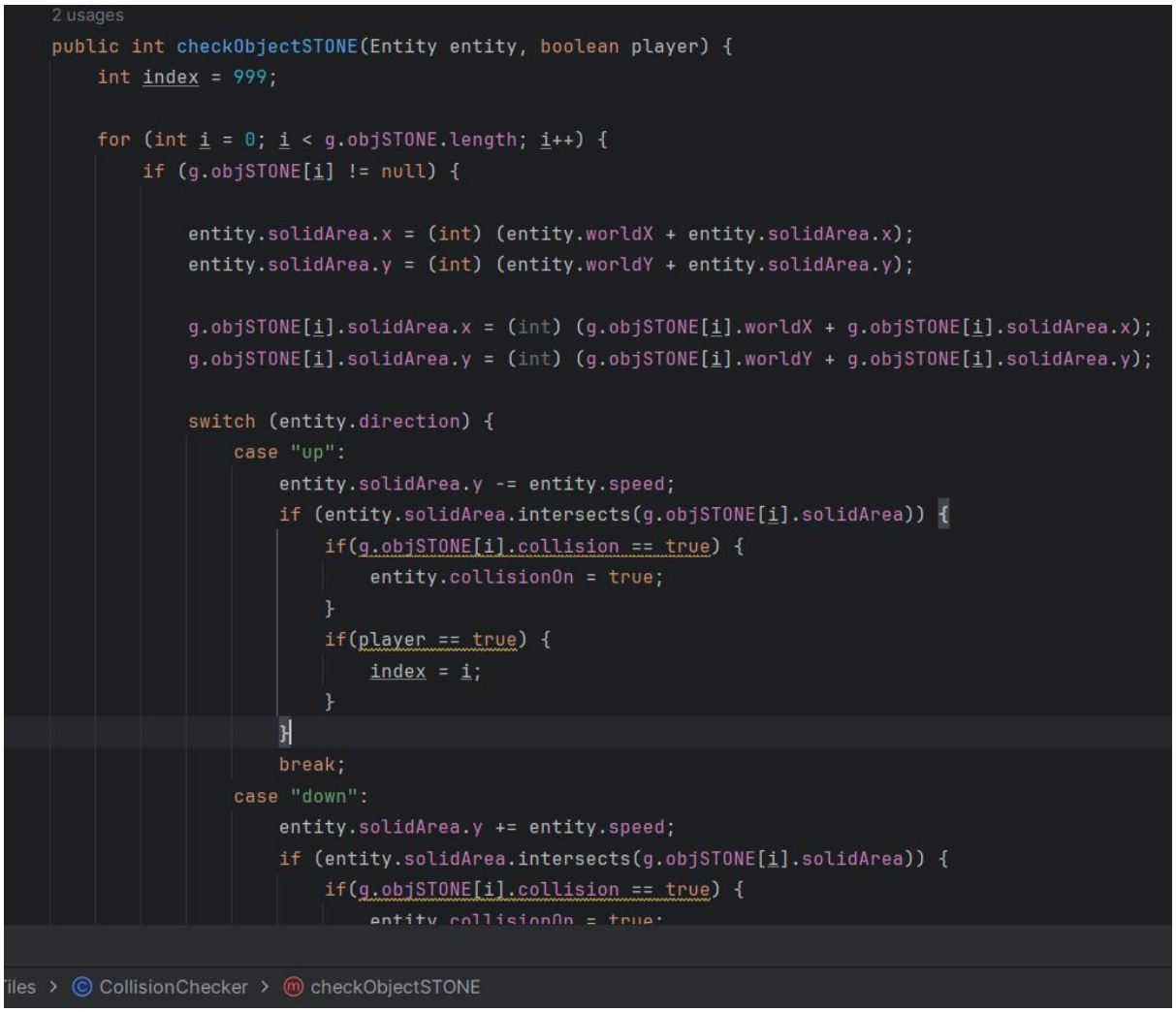
public void pickUpObject(int i) -> metoda pentru a lua un obiect din joc.

public void Draw(Graphics2D g2) -> metoda pentru a desena jucătorul pe ecran utilizând contextul grafic dat.

public void caleFerata() -> metoda pentru a plasa obiectele de tip "cale ferată" în joc în funcție de numărul de șine deținute de jucător.

Descriere algoritmi utilizati

1. **Tratarea coliziunii:**



```
2 usages
public int checkObjectSTONE(Entity entity, boolean player) {
    int index = 999;

    for (int i = 0; i < g.objSTONE.length; i++) {
        if (g.objSTONE[i] != null) {

            entity.solidArea.x = (int) (entity.worldX + entity.solidArea.x);
            entity.solidArea.y = (int) (entity.worldY + entity.solidArea.y);

            g.objSTONE[i].solidArea.x = (int) (g.objSTONE[i].worldX + g.objSTONE[i].solidArea.x);
            g.objSTONE[i].solidArea.y = (int) (g.objSTONE[i].worldY + g.objSTONE[i].solidArea.y);

            switch (entity.direction) {
                case "up":
                    entity.solidArea.y -= entity.speed;
                    if (entity.solidArea.intersects(g.objSTONE[i].solidArea)) {
                        if(g.objSTONE[i].collision == true) {
                            entity.collisionOn = true;
                        }
                        if(player == true) {
                            index = i;
                        }
                    }
                    break;
                case "down":
                    entity.solidArea.y += entity.speed;
                    if (entity.solidArea.intersects(g.objSTONE[i].solidArea)) {
                        if(g.objSTONE[i].collision == true) {
                            entity.collisionOn = true;

iles > CollisionChecker > checkObjectSTONE
```

Pentru tratarea coliziunii cu obiectele de tip „wood” sau „stone”, am folosit functii asemanatoare (exemplu cea de mai sus).

Funcția checkObjectStoneCollision verifică dacă playerul are coliziune cu orice obiect de tip "STONE". Parcurge lista obiectelor "STONE" și, în caz de coliziune, poate seta un indicator de coliziune și returna indexul obiectului "STONE" cu care s-a produs coliziunea sau o valoare specială (999) în caz contrar.

2. Inventar:

```
1 usage
public void drawInventory() {
    int frameX = (int)(g.tileSize*3);
    int frameY = g.tileSize*2;
    int frameWidth = (int)(g.tileSize*8.5);
    int frameHeight = (int)(g.tileSize*6.5);
    drawSubWIndow(frameX, frameY, frameWidth, frameHeight);

    final int slotXstart = frameX + 20;
    final int slotYstart = frameY + 20;
    int slotX = slotXstart;
    int slotY = slotYstart;

    for(int i=0; i<g.player.inventory.size(); i++) {
        g2.drawImage(g.player.inventory.get(i).image, slotX, slotY, width: 160, height: 160, observer: null);
        slotX+=2*g.tileSize;
    }

    int cursorX = slotXstart + (g.tileSize * slotCol);
    int cursorY = slotYstart + (g.tileSize * slotRow);
    int cursorWidth = g.tileSize;
    int cursorHeight = g.tileSize;

    g2.setColor(Color.white);
    g2.drawRoundRect(cursorX, cursorY, width: cursorWidth*2, height: cursorHeight*2, arcWidth: 10, arcHeight: 10);
    if(cursorX == 260) {
        g2.setColor(new Color( r: 0, g: 222, b: 255));
        g2.setFont(g2.getFont().deriveFont(Font.BOLD, size: 40F));
        String text = "WOOD";
```

Pentru inventar, a fost creat un nou stadiu de joc, numit „Inventar”. De fiecare dată când jucătorul apasă butonul „i” în timpul jocului, se va deschide inventarul (vezi pagina 9).

3. Deplasarea NPC-urilor:

Se genereaza un numar random de la 1 la 100. Daca numarul este intre 1 si 25, directia de deplasare a NPC-ului va fi in sus. Daca numarul este intre 25 si 50, directia de deplasare a NPC-ului va fi in jos etc.

```
3 usages
public void setAction() {
    actionLockCounter ++;
    if(actionLockCounter == 120) {
        Random random = new Random();
        int i = random.nextInt( bound: 100) + 1;
        if (i <= 25) {
            direction = "up";
        }
        if (i > 25 && i <= 50) {
            direction = "down";
        }
        if (i > 50 && i <= 75) {
            direction = "left";
        }
        if (i > 75 && i <= 100) {
            direction = "right";
        }
        actionLockCounter = 0;
    }
}
}
```

Sabloane de proiectare

Strategy: În metoda pickUpObjectTREE și pickUpObjectSTONE, exista o structură switch-case care gestionează diferitele acțiuni pe baza tipului de obiect interacționat. Acest mod de a trata acțiunile pe baza tipului de obiect ar putea fi văzut ca o formă simplă a sablonului Strategy, unde fiecare strategie este reprezentată de un caz în switch.

State: Există multiple blocuri if care verifică g.gameState pentru a determina comportamentul tastelor în funcție de starea jocului (g.titleState, g.buttonsState, g.loadingState etc.).

Singleton: Clasa Game este adesea folosită ca un singleton în aplicația ta de jocuri. Singleton-ul este o clasă care poate avea doar o singură instanță și oferă un punct global de acces la acea instanță. În cazul tău, constructorul clasei Game este privat și instanța este obținută folosind metoda statică StartGame() sau StopGame().

Observer: Într-un joc, entitățile pot necesita actualizări în funcție de starea jucătorului sau de evenimente globale din joc. De exemplu, entitățile ar putea reacționa la schimbările de nivel ale jucătorului (nivel).

Template: Metodele updateTren și updateMan, sau UpdateLvl1, UpdateLvl2 și UpdateLvl3, urmează un șablon similar de logică de actualizare, ceea ce sugerează că un Template Method ar putea fi folosit pentru a defini structura comună, lăsând subclasele să implementeze pașii specifici.

6. Resursele biografice utilizate bibliografie:

Tiles:

- https://stock.adobe.com/ro/search/images?k=2d+tiles&asset_id=586406211

- https://minecraft.fandom.com/wiki/Rail

- https://superssset.live/product_details/50257977.html

- https://www.shutterstock.com/image-vector/pixel-christmas-toy-train-games-web-764402140

- https://opengameart.org/content/rpg-character

- https://www.deviantart.com/suemann28/art/Desert-Tree-910747425

- https://www.shutterstock.com/image-vector/snowy-rocks-set-pixel-art-style-2256033721

- https://ninjikin.itch.io/grass