Intergalactical adventure

Diaconu Rareș-George grupa 1207A

-Proiectarea contextului:

Intergalactical adventure se dorește a fi un joc de luptă combinat cu strategie unde jucătorul trebuie să gasească și să invingă inamici de diferite dificultăți. Tematica jocului este lupta împotriva oricărui adversar care poate apărea între protagonist și cele trei comori.

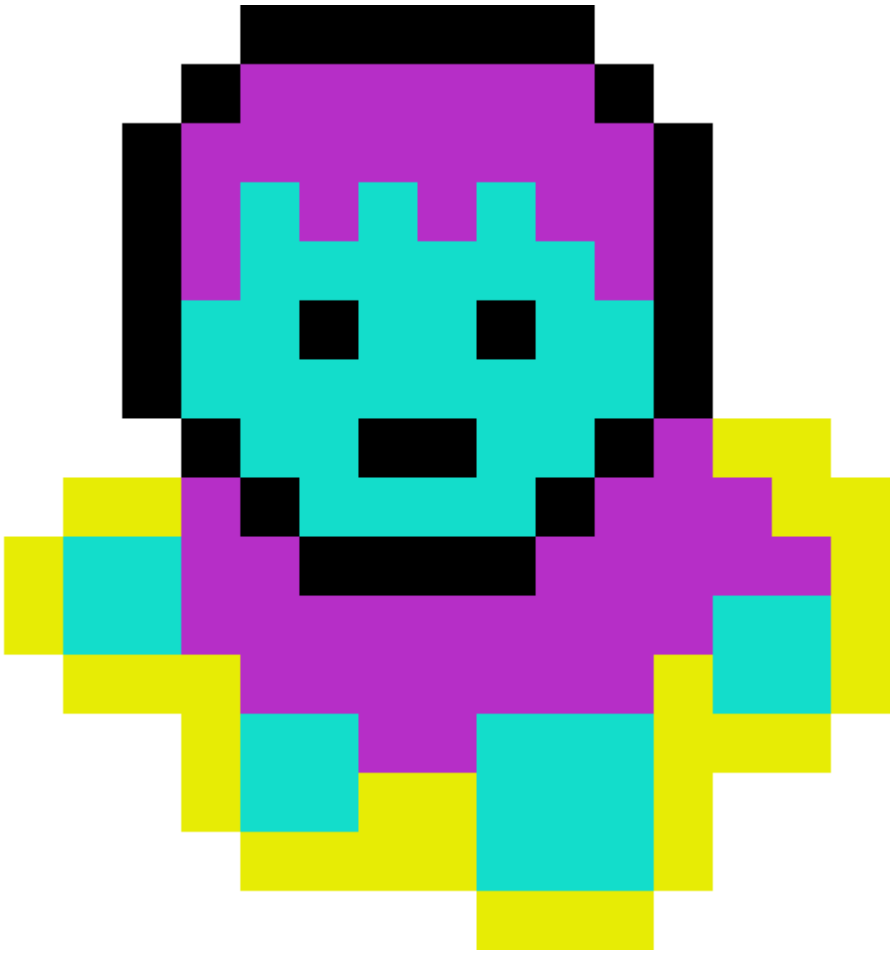
-Pe o planetă îndepărtată numită Tambola, era civilizație nouă ce nu era cunoscută altora din univers. Locuitorii erau necunoscuți altor specii, dar respectau legile naturii, aveau multe bogății pe care le prezervau cu strictețe și nimeni nu putea să le strice fericirea. Cei care trăiesc aici pot rezista pentru eternitate dacă au trei obiecte prețioase ce le oferă tot ce au nevoie în fiecare zi. Acestea sunt: un elixir magic creat de primii tambolieni care au venit pe această planetă și care creează aerul pe care îl respiră locuitorii și care îi menține sănătoși. Apoi este spada magică numită tambalala care le poate arăta viitorul și care îi menține în siguranță de alți atacatori care le-ar dori bogățiile. În final, frunza vrăjită care le oferă sursa de hrană tambolienilor care au doar o rezervă limitată dacă nu o folosesc. Liderul lor este Vagadun, singurul războinic de pe această planetă care a venit atunci când tambalala a fost furată pentru scurt timp și mulți inamicii au năvălit asupra Tambolei. Acesta a reușit să izgonească orice atacator și astfel locuitorii tambolei l-au făcut conducător. Vagadun a adus cu el multe inovații ce au îmbunătățit calitatea vieții de pe Tambola, iar tambolienii îl venerau și aveau încredere oarbă că nimic rău nu se poate întâmpla. Așa că atunci când nu se gândeau, au fost atacați de către niște rivali vechi ce au reușit să treacă de spada magică și care erau mult mai puternici si mai numeroși. Inamicii au pus mâna pe cele trei lucruri esențiale pentru tambolieni ce vor dispărea dacă nu reușesc să le recupereze. Vagadun pleacă în călătoria vieții sale pentru a-și salva poporul și pentru a recupera elixirul, spada si frunza. El trebuie să treacă de mai mulți monștrii sacrii ce nu au fost niciodată înfrânți ce păzesc obiectele și în cât mai scurt timp posibil deoarece cu fiecare zi care trece, foarte mulți tambolieni mor. Oare va reuși Vagadun să recupereze lucrurile și să salveze planeta Tambola de la extincție?

-Proiectarea sistemului:

-Jocul începe cu Vagadun încercând să gasească monstrul sacru pe nume Bouleng. Acesta are unii prieteni care îi vor face viața grea lui Vagadun. Jucătorul va avea de înfruntat inamici mai mici și mai slabi care îi vor scădea viața. Vagadun va începe cu 100% viață, dar nu se va regenera, ci doar va scădea așa că jucătorul trebuie să aibă mare grijă să nu piardă din hp în primele lupte. Însă o lovitură în cap de la ultimiul monstru îl va omorî pe protagonist oricât de multă viață mai are până în acel moment. Ca să ajungă la boss-ul din joc, jucătorul va avea de trecut și de niște vrăjitori care îl vor ajuta cu obiecte și cu găsirea elixirului. Pentru a trece la următoarele niveluri monstrul trebuie să fie infrânt și comoara să fie recuperate. La final, după cele trei niveluri sunt terminate și comorile recuperate jocul este câștigat și planeta Tombola este salvată, iar Vagadun devine o legendă.

-Proiectarea conținutului:

1. Vagadun

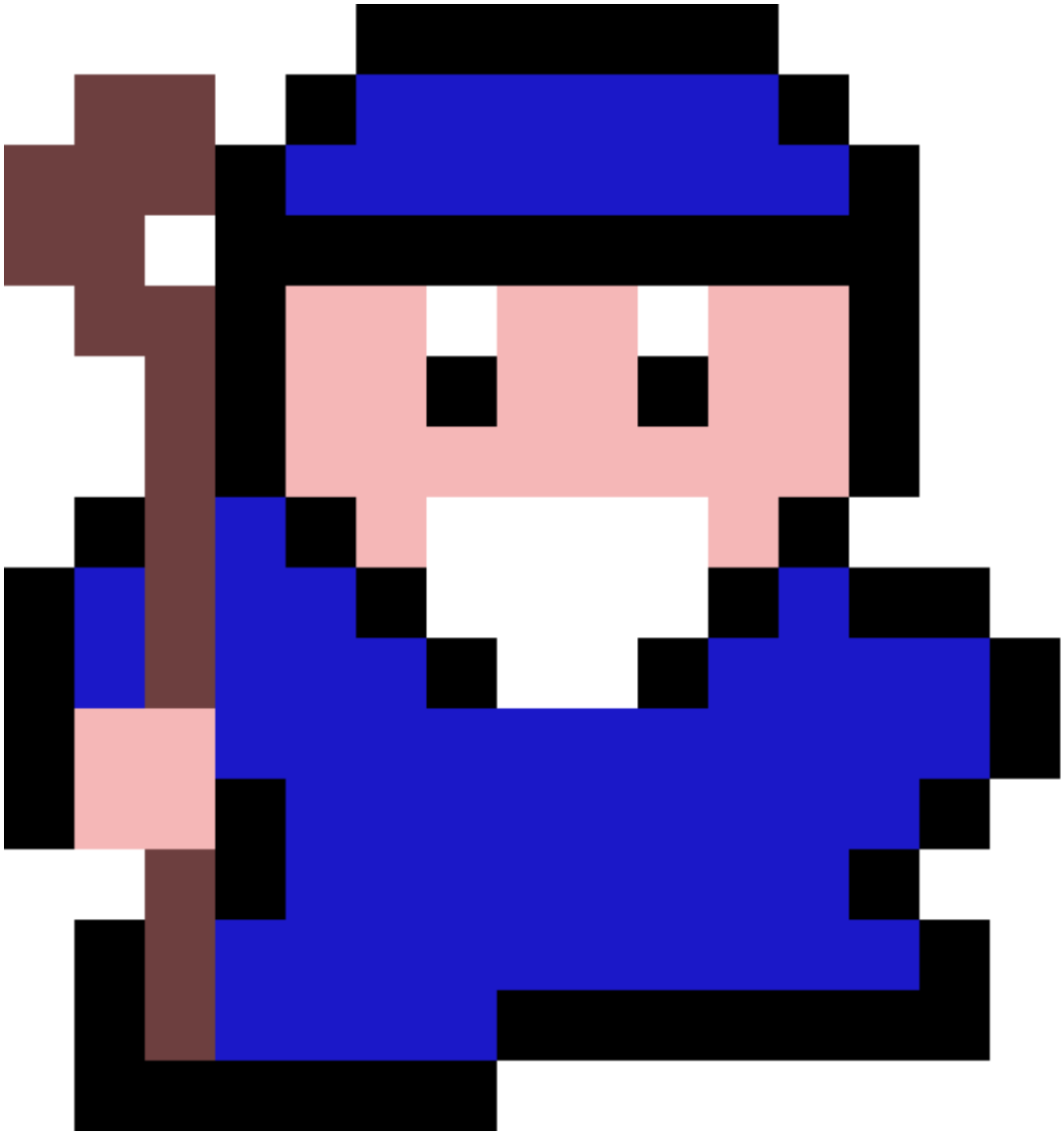


Vagadun este conducătorul planetei Tombola, un om bun la suflet, respectuos și un incredibil luptător care a reușit să salveze planeta de mai multe ori. Trecutul său este unul învăluit în mister deoarece el este singurul de pe Tombola care nu este născut acolo, venind de pe Vadun în invazia de acum mulți. Vagadun i-a apărat atunci pe tombolieni de compatrioții săi și așa a devenit conducător. El tot timpul a fost atent la inamicii pe care i-ar putea avea Tombola și i-a ținut departe. În același timp, a fost un conducător bun și iubitor față de locuitorii din Tombola. Acum el pleacă în cea mai grea și lungă aventură din viața sa.

2. Neoi

Neoi este un vrăjitor pe care Vagadun îl va întâlni pe drum spre elixir și îl va ghida pentru a-l găsi și a trece mai departe. El este un maestru în magia neagră pe care a perfecționat-o cu timpul și doar el o poate stăpâni, dar Neoi este pașnic și nu atacă pe nimeni.

3. Meloi

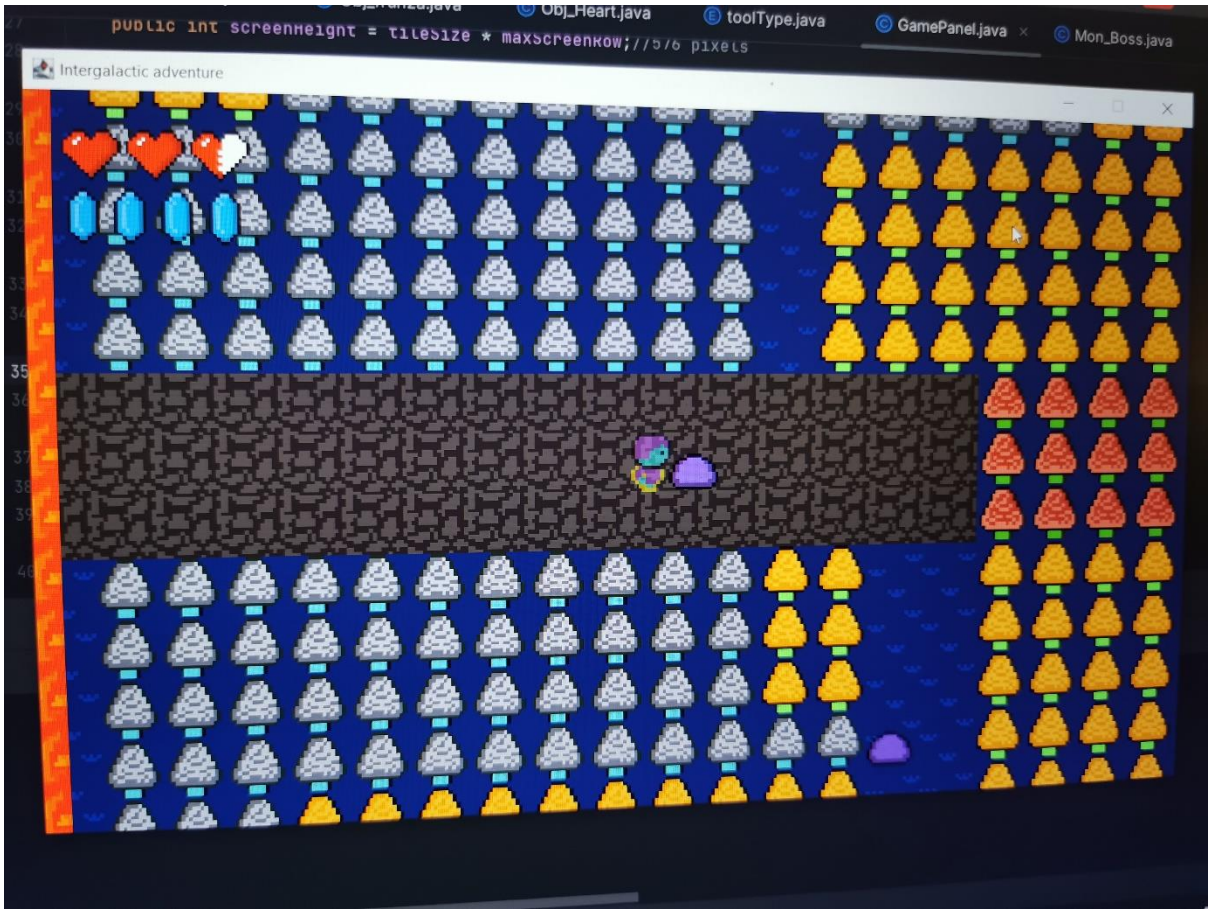


Meloi este fratele mai mare al lui Neoi și va fi întâlnit în nivelul doi. Acesta are foarte multe obiecte de vânzare pentru a reuși de a trece de toți monștrii și de a termina aventura. Trebuie omorâți monștrii pentru a căpăta cât mai mulți bănuți și a putea cumpăra mai multe lucruri de la Meloi.

-Proiectarea nivelurilor:

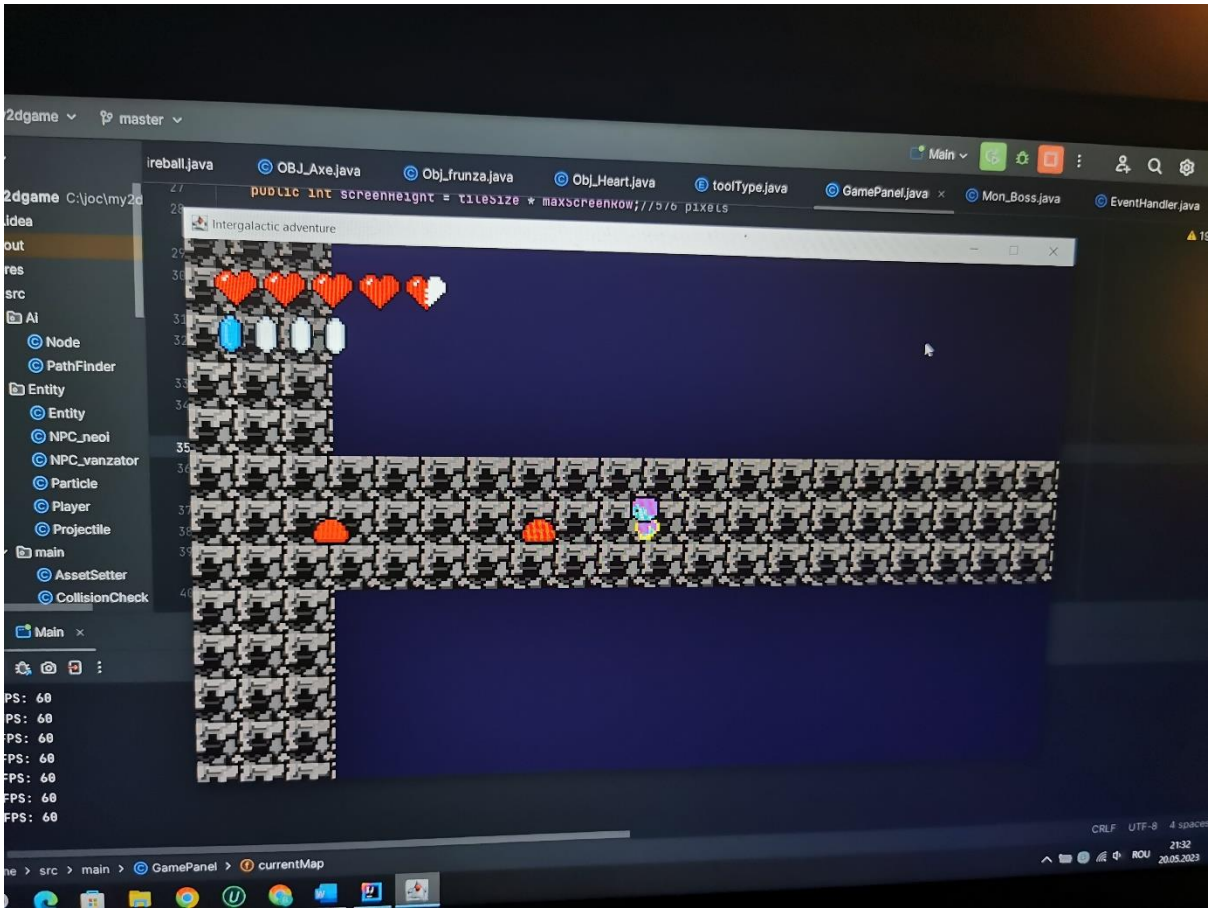
-primul nivel este și cel ușor pentru a da oportunitatea jucătorului de a se obișnui cu mapa, cu abilitățile protagonistului și pentru a nu pierde prea multă viață de care va avea nevoie mai târziu în luptele mai grele. Va fi un drum destul de întortocheat ce trebuie parcurs pentru a ajunge la al doilea nivel. Inamicii mai slabi vor veni odată ce ne apropiem de Bouleng și desigur vom da și de Neoi care ne va ajuta în găsirea elixirului.

-harta primului nivel:

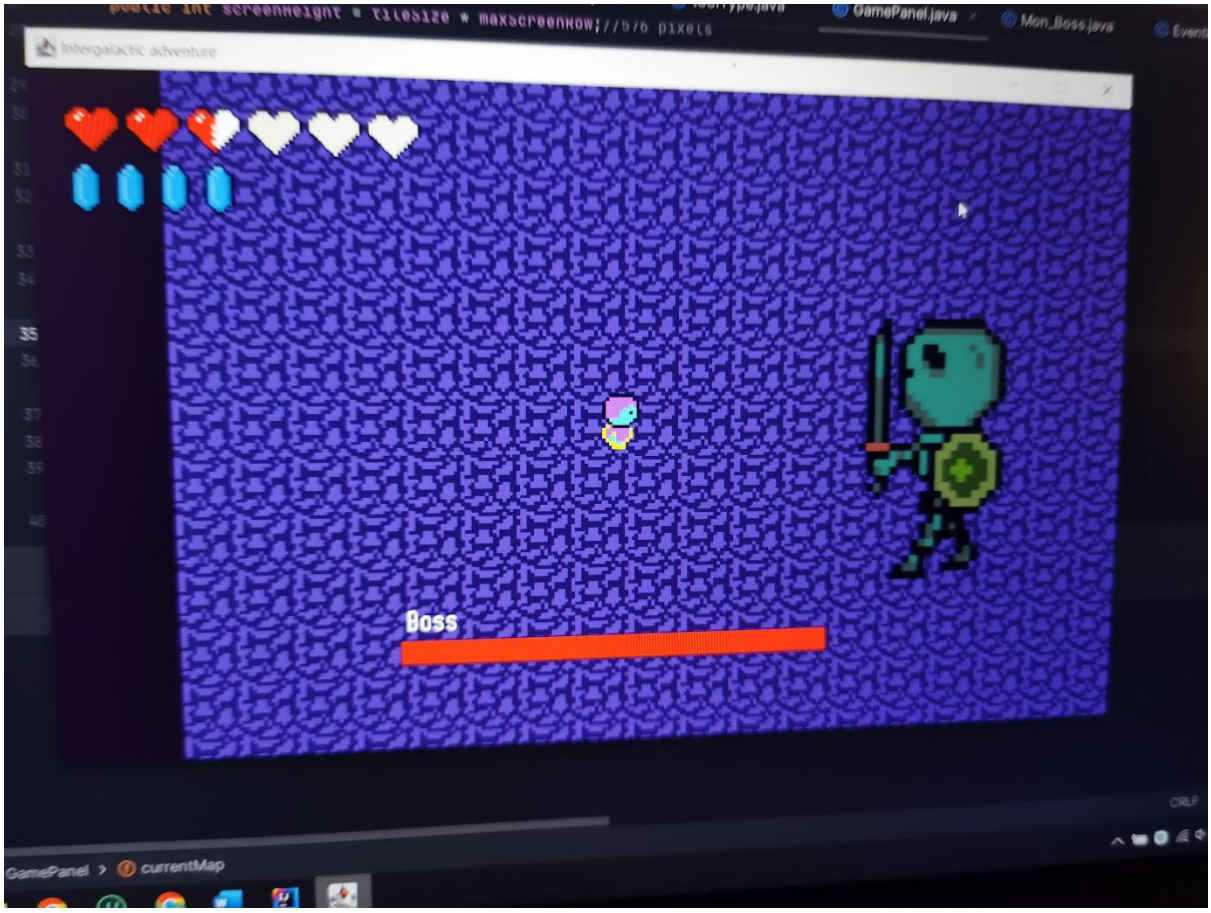


-al doilea nivel este cel intermediar, un nivel ce îi va testa jucătorului întelegerea sa cu Vagadun ce va avea de înfruntat foarte mulți inamici pe parcurs ce vor fi mai puternici .Drumul va fi mai scurt, dar va părea mai lung din cauza valurilor de inamici și a magicianului Meloi care îi va vinde poțiuni și diferite obiecte pentru a ieși învingător din această aventură.

- o poză din nivelul al doilea:

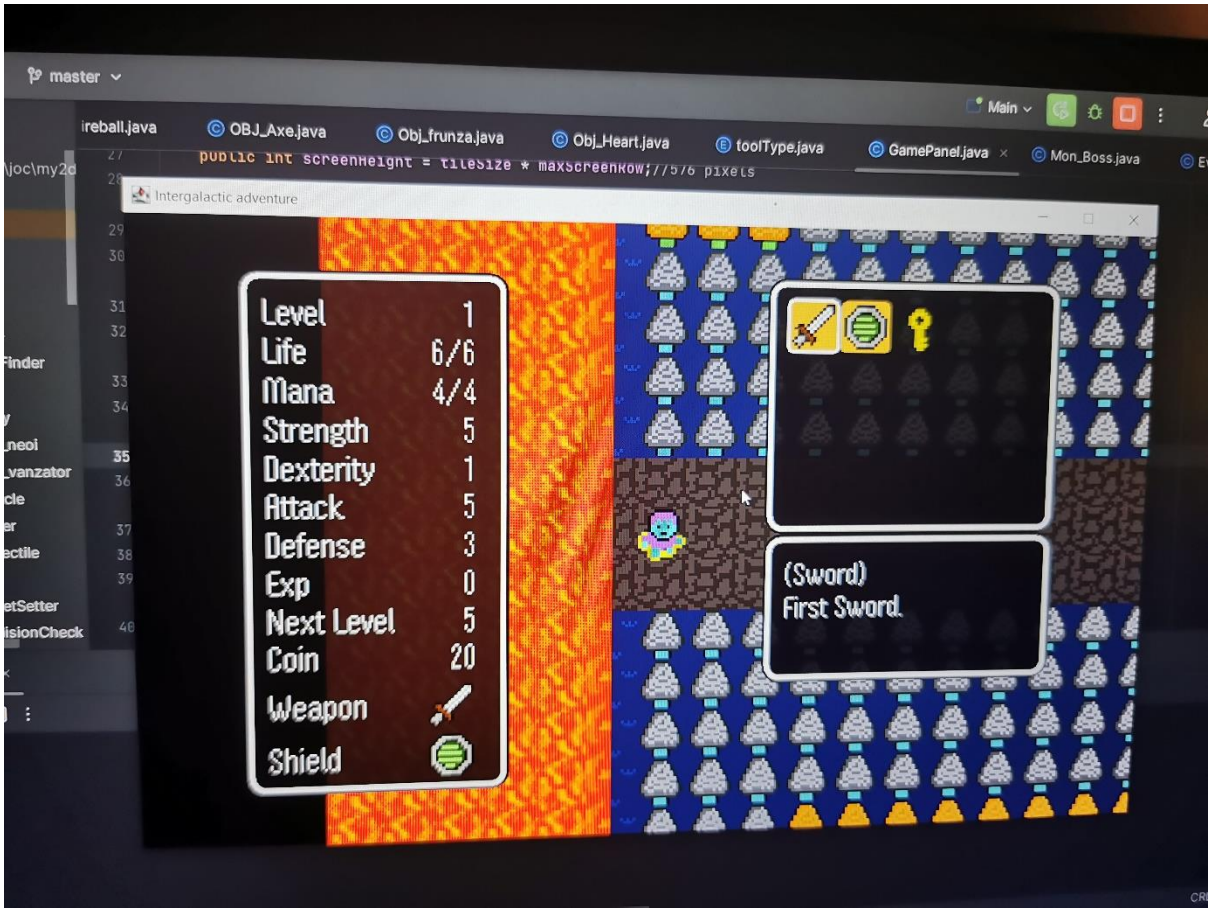


-al treilea nivel este și ultimul unde se poate ajunge cu foarte puțină viață ce îi va face viața jucătorului foarte grea. În acest nivel final nu mai există un vrăjitor. Vom ajunge la Babalong, boss-ul final ce va fi foarte greu de învins mai ales din cauza vieții scăzute. Drumul va fi cel mai scurt deoarece nu va fi foarte mult timp să meargă Vagadun, ci să lupte neîncetat.

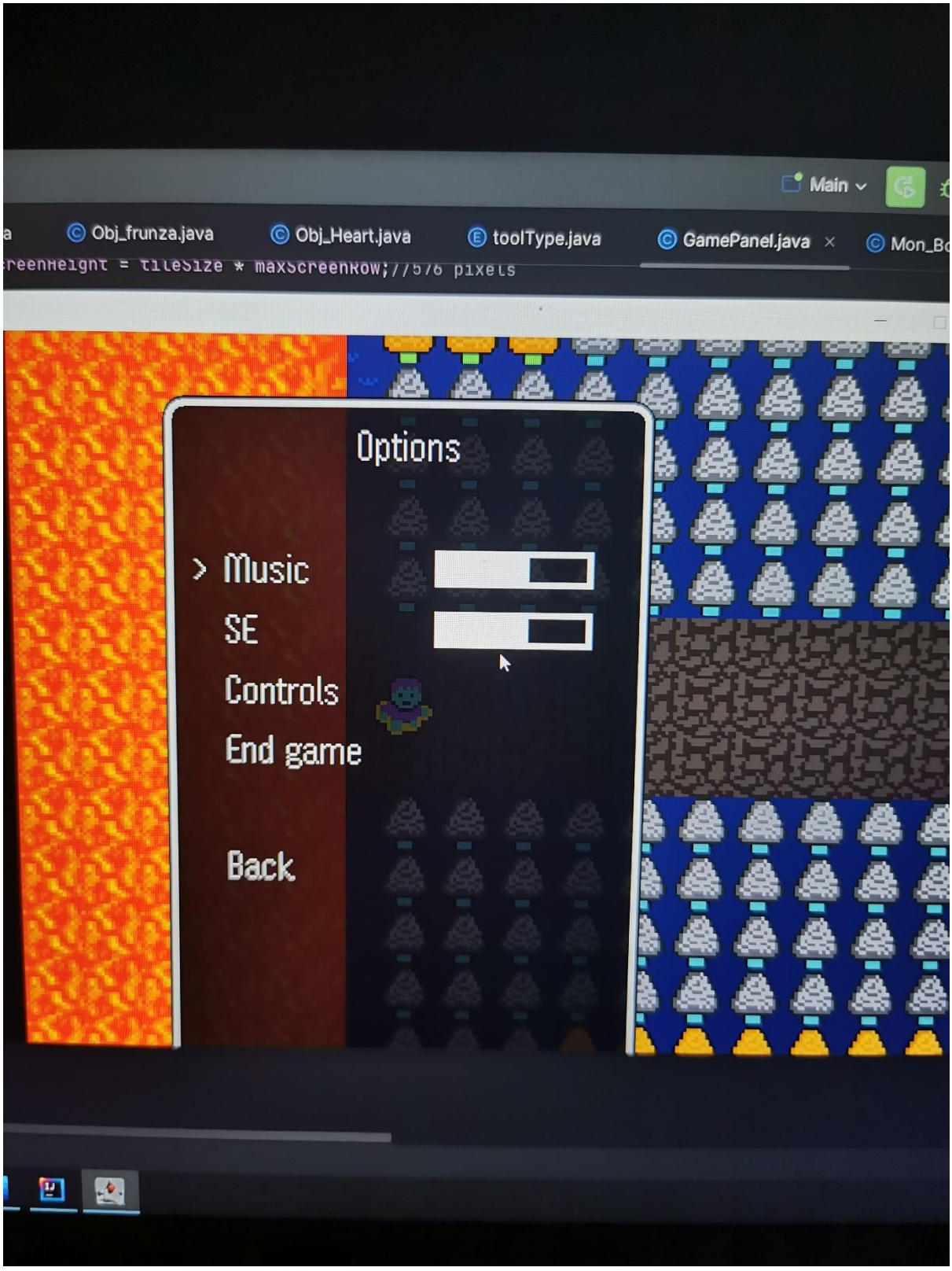


-Proiectarea interfeței:

-la imaginea cu nivelul actual, utilizatorul va putea să apese un buton din partea stânga sus ca să deschidă meniul cu statusul actual:



-interfața atunci când utilizatorul apasă pe butonul de opțiuni:



- **Diagramă proiect**:
- este atașată în arhivă.

- **Descriere detaliată**:

- **Main**: După cum se poate observa din diagrama prezentată mai sus, această clasă creează o instanță de tipul GamePanel, iar mai apoi pornește fluxul activităților prin apelul metodei StartGameThread() din GamePanel.

- **GamePanel**: Rolul acestei clase este de a face inițializările (de exemplu, de a crea instanțe pentru jucători/inamici, pentru a inițializa harta, etc.), dar și de a menține interfața grafică la zi cu ce se întâmplă în joc. Această clasă implementează interfața Runnable pentru a avea comportamentul unui fir de execuție (thread). În momentul apelului metodei StartGame() se instanțiază un obiect de tip Thread pe baza instanței curente a clasei Game. Orice obiect de tip Thread trebuie să implementeze metoda run() care este apelată atunci când firul de execuție este pornit (start()). Această metodă run() inițializează jocul , iar mai apoi controlează numărul de cadre pe secundă printr-o buclă while și "pregătește" noua scenă (Update()) pe care o desenează pe interfața grafică (paintComponent()). Metoda Update() actualizează starea jocului (de exemplu: modifică poziția jucătorilor pe baza tastelor apăsate, schimbă poziția inamicilor folosind chiar tehnici de inteligență artificială, creează diferite elemente (dale), etc.). Dacă ne referim la un joc 2D care are harta pătrată, aceasta este împărțită în celule de aceeași dimensiune (ca o tabelă de șah), iar aceste celule sunt tile-uri care pot avea texturi diferite, pot reprezenta diverse obiecte (de exemplu, proiectilele unui tanc, aceasta poate fi o tuilă cu un rest de elemente de pe hartă dar care are fundalul transparent). Metoda paintComponent() va desena pe interfața grafică modificările făcute de metoda Update(). Interfața grafică este un canvas, făcând o analogie cu realitatea, poate fi considerată o pânză pentru desen, pe care sunt desenate diverse obiecte. Se folosesc aici clasele gameState și toolType de tip enum. Tot aici se află și metodele retry() și restart() care se activează atunci când player-ul moare, iar utilizatorul are șansa de a reîncerca sau a reveni la începutul jocului.

- **Entity**: Clasa aceasta este cea pe care se bazează orice entitate din joc. În interiorul ei, se păstrează informații despre poziția și dimensiunea entității în lumea jocului, cum ar fi coordonatele x și y ale poziției și lățimea și înălțimea entității. Mai mult, aici sunt tratate coliziunile între obiecte, player, tile-uri și orice este în joc în metoda update() și checkColission(). Conține și texturile folosite împreună cu sprite-urile ce sunt utilizate și sunt desenate pe ecran prin metoda draw(). Include și metode care descriu activitățile player-ului și a altor entități din joc (speak(), damagePlayer(), checkDrop(), damageReaction(), dyingAnimation(), attacking(), checkAttackOrNot(), dropItem(), checkStopChasingOrNot(), getOppositeDirection(),checkStartChasingOrNot(),getRandomDirection(), setKnockBack(), moveTowardPlayer(), searchPath()). Tot în această clasă se păstrează și informații despre resursele jucătorului precum sabie sau mână ce pot fi utilizate în joc în metoda setAction().Mai mult, sunt folosite și unele clase ajutătoare precum getCoalCol(), getCoalRow(), getTileDistance(), getCenterY(), getCenterX(), getXDistance(), getYDistance(), getScreenX(), getScreenY(), inCamera(). Această clasă este părinte pentru majoritatea celorlalte entități precum Player, Projectile, Particle, NPC_neoi, Mon_redSlime, InteractiveTile, Mon_Boss, Mon_purpleSlime.

- **Player**: Aici este folosit șablonul de programare singleton. Această clasă implementează protagonistul jocului și toate abilitățile sale împreună cu toate obiectele sale. Abilități precum atacul împreună cu sprite-urile necesare în metoda getPlayerAttackImage() sau să ridice obiecte cu metoda pickUpObject(). Sprite-urile sunt utilizate în metoda getPlayerImage() și getGuardImage(), iar abilitățile inițiale sunt în metoda setDefaultValues() și setItems(). Prin metoda update() se modifică protagonistul, iar cu metoda draw() se desenează pe ecran acele modificări. Mai mult, tot aici sunt implementate și interacțiunile player-ului cu celelalte entități precum monștrii sau npc-urile prin intermediul metodelor interactNpc(), contactMonster() și damageMonster(). Tot în această clasă sunt implementate și interacțiunile cu obiectele cu ajutorul metodelor canObtainItem(), selectItem(), checkLevelUp() și searchItemInInventory(). Această clasă este moștenită din clasa Entity și avem și metodele care ajută la metoda retry(), acestea fiind setDefaultPositions() și restoreLifeAndMana().

- **KeyHandler**: Această clasă este cea care implementează interacțiunea utilizatorului cu jocul. Aici se implementează interfata KeyListener, cu cele 3 metode principale: keyPressed(), keyTyped() și keyReleased(). Metoda keyPressed() verifică starea actuală a jocului (gamestate) și apelează una dintre metodele: titleState(), playState(), pauseState(), dialogState(), characterState(), optionState() și exitState(), în funcție de starea jocului. Metoda keyReleased() verifică când utilizatorul nu mai apasă tasta și oprește acțiunea jucătorului de a se deplasa cel mai des. Această clasă gestionează, de asemenea, evenimente legate de tastatură precum tasta apăsată sau tasta eliberată, și acționează în consecință în funcție de starea curentă a jocului și de necesitățile acestuia. Metoda titleState() este una dintre metodele apelate în clasa care implementează interacțiunea utilizatorului cu jocul, în momentul în care starea jocului este setată pe modul "titlu" (titleState). Această metodă este responsabilă pentru afișarea ecranului de titlu al jocului și gestionarea interacțiunii utilizatorului în acest context.

În mod normal, metoda titleState() va afișa ecranul de titlu al jocului, care poate conține elemente precum logo-ul jocului, butoane pentru începerea jocului, setări sau opțiuni, și eventual un fundal grafic corespunzător. Utilizatorul poate interacționa cu aceste elemente folosind tastatura sau mouse-ul, iar metoda titleState() va gestiona evenimentele generate de aceste interacțiuni.

De exemplu, utilizatorul poate apăsa o anumită tastă pentru a începe jocul, iar metoda titleState() va detecta aceasta acțiune prin intermediul metodei keyPressed() din interfața KeyListener, și va schimba starea jocului la modul "joc" (playState), inițiind astfel începerea efectivă a jocului. Similar, utilizatorul poate apăsa alte taste pentru a accesa setările, a închide jocul sau a realiza alte acțiuni specifice.

Metoda titleState() poate conține, de asemenea, logica necesară pentru a gestiona animațiile, sunetele sau alte elemente grafice sau sonore specifice ecranului de titlu, și poate interacționa cu alte clase sau obiecte din joc pentru a implementa funcționalitățile specifice acestui modul al jocului.

Metoda playState() implementează toate tastele pe care le poate folosi utilizatorul și ce fac fiecare în parte. Aceasta poate include tastatură standard (cu diacritice) sau alte dispozitive de intrare, cum ar fi gamepad-uri sau mouse-ul.

De exemplu, dacă utilizatorul apasă tasta "W" (sau "În sus" în cazul tastaturii cu diacritice), metoda playState() poate actualiza poziția personajului principal pentru a-l deplasa în sus pe

ecran. Dacă utilizatorul apasă tasta "S" (sau "În jos"), metoda poate actualiza poziția personajului pentru a-l deplasa în jos, și tot așa.

De asemenea, metoda playState() poate gestiona tastele pentru acțiuni specifice, cum ar fi atacul sau săritul, sau tastele pentru interacțiunea cu obiecte din joc, cum ar fi colectarea de resurse sau deschiderea unor uși.

În funcție de logica specifică a jocului, metoda playState() poate implementa diferite acțiuni pentru fiecare tastă apăsată de utilizator. De exemplu, poate actualiza starea personajului, verifica coliziuni cu obiecte din joc, actualiza obiectivele nivelului, gestiona inventarul sau alte acțiuni specifice jocului în care este implementată.

- **TileManager**: Clasa aceasta implementează harta jocului și importă tile-urile ce vor fi folosite și coliziunile de la fiecare tile în parte. Metoda getTileImage() este cea care ajută la importarea tile-urilor ce vor fi folosite în metodele setup() și loadMap(). Metoda loadMap() este cea care asamblează harta așa cum trebuie să arate în joc, adică dispune tile-urile în pozițiile corecte și creează coliziuni între acestea. Metoda setup pregătește harta pentru a fi completată de metoda loadMap(). La final, metoda draw() doar desenează pe ecran harta jocului.

Metoda getTileImage() este o metodă care face parte din clasa care implementează harta jocului și are rolul de a returna imaginea asociată unui anumit tip de tile în funcție de codul sau identificatorul său. Această metodă primește ca parametru codul sau identificatorul tile-ului pentru care se dorește obținerea imaginii și caută în resursele jocului imaginea corespunzătoare acestui tile. Imaginile tile-urilor pot fi stocate într-un format specific, cum ar fi un array bidimensional sau o hartă de texturi, și metoda getTileImage() are rolul de a accesa aceste resurse și de a returna imaginea corespunzătoare tile-ului cerut. Această imagine poate fi ulterior utilizată în metodele de desenare (cum ar fi metoda draw() menționată anterior) pentru a afișa tile-ul pe ecran în poziția corectă în cadrul hărții jocului.

Metoda setup() este o metodă care face parte din clasa care implementează harta jocului și are rolul de a pregăti harta pentru a fi completată de metoda loadMap(). Această metodă poate implica inițializarea resurselor necesare pentru harta jocului, cum ar fi încărcarea imaginilor tile-urilor, definirea coliziunilor, stabilirea dimensiunilor hărții și orice alte setări necesare înainte de a putea fi încărcată efectiv harta jocului.

De exemplu, în această metodă se pot încărca imaginile tile-urilor folosite în joc, se pot defini coliziunile dintre diferitele tipuri de tile-uri (cum ar fi tile-uri care pot fi trecute sau tile-uri care reprezintă obstacole), se pot stabili dimensiunile hărții jocului și alte setări necesare pentru a pregăti harta pentru încărcare.

Metoda setup() poate fi apelată înainte de metoda loadMap() pentru a asigura că harta jocului este pregătită înainte de a fi încărcată, și astfel să se asigure o încărcare corectă și funcțională a hărții în joc.

Metoda draw() este responsabilă de desenarea hărții jocului pe ecran, după ce aceasta a fost pregătită și asamblată în metodele setup() și loadMap(). Ea se ocupă de desenarea tile-urilor pe ecran în pozițiile corecte pentru a crea aspectul hărții jocului. Aceasta poate implica utilizarea coordonatelor x și y ale fiecărui tile pentru a plasa aceste imagini la locul potrivit pe ecran.Aici este implementat și design pattern-ul flyweight.

- **UI**: Prin intermediul acestei clasei, interacțiunea cu utilizatorul este gestionată într-un mod detaliat. Metoda draw() din interiorul clasei decide în ce stare de joc (gamestate) se află jocul și ce trebuie să apară pe ecran în fiecare stare. Pentru fiecare stare de joc în parte, există metode precum drawTitleScreen(), drawPauseScreen(), drawDialogueScreen(), drawOptionsScreen(), drawCharacterScreen() care se ocupă de desenarea elementelor specifice ale fiecărei stări.

În plus, în această clasă sunt implementate și metodele drawMessage() și addMessage() pentru a comunica cu utilizatorul prin afișarea de mesaje pe ecran. Textul folosit în aceste mesaje este afișat cu un font specific, care este utilizat în întreaga clasă UI (interfață utilizator).

De asemenea, sunt implementate metodele drawSubWindow() pentru a crea ferestre secundare (subwindow-uri) și pentru a ajuta la implementarea celorlalte metode de desenare, precum getXForCenteredText(), getXForAlignedRightText() și getItemIndexOnSlot() pentru a poziționa textul în concordanță cu imaginile sau alte elemente de pe ecran. Aceste metode contribuie la o prezentare coerentă și bine aliniată a textului și elementelor grafice pe ecran.

În ansamblu, această clasă gestionează detaliat modul în care jocul interacționează cu utilizatorul prin desenarea elementelor pe ecran, afișarea de mesaje și poziționarea corectă a acestora. Aceasta contribuie la crearea unei experiențe de utilizare coerente și plăcute în cadrul jocului. Mai mult toate state-urile din joc sunt implementate aici de la opțiuni și meniu la vânzarea item-urilor și cumpărarea lor sau finalul jocului și tranziția dintre niveluri.

- **AssetSetter**: În această clasă se pot plasa diverse entități și obiecte pe hartă în diferite poziții. Metodele setObject(), setNPC(), setMonster() și setInteractiveTile() sunt utilizate pentru a implementa aceste setări.

Metoda setObject() permite plasarea unui obiect pe hartă într-o anumită poziție. Aceasta poate fi folosită pentru a plasa obiecte pe harta jocului.

Metoda setNPC() permite plasarea unui NPC (personaj nejucabil) pe hartă într-o anumită poziție. NPC-urile pot fi personaje non-jucabile cu care jucătorul poate interacționa, cum ar fi personaje de quest sau vânzători într-un joc de rol.

Metoda setMonster() permite plasarea unui monstru pe hartă într-o anumită poziție. Monștrii pot fi inamici cu care jucătorul se confruntă în joc și cu care trebuie să lupte pentru a progresa.

Metoda setInteractiveTile() permite plasarea unui element interactiv pe hartă într-o anumită poziție. Acest element poate fi un obiect cu care jucătorul poate interacționa.

Aceste metode permit plasarea și configurarea diferitelor entități și obiecte pe hartă în pozițiile dorite, contribuind la crearea lumii și a experienței de joc în cadrul jocului implementat.

- **CollisionCheck**: Această clasă ajută la implementarea coliziunilor cu diferite obiecte și entități. Metoda checkTile() verifică coliziunile cu tile-uri folosite în joc, checkObject() verifică coliziunile cu obiecte, checkEntity() verifică coliziunile cu entități, iar checkPlayer() verifică coliziunile cu personajul jucător.

Metoda checkTile() este responsabilă de verificarea coliziunilor dintre jucător și tile-urile hărții, cum ar fi pământul, copacii sau alte obiecte statice. Aceasta poate verifica dacă jucătorul a intrat într-un tile interzis sau dacă se poate deplasa liber pe tile-urile permise.

Metoda checkObject() se ocupă de verificarea coliziunilor dintre jucător și obiectele din joc, cum ar fi cutii, comori sau alte obiecte interactive. Aceasta poate verifica dacă jucătorul a intrat într-o zonă ocupată de un obiect, dacă a interacționat cu obiectul sau dacă poate să îl ridice sau să îl folosească în vreun fel.

Metoda checkEntity() verifică coliziunile dintre jucător și entitățile din joc, cum ar fi monștri, NPC-uri sau alte personaje. Aceasta poate verifica dacă jucătorul a intrat într-o zonă ocupată de o entitate, dacă a fost atacat de o entitate sau dacă a interacționat cu aceasta într-un mod specific.

Metoda checkPlayer() se ocupă de verificarea coliziunilor dintre jucător și alte jucători sau entități controlate de alți jucători în cazul unui joc multiplayer. Aceasta poate verifica dacă jucătorul a intrat într-o zonă ocupată de un alt jucător sau dacă a interacționat cu acesta într-un mod specific.

Aceste metode ajută la gestionarea coliziunilor dintre jucător și diferite obiecte sau entități în joc, asigurând un comportament corespunzător al jocului și o experiență de joc plăcută pentru utilizator.

- **EventHandler:** Această clasă este folosită la implementarea schimbării mapelor prin intermediul metodelor hit() și teleport(). Metoda checkEvent() este cea care verifică dacă un anumit tile este lovit pentru a teleporta jucătorul pe o altă mapă.

- **NPC_neoi**: Această clasă implementează NPC-ul Neoi, care îl va ajuta pe jucător în aventura sa. Metoda getImage() utilizează sprite-urile specifice pentru a afișa imaginea NPC-ului pe ecran, setDialogue() stabilește dialogul pe care NPC-ul îl va avea cu jucătorul, setAction() specifică acțiunile pe care NPC-ul le poate întreprinde, iar funcția speak() facilitează interacțiunea cu NPC-ul în timpul dialogului. Această clasă este o clasă derivată (copil) a clasei Entity.

Metoda getImage() returnează imaginea NPC-ului Neoi, care este afișată pe ecran. Aceasta utilizează sprite-urile specifice pentru a reprezenta aspectul vizual al NPC-ului.

Metoda setDialogue() stabilește dialogul pe care NPC-ul Neoi îl va avea cu jucătorul. Acesta poate conține mesaje, întrebări sau opțiuni de dialog pe care jucătorul le poate selecta.

Metoda setAction() specifică acțiunile pe care NPC-ul Neoi le poate întreprinde în joc. Aceasta poate fi, de exemplu, să ofere un obiect jucătorului, să deschidă o ușă sau să activeze o anumită funcționalitate în joc.

Funcția speak() facilitează interacțiunea cu NPC-ul Neoi în timpul dialogului. Aceasta poate afișa mesajele de dialog ale NPC-ului pe ecran, să aștepte răspunsurile jucătorului și să gestioneze logica dialogului între NPC și jucător.

Această clasă NPC este o clasă derivată (copil) a clasei Entity, care este o clasă de bază pentru diferite entități din joc, cum ar fi personaje, monștri, obiecte etc. Aceasta adaugă funcționalități specifice pentru implementarea NPC-ului Neoi și interacțiunea acestuia cu jucătorul în joc.

- **NPC_vânzător**: Această clasă implementează NPC-ul Meloi, care îl va ajuta pe jucător în aventura sa. Metoda getImage() utilizează sprite-urile specifice pentru a afișa imaginea NPC-ului pe ecran, setDialogue() stabilește dialogul pe care NPC-ul îl va avea cu jucătorul, setAction() specifică acțiunile pe care NPC-ul le poate întreprinde, iar funcția speak()

facilitează interacțiunea cu NPC-ul în timpul dialogului. Această clasă este o clasă derivată (copil) a clasei Entity.

Metoda getImage() returnează imaginea NPC-ului Meloi, care este afișată pe ecran. Aceasta utilizează sprite-urile specifice pentru a reprezenta aspectul vizual al NPC-ului.

Metoda setDialogue() stabilește dialogul pe care NPC-ul Meloi îl va avea cu jucătorul. Acesta poate conține mesaje, întrebări sau opțiuni de dialog pe care jucătorul le poate selecta.

Funcția speak() facilitează interacțiunea cu NPC-ul Meloi în timpul dialogului. Aceasta poate afișa mesajele de dialog ale NPC-ului pe ecran, să aștepte răspunsurile jucătorului și să gestioneze logica dialogului între NPC și jucător.

Această clasă NPC este o clasă derivată (copil) a clasei Entity, care este o clasă de bază pentru diferite entități din joc, cum ar fi personaje, monștri, obiecte etc. Aceasta adaugă funcționalități specifice pentru implementarea NPC-ului Meloi și interacțiunea acestuia cu jucătorul în joc.

- **Particle**: Aici se implementează particulele care să sară atunci când jucătorul lovește cu toporul un interactive tile. Metoda update() este folosită pentru a modifica starea particulei pe parcursul timpului, iar metoda draw() este utilizată pentru a desena evoluția particulei pe ecran. Această clasă este o clasă derivată (mostenită) din clasa Entity.

Metoda update() actualizează starea particulei în funcție de timpul trecut și de logica implementată în cadrul acestei metode. De exemplu, poate să actualizeze poziția, viteza, durata de viață și alte caracteristici ale particulei în timpul evoluției sale.

Metoda draw() este responsabilă pentru desenarea particulei pe ecran. Aceasta utilizează informațiile actualizate din metoda update() pentru a determina cum ar trebui să fie desenată particula în cadrul scenei de joc, cum ar fi poziția, dimensiunea, culoarea și aspectul vizual al particulei.

Această clasă de particule este o clasă derivată (mostenită) din clasa Entity, care este o clasă de bază pentru diferite entități din joc, cum ar fi personaje, monștri, obiecte etc. Aceasta adaugă funcționalități specifice pentru implementarea particulelor care să sară în urma loviturilor cu toporul și pentru desenarea lor pe ecran în joc.

- **Projectile**: Această clasă implementează proiectilul pe care jucătorul îl poate trage. Metoda set() este folosită pentru a seta poziția și alte proprietăți ale proiectilului în spațiul de joc. Metoda update() este folosită pentru a actualiza starea proiectilului în timpul mișcării acestuia pe ecran. Metodele haveResources() și subtractResources() sunt utilizate pentru a verifica dacă jucătorul are suficiente resurse pentru a folosi proiectilul și pentru a scădea nivelul de resurse (de exemplu, nivelul de mana) rămas în cazul utilizării unui proiectil.

Metoda set() permite setarea poziției, vitezei, direcției și altor proprietăți ale proiectilului în spațiul de joc, în funcție de logica și regulile jocului.

Metoda update() actualizează starea proiectilului în timpul mișcării acestuia pe ecran. De exemplu, poate actualiza poziția proiectilului în funcție de viteză și direcția sa, verifica coliziunile cu alte obiecte sau entități din joc și gestiona alte aspecte ale proiectilului, cum ar fi durata de viață sau efectele speciale asociate cu acesta.

Metodele haveResources() și subtractResources() sunt utilizate pentru a verifica dacă jucătorul are suficiente resurse (de exemplu, nivelul de mana) pentru a folosi proiectilul și pentru a scădea nivelul de resurse rămas în cazul utilizării acestuia. Aceste metode pot implica interacțiunea cu alte componente ale jocului pentru a verifica și gestiona resursele jucătorului în mod corespunzător.

Această clasă de proiectil este responsabilă pentru implementarea logică și funcționalitățile asociate proiectilului în joc, inclusiv mișcarea acestuia, verificarea resurselor necesare pentru utilizarea proiectilului și actualizarea stării acestuia pe parcursul timpului. La fel ca celelalte clase menționate anterior, aceasta clasă este derivată din clasa Entity.

-**Mon_purpleSlime**: Această clasă implementează monștrii din primul nivel al jocului. Metoda getImage() utilizează sprite-urile specifice pentru a afișa aspectul vizual al acestor creaturi. Metoda setAction() implementează abilitățile și comportamentul monștrilor în joc. Metoda damageReaction() verifică reacția lor atunci când sunt omorâți de către jucător, cum ar fi animații speciale sau sunete. Metoda checkDrop() verifică ce obiecte vor fi lăsate în urma morții monștrilor, cum ar fi inimile, manele sau monedele.

Metoda getImage() este responsabilă de încărcarea și afișarea sprite-urilor specifice pentru aspectul vizual al monștrilor în joc, în funcție de tipul și starea acestora.

Metoda setAction() implementează logica și comportamentul monștrilor în timpul jocului, inclusiv mișcarea, atacul, defensa sau alte abilități speciale pe care le pot avea monștrii.

Metoda damageReaction() gestionează reacția monștrilor atunci când sunt loviți și omorâți de către jucător, cum ar fi animații de moarte sau sunete speciale, și poate implica și alte acțiuni, cum ar fi actualizarea punctajului sau gestionarea resurselor jucătorului.

Metoda checkDrop() verifică ce obiecte sau resurse vor fi lăsate în urma morții monștrilor și le plasează pe hartă, astfel încât jucătorul să le poată colecta. Aceasta poate implica generarea aleatoare a obiectelor, verificarea resurselor disponibile sau alte acțiuni specifice asociate cu drop-ul monștrilor în joc.

Această clasă de monștri din primul nivel este responsabilă pentru implementarea logicii și comportamentului acestora în joc, inclusiv aspectul vizual, abilitățile, reacțiile la daune și gestionarea drop-ului de obiecte. Ca și celelalte clase menționate anterior, aceasta clasă este derivată din clasa Entity.

- **Mon_redSlime**: Această clasă implementează monștrii din al doilea nivel al jocului. Metoda getImage() utilizează sprite-urile specifice pentru a afișa aspectul vizual al acestor creaturi. Metoda setAction() implementează abilitățile și comportamentul monștrilor în joc. Metoda damageReaction() verifică reacția lor atunci când sunt omorâți de către jucător, cum ar fi animații speciale sau sunete. Metoda checkDrop() verifică ce obiecte vor fi lăsate în urma morții monștrilor, cum ar fi inimile, manele sau monedele.

Metoda getImage() este responsabilă de încărcarea și afișarea sprite-urilor specifice pentru aspectul vizual al monștrilor în joc, în funcție de tipul și starea acestora.

Metoda setAction() implementează logica și comportamentul monștrilor în timpul jocului, inclusiv mișcarea, atacul, defensa sau alte abilități speciale pe care le pot avea monștrii. Acest monstru poate trage proiectile.

Metoda damageReaction() gestionează reacția monștrilor atunci când sunt loviți și omorâți de către jucător, cum ar fi animații de moarte sau sunete speciale, și poate implica și alte acțiuni, cum ar fi actualizarea punctajului sau gestionarea resurselor jucătorului.

Metoda checkDrop() verifică ce obiecte sau resurse vor fi lăsate în urma morții monștrilor și le plasează pe hartă, astfel încât jucătorul să le poată colecta. Aceasta poate implica generarea aleatoare a obiectelor, verificarea resurselor disponibile sau alte acțiuni specifice asociate cu drop-ul monștrilor în joc.

Această clasă de monștri din primul nivel este responsabilă pentru implementarea logicii și comportamentului acestora în joc, inclusiv aspectul vizual, abilitățile, reacțiile la daune și gestionarea drop-ului de obiecte. Ca și celelalte clase menționate anterior, aceasta clasă este derivată din clasa Entity.

- **Mon_Boss**: Această clasă implementează monstrul final. Metoda getImage() utilizează sprite-urile specifice pentru a afișa aspectul vizual al acestei creaturi. Metoda getAttackImage() utilizează sprite-urile specifice pentru a afișa aspectul vizual al acestei creaturi atunci când atacă. Metoda setAction() implementează abilitățile și comportamentul monștrilor în joc. Metoda damageReaction() verifică reacția lor atunci când sunt omorâți de către jucător, cum ar fi animații speciale sau sunete.

Metoda getImage() este responsabilă de încărcarea și afișarea sprite-urilor specifice pentru aspectul vizual al monștrilor în joc, în funcție de tipul și starea acestora.

Metoda setAction() implementează logica și comportamentul monștrilor în timpul jocului, inclusiv mișcarea, atacul, defensa sau alte abilități speciale pe care le pot avea monștrii.

Metoda damageReaction() gestionează reacția monștrilor atunci când sunt loviți și omorâți de către jucător, cum ar fi animații de moarte sau sunete speciale, și poate implica și alte acțiuni, cum ar fi actualizarea punctajului sau gestionarea resurselor jucătorului.

Metoda mesajTerminareJoc() verifică dacă monstrul este înfrânt și afișează un mesaj de felicitare.

Această clasă a monstrului este responsabilă pentru implementarea logicii și comportamentului acestuia în joc, inclusiv aspectul vizual, abilitățile, reacțiile la daune și gestionarea drop-ului de obiecte. Ca și celelalte clase menționate anterior, aceasta clasă este derivată din clasa Entity.

- **InteractiveTile**: Clasa interactiveTile este cea care implementează tile-uri ce pot fi tăiate de către jucător cu toporul și care pot fi înlocuite. Metoda getDestroyedForm() ajută la găsirea formei distruse a tile-urilor. Metodele update() și draw() actualizează starea și forma tile-rilor și desenează aceste actualizări pe ecran. Această clasă este moștenită din clasa Entity.

Metoda getDestroyedForm() este responsabilă pentru a returna forma tile-ului după ce a fost tăiat de către jucător cu toporul. Aceasta poate implica schimbarea texturii sau aspectului tile-ului pentru a reflecta starea sa de distrugere.

Metodele update() și draw() sunt responsabile pentru actualizarea stării și formei tile-urilor și pentru desenarea acestora pe ecran. Metoda update() poate implica schimbarea stării tile-ului în funcție de interacțiunile jucătorului cu acesta, cum ar fi tăierea cu toporul. Metoda draw() poate implica desenarea texturii sau aspectului actualizat al tile-ului pe ecran, pentru a reflecta schimbările aduse de metoda update().

Clasa interactiveTile este responsabilă pentru implementarea tile-urilor interactive din joc, care pot fi tăiate și înlocuite de către jucător. Aceasta poate fi utilizată pentru a crea mecanici de joc interesante și interacțiuni interactive cu mediul înconjurător.

- **It_dayTree**: Clasa aceasta ajută la implementarea tile-urilor interactive, de aceea este și moștenită din clasa interactiveTile. Metodele care apar în aceasta sunt toate moștenite din cele precedente, precum: getDestroyedForm() sau isCorrectItem().

Metoda getDestroyedForm() este responsabilă pentru a returna forma tile-ului după ce a fost tăiat de către jucător cu toporul, similar cu implementarea din clasa interactiveTile.

Metoda isCorrectItem() poate fi responsabilă pentru a verifica dacă jucătorul are un anumit obiect sau resursă necesară pentru a interacționa cu tile-ul în mod corespunzător. Aceasta poate fi utilizată pentru a implementa restricții sau cerințe specifice pentru interacțiunea cu tile-urile interactive.

Această clasă, moștenind din clasa interactiveTile, poate adăuga funcționalități suplimentare sau modificări specifice pentru tile-urile interactive, în funcție de necesitățile jocului sau de design-ul specific al nivelului.

- **IT_trunk**: Clasa aceasta implementează forma distrusă a tile-urilor interactive. Este moștenită din clasa interactiveTile. Această clasă, moștenind din clasa interactiveTile, poate conține funcționalități specifice pentru forma distrusă a tile-urilor, cum ar fi aspectul grafic, comportamentul sau proprietățile speciale ale formei distruse.

- **Sound**: Clasa Sound este cea care implementează muzica și sunetele speciale în acest joc. Metoda play() pornește redarea sunetului, stop() oprește redarea acestuia, iar loop() permite redarea continuă a melodiei. Metoda checkVolume() ajută la verificarea volumului și ajustarea acestuia.

Metoda play() este responsabilă pentru a porni redarea sunetului, indiferent dacă este vorba despre muzică de fundal sau sunete speciale asociate evenimentelor din joc. Aceasta poate implica încărcarea și redarea unui fișier audio specific folosind librării sau metode specifice ale platformei pe care rulează jocul.

Metoda stop() permite oprirea redării sunetului în timpul jocului. Aceasta poate implica oprirea imediată a redării unui sunet sau a melodiei de fundal, în funcție de implementarea specifică.

Metoda loop() permite redarea continuă a melodiei de fundal, astfel încât aceasta să se repete în mod automat atunci când ajunge la finalul redării. Aceasta poate fi utilă pentru a crea o atmosferă sonoră coerentă în joc.

Metoda checkVolume() verifică și ajustează volumul sunetelor în joc, în funcție de preferințele jucătorului sau de setările sistemului. Aceasta poate implica citirea setărilor de volum dintr-un fișier de configurare sau interacțiunea cu API-uri specifice ale sistemului de operare pentru a regla volumul sunetelor.

Clasa Sound este responsabilă pentru implementarea și gestionarea muzicii și sunetelor speciale în joc, inclusiv pornirea, oprirea, redarea continuă și ajustarea volumului. Aceasta poate fi utilizată pentru a crea o experiență auditivă plăcută și captivantă în timpul jocului.

- **OBJ:** Clasa aceasta este crucială pentru funcționarea obiectelor în joc. Metoda draw() se ocupă de afișarea sprite-urilor obiectelor pe ecran și de aplicarea tuturor actualizărilor necesare. Metoda use() determină dacă un obiect poate fi folosit de către jucător în funcție de anumite condiții sau criterii. Metoda setup() are rolul de a asigura încărcarea sprite-urilor obiectelor din resursele jocului, pregătindu-le pentru utilizare în cadrul jocului. Această clasă oferă un cadru robust pentru gestionarea obiectelor și interacțiunea cu acestea în cadrul jocului.

- **Obj_Coin, Obj_Heart, Obj_Key, Obj_Mana, Obj_Potion_Mana, Obj_Potion_Blue**: Aceste clase implementează anumite obiecte care pot fi folosite de către jucător prin intermediul metodei use(). Sunt moștenite din clasa OBJ, care furnizează funcționalități de bază pentru gestionarea obiectelor în joc. Fiecare clasă specifică de obiect poate avea propriile metode și proprietăți care permit utilizarea, afișarea și interacțiunea cu obiectele respective în cadrul jocului.

- **Obj_Axe, Obj_Sword_1, Obj_shield_green, Obj_frunza, Obj_elixir, Obj_spada:** Aceste clase implementează anumite obiecte ce nu pot fi utilizate de către jucător. Sunt moștenite din clasa OBJ, care furnizează funcționalități de bază pentru gestionarea obiectelor în joc.

- **Obj_Fireball, Obj_Rock:** Aceste clase implementează logica proiectilelor pentru jucător și monstru în joc. Ele sunt mostenite din clasa projectile, care oferă funcționalități de bază pentru gestionarea proiectilelor. Metoda getImage() este folosită pentru a obține imaginea asociată cu proiectilul, iar metodele haveResources() și subtractResources() sunt folosite pentru a verifica și a scădea resursele necesare pentru utilizarea proiectilului, respectiv pentru a fi consistente cu logica generală a jocului.

- **SaveLoad**: Clasa aceasta implementează baza de date cu ajutorul căreia se salvează viața jucătorului, nivelul de mana rămas și nivelul la care se află. Metoda save() salvează datele în baza de date si creează tabelul din aceasta, iar funcția load() ia datele și le folosește.

- **DataStorage:** Clasa aceasta stochează informațiile rămase de salvat și le pregătește pentru a fi salvate.

- **SaveOthers:** Clasa aceasta salvează informațiile din DataStorage() precum inventarul jucătorului, starea inventarului sau obiectele de pe hartă.

- **PathFinder:** Această clasă implementează mecanismele din spatele algoritmului de pathfinding, furnizând mecanismele necesare pentru găsirea celor mai scurte rute între două puncte într-un graf sau o rețea de noduri. Clasa setNodes() este o metodă sau funcție utilizată într-o implementare specifică a algoritmului de pathfinding pentru a seta sau defini nodurile disponibile în rețeaua sau graful dat. Aceasta primește ca parametri nodurile și poate fi responsabilă de inițializarea și configurarea nodurilor în vederea aplicării algoritmului de căutare a drumului.

Implementarea exactă a metodei setNodes() poate varia în funcție de contextul și limbajul de programare utilizat. În general, aceasta poate primi o listă de noduri sau poate accesa o structură de date care conține informațiile despre nodurile rețelei sau grafului.

Metoda setNodes() poate fi responsabilă de crearea obiectelor de tip nod și de atribuirea sau setarea proprietăților specifice ale fiecărui nod, cum ar fi coordonatele, conexiunile către alte noduri și alte informații relevante pentru algoritmul de pathfinding.

De exemplu, într-o implementare simplă a algoritmului A*, metoda setNodes() poate crea obiecte de tip nod și le poate atribui coordonate x și y pe o hartă, precum și conexiuni cu alte noduri. Aceste conexiuni pot fi stocate sub forma unei liste de vecini sau prin intermediul unei structuri de date, cum ar fi o matrice de adiacență sau o listă de muchii.

Prin utilizarea metodei setNodes() și definirea corespunzătoare a nodurilor într-un sistem de rețele sau grafuri, algoritmul de pathfinding poate să opereze asupra acestor noduri pentru a găsi drumul optim între două puncte în rețea.

Metoda resetNodes() este o metodă sau funcție utilizată în implementarea algoritmului de pathfinding pentru a reseta starea nodurilor din rețea sau graful dat. Aceasta are rolul de a readuce toate nodurile la starea inițială sau de a șterge orice informație calculată anterior, pregătindu-le pentru o nouă rulare a algoritmului de pathfinding.

Implementarea exactă a metodei resetNodes() poate varia în funcție de contextul și limbajul de programare utilizat. În general, aceasta poate parcurge toate nodurile din rețea și să le readucă la valorile sau starea inițială.

Metoda resetNodes()are următoarele responsabilități:

Resetează costurile asociate fiecărui nod: Dacă algoritmul de pathfinding utilizează costuri pentru a calcula distanțe sau estimări, metoda poate resetă costurile asociate fiecărui nod la valorile inițiale sau la o valoare standard.

Restabilește starea deschis/inchis a nodurilor: În algoritmul A* sau alte algoritme bazate pe listele deschise și închise, metoda poate reseta toate nodurile la starea deschisă sau închisă inițială, pentru a putea relua algoritmul de la început.

Șterge informațiile adiționale calculate: Dacă au fost calculate informații suplimentare pentru fiecare nod în timpul algoritmului de pathfinding (cum ar fi drumul minim sau nodul anterior), metoda resetNodes() poate șterge aceste informații pentru a pregăti nodurile pentru o nouă rulare a algoritmului.

Prin utilizarea metodei resetNodes(), se poate asigura că algoritmul de pathfinding pornește de la starea inițială și că nodurile sunt pregătite pentru a fi procesate în vederea găsirii unui nou drum optim între două puncte în rețea sau graful dat.

Metoda search() este o metodă sau funcție utilizată în implementarea algoritmului de pathfinding pentru a căuta cel mai scurt drum între două noduri într-un graf sau o rețea. Această metodă conține logica principală a algoritmului de căutare și este responsabilă de explorarea nodurilor în conformitate cu algoritmul specific utilizat.

Implementarea exactă a metodei search() poate varia în funcție de algoritmul de pathfinding utilizat și de contextul în care este aplicat. În general, aceasta va implica un ciclu de explorare și actualizare a nodurilor până când se găsește drumul dorit sau până când toate nodurile accesibile au fost explorate.

Metoda este folosită la Npc_Neoi si la toți monștrii mai puțin boss-ul.

-**Bibliografie**:

-https://www.youtube.com/watch?v=om59cwR7psI&list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq.