MoonLight

Archir Maria-Mirabela

Grupa 1208B

I. Proiectarea Contextului

A) Definirea genului de joc:

MoonLight este un joc de tip platformă, single-player, în care jucătorul controlează un personaj supus mișcării prin diferite niveluri, ce sunt creionate într-o perspectivă 2D. Selen, protagonista noastră, trebuie să parcurgă în total trei niveluri, a căror complexitate crește gradual. Pe parcursul fiecărei provocări, prințesa trebuie să facă față la diferite obstacole, să supraviețuiască unor ființe supranaturale care o atacă și să își facă drumul mai ușor prin colecționarea stelelelor călăuză.

Unul dintre scopurile jocului îl presupune colectarea banutilor aurii . Eroina are la îndemână o serie de abilități esențiale precum: săritul, alergatul și arme puternice ce sunt utilizate împotriva antagoniștilor perseverenți. Dacă prințesa va reuși să treacă peste toate aceste impedimente, va dobândi reîntoarcerea în ținutul natalși eliberarea Lunii-personaj parental.

B) Definirea tematicii:

Tema jocului MoonLight este una de aventură și acțiune. Firul narativ al poveștii are la bază încercarea de a salva figura maternă a personajului principal. Această intrigă conduce la evenimente de intensitate maximă, însumând aventura

eroinei Selen, care trebuie să navigheze prin diferite dimensiuni și să supraviețuiască probelor necesare.

C) Definirea poveștii:

A fost odată ca niciodată, în adâncurile unui univers vast și îndepărtat, o tânără prințesă pe nume Selen. Fata cu ochii mari și migdalați trăia într-o lume plină de speranță, lume în care Luna îi era mamă, iar Stelele călăuză. Dar odată cu trecerea timpului, o amenințare periculoasă se apropia de casa ei, războiul furându-i mama și lăsând în urma lui, locul cioburilor de sticlă .

Într-o noapte, în timp ce se ruga spre cerul nopții, fiica Lunii a observat o stea strălucitoare care părea să-i strige numele. Selen a simțit o stranie trăire în suflet și a știut că aceasta era chemarea ei spre o călătorie nouă ce avea să aducă redobândirea liniștii în regat, alături de brațele calde ale maicii sale. Așa că, cu inima plină de îndrăzneală și emoții, fata a plecat de pe planeta sa natală și s-a aventurat spre alte zări, libertății.

Tânăra prințesă este supusă unui proces complex, menit să o treacă prin diferite situații dificile și lecții de viață. Aceasta are de descoperit trei lumi diferite față de ceea ce știuse până acum, așa că în drumul său se confruntă cu tot felul de piedici neștiute. În fiecare tărâm, Selen este nevoită să lupte împotriva unor creaturi periculoase și să găsească soluții logice la probleme neprevăzute. Traiectoria acestei aventuri aduce ascensiuni majore asupra eroinei noastre, care își dezvoltă inteligența și înțelepcinea.

Finalul poveștii, reiterează “drumul Lunii”, parcurs de Selen care reușește să ajungă în sfârșit acasă și să își salveze mama.

II. Proiectarea Sistemului

A) Definirea regulilor principale/Game Mechanics:

Mecanica jocului propus este constituită dintr-o serie de reguli ce au menirea de a guverna și ghida acțiunile jucătorului, urmărind, în aceeași ordine de idei, și răspunsul interacțiunii. MoonLight are la bază un set de reguli bine stabilite care urmăresc câteva aspecte precum:

➔ **Modul în care pornește jocul**:

Se deschide jocul de pe dispozitivul dorit și se pune în acțiune prin intermediul butonui de start. Rezultatul acestei acțiuni va fi vizibil pe ecran, prin apariția unei mici descrieri despre modul în care jocul nostru functionează. Odată stabilit acest aspect, jucătorul va trece spre următorul pas, și anume selectarea primului nivel, intrând în atmosferă.

➔ **Cum se joacă?**

Aceste resurse sunt puse la dispoziția jucătorului încă de la început: ne confruntăm cu un joc de platformă 2D, surprins dintr-o perspectivă side view, de tip single player în care personajul principal, prințesa Selen, este controlată prin intermediul tastelor direcționale. Aceasta dispune de o serie de opțiuni: se poate deplasa dreapta/stanga, poate sări în sus și în jos, fiind afectată de gravitație.

Obiectivul central este reprezentat de învingerea inamicilor. În cadrul celor trei lumi supranaturale, identificate drept

niveluri, prințesa trebuie să treacă diferite obstacule precum prăpăstii periculoase, drumuri încurcate și inamici periculoși.

De asemenea, ea trebuie sa colecteze cât mai multe steluțe și banuti, ce o vor ajuta în diverse situații. Lupta cu antagoniștii este una destul de strânsă, personajul trebuie să se întâlnească cu aceștia, și să îi ucidă.

➔ **Cum se progresează:**

Trecerea de la un nivel la altul presupune îndeplinirea unor serii de îndatorii. În funcție de complexitatea lumii, scopul fiecărui nivel este de a colecta numărul bănuți/steluțe și de a încerca anihilarea inamicilor pe traseu. În momentul în care eroul reușește să scape de toți inamicii, acesta poate trece în etapa urmatoare.

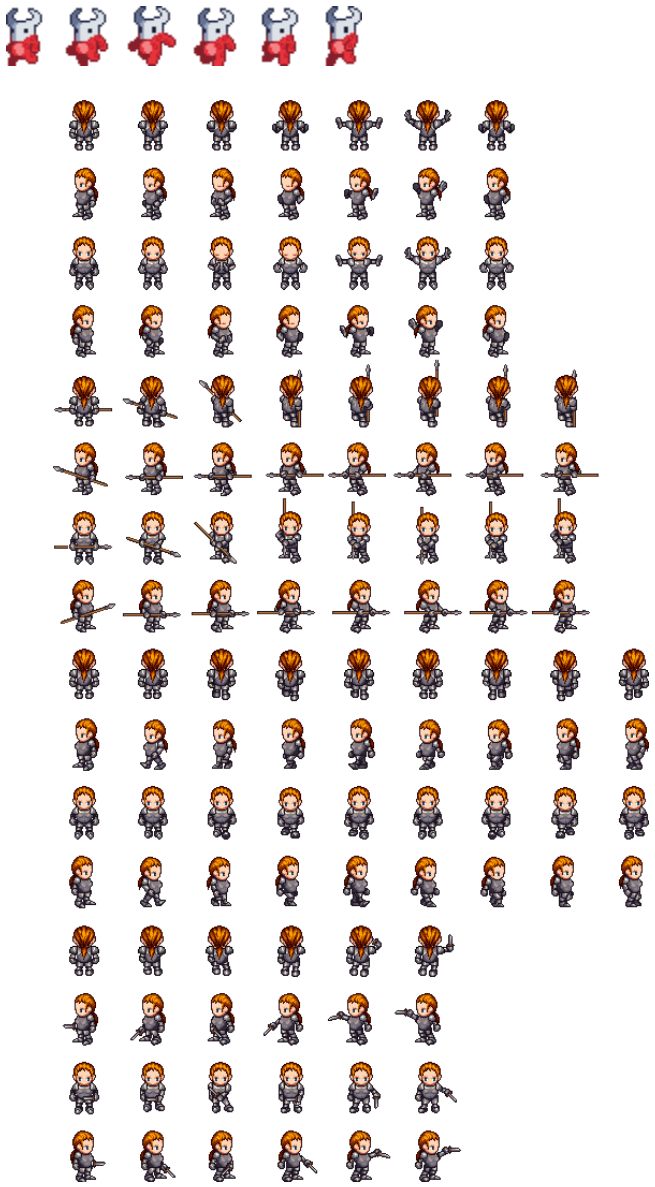
➔ **Cum se castigă:**

Pentru a caștiga jocul MoonLight, este necesară parcurgerea tuturor nivelurilor. Bătălia supremă și adevărata provocare este dată în ultimul nivel, în care Selen trebuie să parcurgă cel mai aprig traseu și să se lupte cu un număr semnificativ de răufăcători, scăpand de oamenii răi ce i au furat mama.

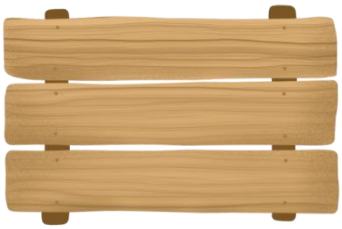
III. Proiectarea Conținutului

A) Definirea caracterelor:

După cum se poate observa, în imaginile de mai jos sunt prezentate o serie din spritesheet-urile ce vor fi utilizate în cadrul acestui joc..



B) Definirea animaţiilor şi a obiectelor



IV. Proiectarea Nivelurilor

A) Definirea hărţii:

Definirea hărții unui joc implică stabilirea terenului, a elementelor și a caracteristicilor sale, precum și modul în care acestea interactionează între ele. În principiu, harta jocului este o reprezentare grafică sau textuală a terenului , care arată obstacolele, resursele, locațiile și alte elemente importante.

În imaginea de mai jos a fost proiectată o schiță a primului nivel. Imaginile caracterelor și a animațiilor sunt doar un model, în construcția personajului principal și a inamicilor se vor utiliza spritesheet-urile declarate în secțiunea “Proiectarea Conținutului”.

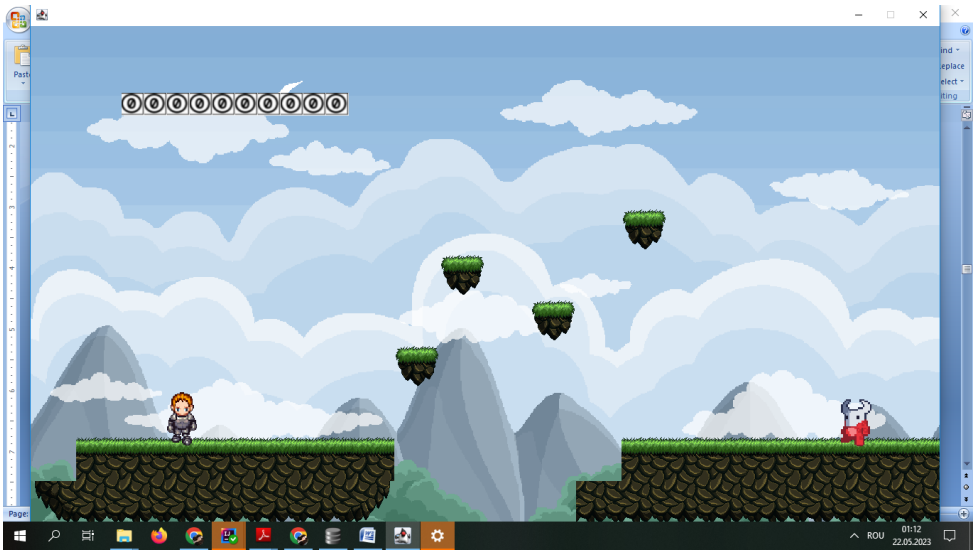
Schița servește drept exemplu pentru modul în care va fi realizată harta jocului, ipostaza caracterelor și modul în care a fost gândita plasarea obstacolelor. De asemenea, a fost utilizată o abordare simplistă, s-a dorit ca nivelul să fie ușor de parcurs pentru jucători și să introducă mencanicile de bază.

După cum se poate observa, tabloul în care se desfasoară prima probă a jocului este un tărâm senin, împrejmuit de copaci și vegetație, creionând imaginea unei păduri răsfirate. Harta este fundamentul unei mulțimi de piese/tile-uri ce asamblează în cazul de față următoarele elemente: blocuri solide necesare deplasamentului;

Eroina trebuie să se mentină la nivelul solului astfel încât să poată înainta și a se întâlni cu dușmanii săi.

Deplasamentul se face spre dreapta, iar mișcarea de bază în cazul de față este jump-ul.

Nivelul se încheie cu succes dacă toți inamicii sunt omorâți.

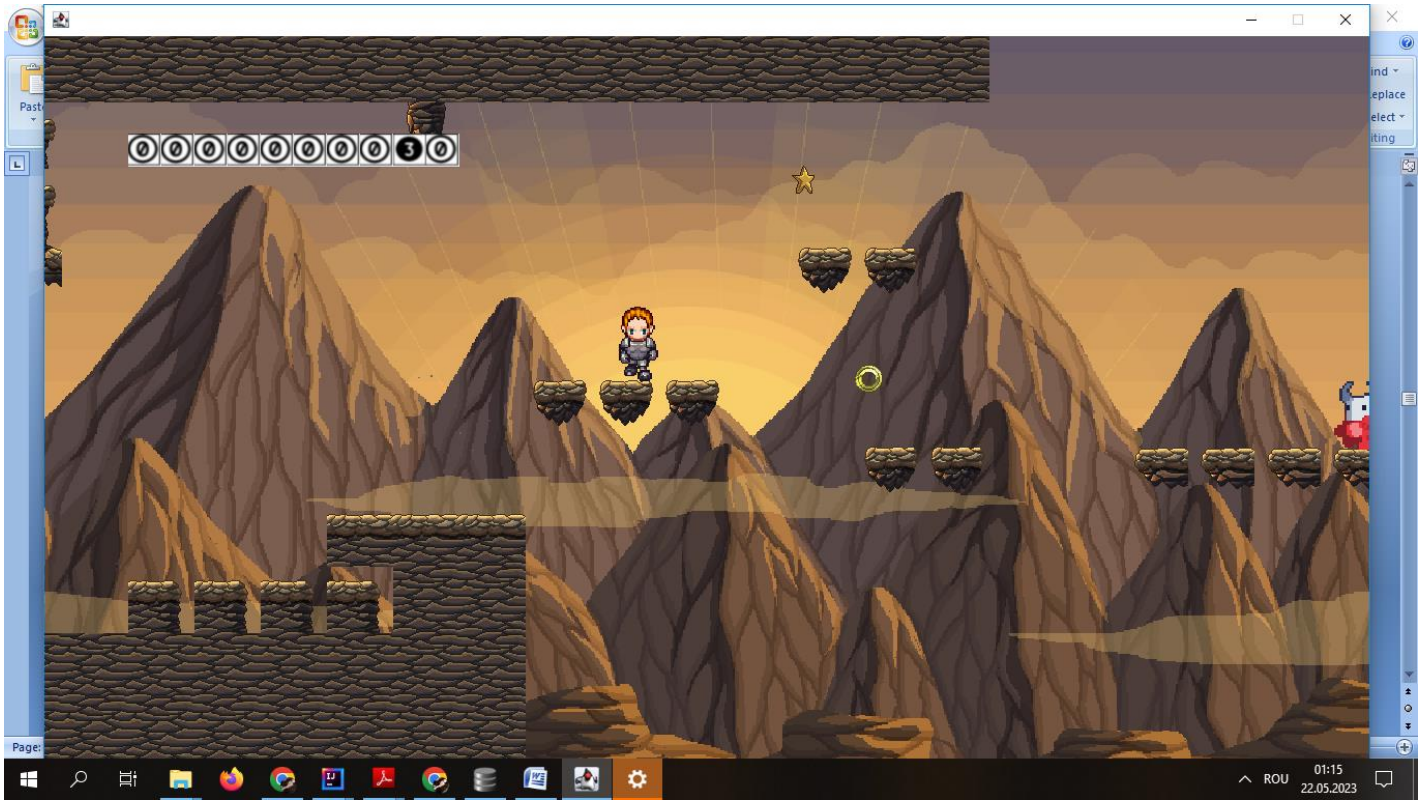


Nivelul 2

Nivelul 2 are o asceniune susceptibilă încă de la prima vedere. De data aceasta suntem proiectați într-o dimensiune căluduroasă, ce imprimă ideea de Tărâm arid , alături de piesele hărții - natură diferită față de primul caz.

Inițial, personajul central este poziționat pe o suprafață solidă, de unde are posibilitatea de a colecta câteva steluțe. În momentul avansării, Selen trebuie

să depășească prăpastiile adânci prin sărituri repetate. Remarcabil față de primul nivel este faptul că, de data aceasta avem un număr mult mai mare de antagoniști ce trebuiesc evitați, deci putem pune problema unei complexități ridicate.

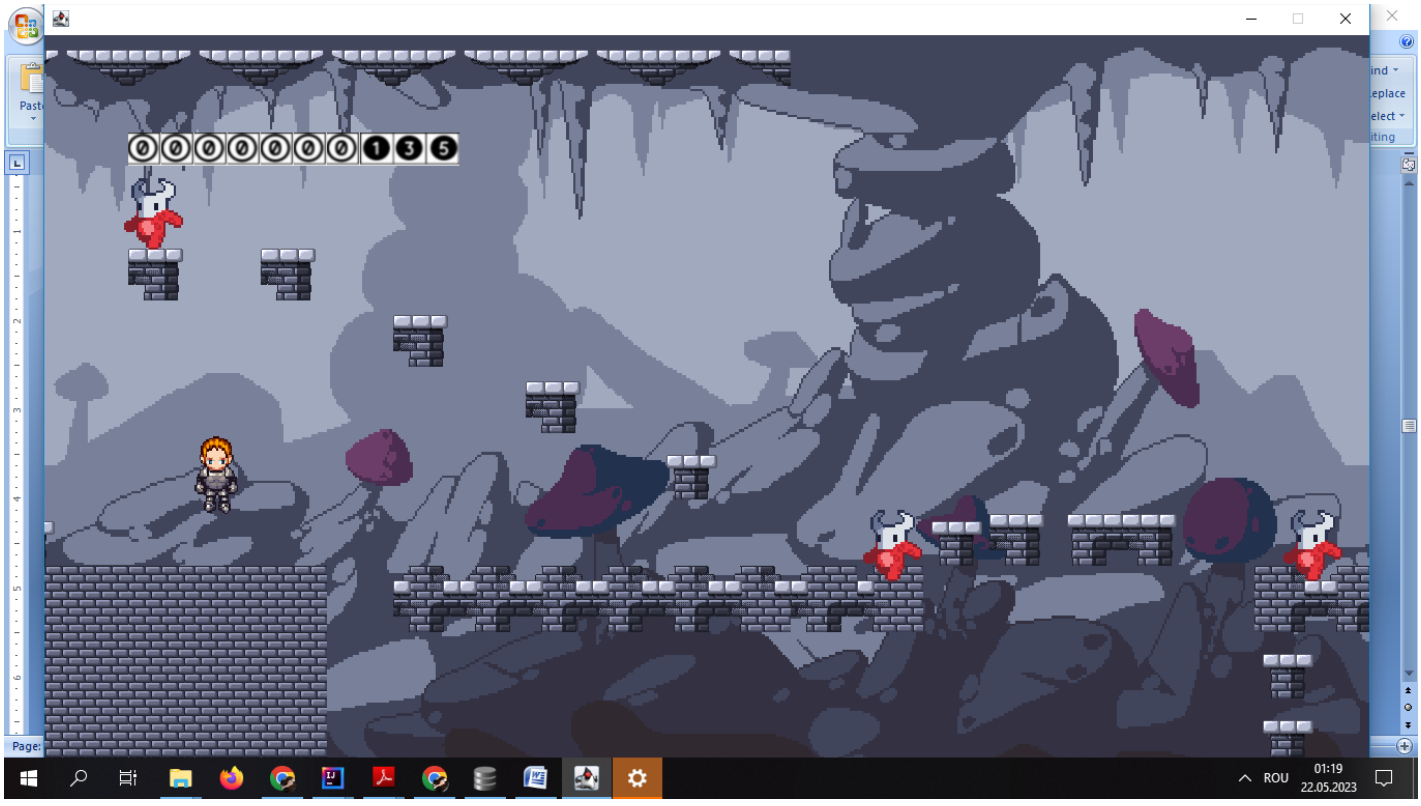


În aceeași ordine de idei, prințesa trebuie să parcurgă o serie de căi bifurcate pentru a obține steluțele și bănuți

Nivelul 3

Nivelul 3, și cel din urmă, transpune aventura finală a protagonistei noastre, care se află de această dată, într-un tărâm mult mai periculos. Numărul dușmanilor săi crește semnificativ, iar viteza cu care aceștia se deplasează este cu mult mai debordantă.

Un prim element dificil îl reprezintă urcarea treptată în partea de sus a jocului, lupta cu inamicii având loc pe bucățele mult mai înguste de sol.



V. Proiectarea Interfeței

Definirea meniului de joc și a interfeĢei:

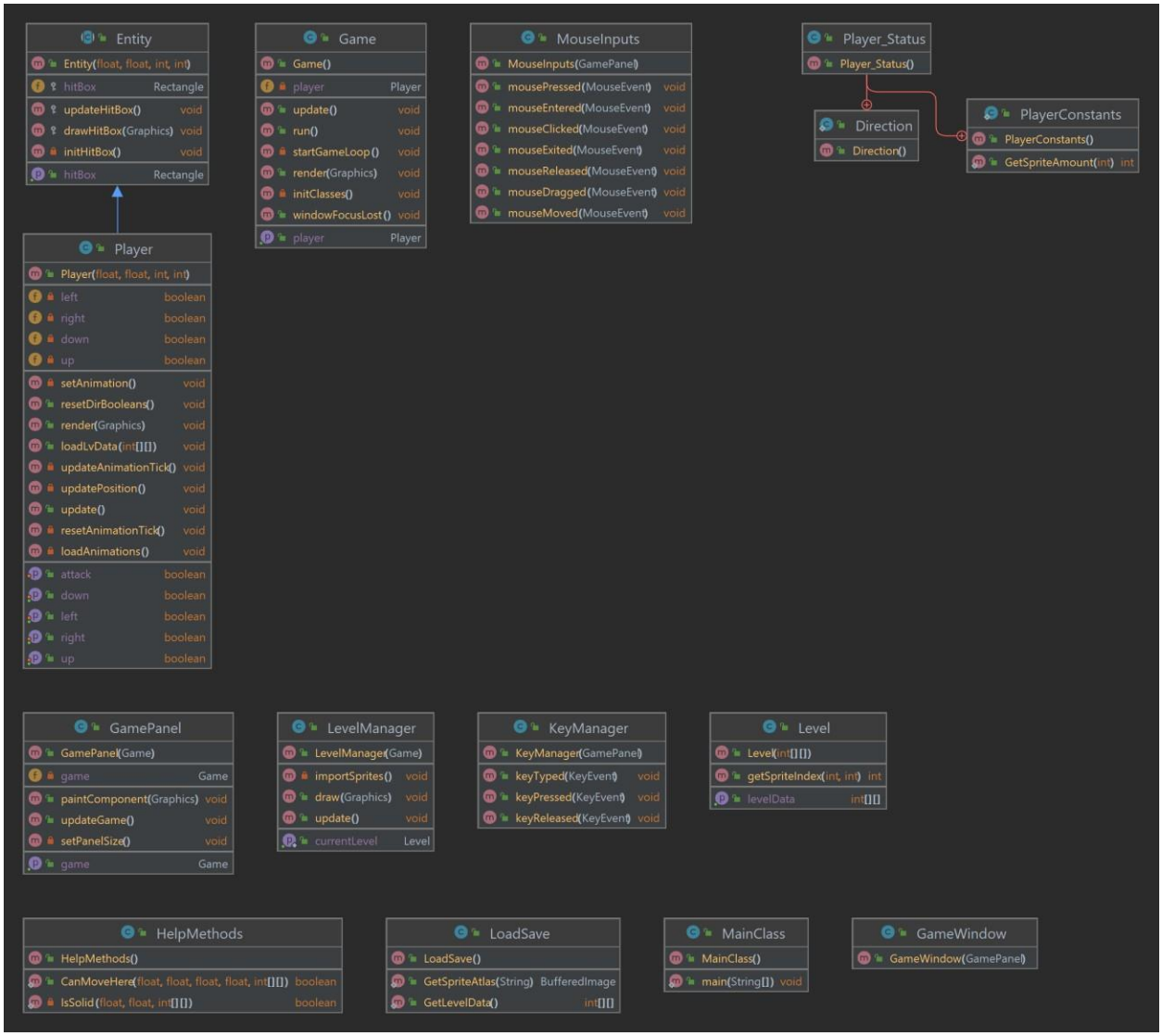
Meniul este relativ, destul de simplu. Imediat lansat în execuție, utilizatorul va interacționa cu interfața din cadrul jocului, adică prima dată pe ecran va fi transpusă o imagine vizuală-semnificativă- ce va conține butonul „Play” , „Quit” si “Option”. Odată ce se va selecta prima varintă, interfața prezentată va fi cea a primului nivel.

->Etapa 2

Etapa 2:

Arhitectural Design Document

Diagrame UML:



Descriere clase proiect:

Clasa MainClass:

Pe baza diagramei prezentate, se poate observa ca in clasa intitulata „Main Class” se creeaza metoda principala main dupa care Java se iota ori de cate ori incepe programul. In cadrul acestei functii, se creeaza o instanta de tipul Game ce are menirea de porni activitatea prin apelul metodei StartGame().

Clasa Game:

Aceasta clasa poate sa mai fie numita si „Clasa Main”, deoarece are menirea de a adauga si pune totul impreuna. Aceasta clasa reprezinta nucleul jocului si coordoneaza toate entitatile si nivelurile sale. Ofera, de asemenea, o interfata grafica pentru joc, prin intermediul obiectelor GameWindow si Game Panel.

Metoda constructorului initializeaza instantele claselor GameWindow si GamePanel si incepe bucla principala a jocului prin apelarea functiei startGameLoop();

Functia InitClasses() initializeaza instantele claselor LevelManager si Player, incarcand datele nivelului curent.

Functia startGameLoop() porneste un nou fir de executie prin implementarea interfeteu Runnable.

Functia update() apeleaza metodele de actualizare pentru LevelManager si Player.

Functia render(Graphics g) primeste un obiect Graphics si utilizeaza acest obiect pentru a desena nivelul si jucatorul.

Functia run() implementeaza bucla principala a jocului si isi propune sa mentina un numar constant de cadre pe secunde si de actualizari.

Functia getPlayer() returneaza o instanta a Player-ului;

Clasa GameWindow:

Această clasă reprezintă fereastra jocului și este responsabilă pentru afișarea interfeței grafice și manipularea evenimentelor de focus ale ferestrei.

Funcția constructorului primește o instanță GamePanel și creează un obiect JFrame, setează opțiuni precum acțiunea butonului de închidere, localizarea și dimensiunea ferestrei, precum și afișarea acesteia.

De asemenea, se adaugă un ascultător de evenimente de focus al ferestrei pentru a menține focusul pe panoul de joc când utilizatorul face clic în afara ferestrei.

Funcția windowGainedFocus() din clasa WindowFocusListener aferentă clasei GameWindow, este apelată atunci când fereastra primește focusul, iar aceasta redirecționează evenimentul la panoul de joc pentru a menține focusul pe joc.

Funcția windowLostFocus() din clasa WindowFocusListener este goală în această implementare și nu face nimic.

Clasa GamePanel:

Clasa GamePanel este o componentă grafică a jocului și reprezintă panoul principal al jocului. Ea extinde clasa JPanel și conține toate elementele grafice și evenimentele de interacțiune cu mouse-ul și tastatura necesare pentru a rula jocul.

Metoda constructor GamePanel are un parametru Game și un obiect MouseInputs. Ea initializează dimensiunea panoului și adaugă un manager de evenimente MouseInputs pentru a captura evenimentele de mișcare a mouse-ului și adaugă un manager de evenimente KeyManager pentru a captura evenimentele de tastatură. Metoda setPanelSize() stabilește dimensiunile panoului și metoda updateGame() este goală și nu face nimic.

Metoda paintComponent(Graphics g) este responsabilă pentru desenarea elementelor grafice pe panoul de joc. Ea suprascrie metoda paintComponent() din clasa JPanel și primește un obiect Graphics g ca parametru. Apoi, metoda apelază metoda render(g) din clasa Game, care este responsabilă pentru desenarea nivelului curent și a jucătorului.

Functia getGame() returnează obiectul de tip Game din această instanță a GamePanel. Acest lucru ne permite să accesăm și să lucrăm cu obiectul Game din afara clasei GamePanel.

Clasa KeyManager:

Această clasă este responsabilă de gestionarea evenimentelor de tastatură și transmite aceste evenimente obiectului GamePanel.

Metoda keyPressed() este apelată atunci când o tastă este apăsată. Ea verifică codul tastei și setează starea corespunzătoare a obiectului Player (determinat prin intermediul Game), pentru a indica direcția în care jucătorul trebuie să se miște.

Metoda keyReleased() este apelată atunci când o tastă este eliberată. Ea face același lucru ca și keyPressed(), dar resetează starea tastelor direcționale corespunzătoare, astfel încât jucătorul să nu mai meargă în direcția respectivă.

Metoda keyTyped() este apelată atunci când o tastă este apăsată și eliberată. Această metodă nu este utilizată în acest cod și, prin urmare, este goala.

Clasa MouseInputs:

Această clasă implementează interfețele MouseListener și MouseMotionListener și este utilizată pentru a gestiona intrările de la mouse.

Metoda mouseClicked este apelată atunci când un buton de mouse este apăsat și eliberat într-un interval scurt de timp, indicând un clic. În acest caz, dacă butonul apăsat este butonul stâng, jucătorul este setat pentru a ataca.

Metodele mousePressed, mouseReleased, mouseEntered, mouseExited, mouseDragged și mouseMoved sunt metodele definite de interfețele implementate și rămân goale în această clasă, dar ar putea fi implementate pentru a trata evenimentele de mouse relevante în joc

Clasa Entity:

Aceasta clasă reprezintă o clasă abstractă pentru toate entitățile din joc, cum ar fi jucătorul sau inamicii. Aceasta conține variabile pentru poziția (x și y), lățime și înălțime, precum și un hitbox (zonă de coliziune) pentru entitate.

Constructorul Entity(float x, float y, int width, int height) inițializează variabilele pentru poziție, lățime și înălțime și apelează funcția initHitBox() pentru a inițializa hitbox-ul entității.

Funcția protejată drawHitBox(Graphics g) este folosită pentru a desena hitbox-ul entității în cadrul jocului.

Funcția privată initHitBox() initializează hitbox-ul entității cu poziția și dimensiunile sale inițiale.

Funcția protejată updateHitBox() actualizează poziția hitbox-ului în funcție de poziția entității.

Funcția publică Rectangle getHitBox() returnează hitbox-ul entității pentru a fi folosit în detectarea coliziunilor cu alte entități din joc.

Clasa Player:

Clasa Player este o subclasă a clasei Entity și descrie entitatea jucătorului din joc. Aceasta conține funcții pentru gestionarea mișcării jucătorului, animațiilor și a interacțiunii cu harta.

Metoda update() este apelată în fiecare cadru și actualizează poziția, hitbox-ul și animația jucătorului.

Metoda render(Graphics g) desenează jucătorul pe ecran, utilizând sprite-urile încărcate.

Metoda updateAnimationTick() actualizează animația jucătorului, incrementând indexul animației și verificând dacă trebuie să se schimbe animația.

Metoda setAnimation() setează animația jucătorului în funcție de acțiunea curentă.

Metoda resetAnimationTick() resetează indexul și timpul de animație.

Metoda updatePosition() actualizează poziția jucătorului în funcție de comenzile primite de la tastatură și verifică coliziunile cu harta.

Metoda loadAnimations() încarcă sprite-urile necesare pentru animația jucătorului.

Metoda loadLvData() încarcă datele de hartă, necesare pentru verificarea coliziunilor cu jucătorul.

Metodele resetDirBooleans() și setAttack(boolean attacking) setează starea de atac și direcțiile de mișcare ale jucătorului.

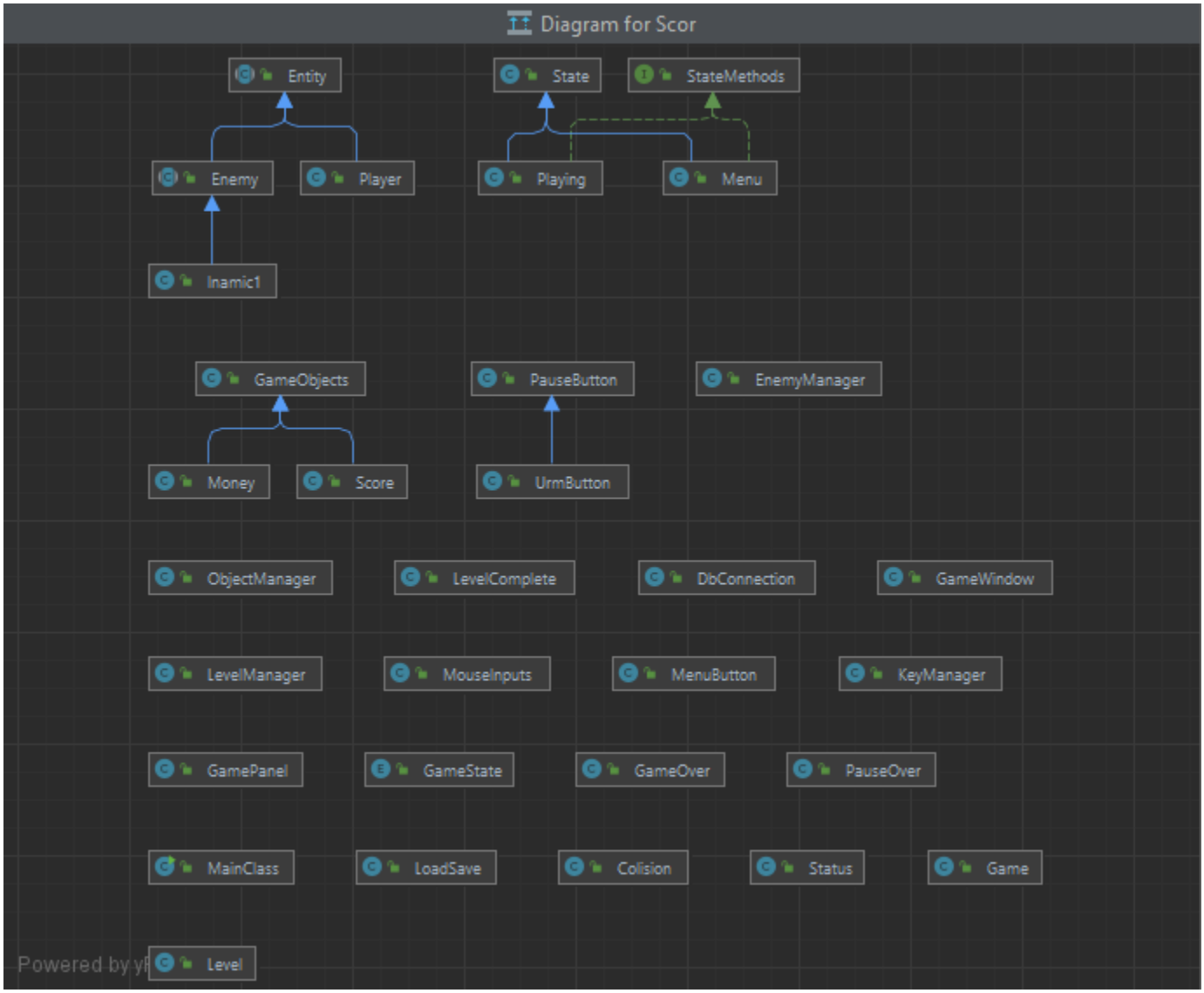
Se vor descrie si urmatoarele clase!

->Etapa 3

Etapa 2:

Arhitectural Design Document

Diagrame UML:



Descriere clase proiect si algoritmi utilizati:

Algoritmul de coliziune:

Clasa Colision este o clasă utilitară care conține diverse metode pentru gestionarea coliziunilor și detecția acestora într-un joc.

Algoritmul se bazează pe o matrice de date a nivelului jocului (lvlData) care reprezintă harta în formă de grid, unde fiecare celulă din matrice poate fi solidă sau liberă.

Iată o descriere a metodelor din clasa Colision:

CanMoveHere(float x, float y, float width, float height, int[][] lvlData): Metoda verifică dacă un obiect poate fi mutat într-o anumită poziție. Aceasta verifică dacă punctele din colțurile dreptunghiului descris de coordonatele (x, y) și dimensiunile (width, height) nu se ciocnesc de obiecte solide din harta (lvlData).

IsSolid(float x, float y, int[][] lvlData): Metoda verifică dacă un punct specificat (x, y) se află pe o celulă solidă din harta (lvlData). Punctul este convertit în coordonate de celule (tile) și se verifică valoarea corespunzătoare din matricea lvlData. Dacă valoarea este mai mică decât 40 și mai mare sau egală cu 0, înseamnă că celula este solidă.

IsTileSolid(int tileX, int tileY, int[][] lvlData): Metoda verifică dacă o celulă specificată prin coordonatele de celulă (tileX, tileY) din matricea lvlData este solidă. Verifică valoarea corespunzătoare din matrice și returnează true dacă este mai mică decât 40 și mai mare sau egală cu 0.

GetEntityPosNextToWall(Rectangle2D.Float hitBox, float xspeed): Metoda returnează poziția pe axa X a entității după ce a întâlnit un perete. Se calculează poziția pe axa X a entității și offset-ul pentru a se plasa în mod corespunzător lângă perete, luând în considerare viteza (xspeed) și dimensiunea entității (hitBox).

GetEntityPosUnderRoofOrAboveFloor(Rectangle2D.Float hitBox, float airSpeed): Metoda returnează poziția pe axa Y a entității după ce a întâlnit un acoperiș sau un podea. Se calculează poziția pe axa Y a entității și offset-ul pentru a se plasa în mod corespunzător sub acoperiș sau deasupra podelei, luând în considerare viteza în aer (airSpeed) și dimensiunea entității (hitBox).

IsEntityOnFloor(Rectangle2D.Float hitBox, int[][] lvlData): Metoda verifică dacă o entitate se află pe podea. Verifică dacă nu există obiecte solide sub dreptunghiul de coliziune

Algoritmul pentru obiecte:

Clasa ObjectManager este responsabilă de gestionarea obiectelor din joc și de controlul modului în care acestea apar și interacționează cu jucătorul.

Algoritmul de creare a obiectelor și de afișare în joc în clasa ObjectManager are următoarele etape:

Încărcarea imaginilor necesare pentru obiecte din fișiere. Imaginile sunt împărțite în mai multe sprite-uri pentru a permite animații și varietate în aspectul obiectelor. În acest scop, se folosesc metodele loadImgs() și LoadSave.GetSpriteAtlas() pentru a obține imaginile necesare pentru monede și puncte.

Crearea și inițializarea obiectelor necesare. În constructorul clasei ObjectManager, se creează și se initializează obiectele de tip Score și se adaugă într-o listă. De asemenea, se inițializează lista money în care vor fi stocate obiectele de tip Money care pot fi colectate în joc. Aceste obiecte sunt preluate de la nivelul curent primit ca argument în metoda loadObjects().

Verificarea coliziunii jucătorului cu obiectele de tip Money. În metoda checkObjectTouched(), se verifică dacă hitbox-ul jucătorului se intersectează cu hitbox-ul unei monede active din lista money. În cazul în care există o coliziune, se apelează metoda applyEffectToPlayer() pentru a aplica efectul monedei asupra jucătorului.

Aplicarea efectului obiectelor asupra jucătorului. În metoda applyEffectToPlayer(), în funcție de tipul monedei, se modifică scorul jucătorului. Dacă moneda este de tip COIN, se adaugă 15 la scorul jucătorului, iar dacă este de tip STAR, se adaugă 30.

Actualizarea stării obiectelor. În metoda update(), se apelează metoda update() pentru fiecare obiect din lista money. Aceasta permite actualizarea animațiilor sau a altor stări ale obiectelor.

Desenarea obiectelor pe ecran. În metoda draw(), se desenează obiectele de tip

Money și obiectele de tip Score. Pentru fiecare monedă activă din lista money, se desenează imaginea corespunzătoare folosind metoda drawImage(). De asemenea, se desenează scorul jucătorului utilizând imaginile din scoreImages. Acestea sunt alese în funcție de scorul jucătorului și sunt afișate în ordine inversă pentru a păstra ordinea corectă a cifrelor.

Resetarea obiectelor. În metoda resetAllObjects(), se resetează starea tuturor obiectelor de tip Money din lista money. Aceasta înseamnă că toate monedele devin active și disponibile pentru

Algoritmul de deplasare inamici:

Modul în care inamicul se mișcă în joc este gestionat de mai multe metode și variabile din clasele Enemy și Inamic1. Voi detalia pas cu pas modul în care inamicul se mișcă folosind codul dat.

Inițializarea și configurarea inamicului:

La crearea unui obiect Inamic1 se inițializează variabilele și se stabilește dimensiunea hitbox-ului și a cutiei de atac a inamicului.

Constructorul clasei Inamic1 primește coordonatele de poziție (x, y) și inițializează hitbox-ul și cutia de atac cu aceste valori.

Actualizarea mișcării și stării inamicului:

Metoda update(int[][] lvlData, Player player) este responsabilă de actualizarea mișcării și stării inamicului în funcție de datele nivelului și poziția jucătorului.

Inițial, se verifică dacă este prima actualizare a inamicului (variabila firstUpdate) și dacă inamicul se află în aer (inAir).

Dacă inamicul se află în aer, se actualizează poziția acestuia pe verticală folosind viteza de cădere (fallSpeed) și gravitația (gravity).

În caz contrar, se verifică starea curentă a inamicului (enemyState) și se efectuează

acțiunile corespunzătoare:

Dacă starea este IDLE (inamicul este în repaus), starea este schimbată în WALK (inamicul începe să meargă).

În starea WALK, inamicul poate să se miște pe orizontală verificând coliziunile cu pereții și podeaua și schimbând direcția de mers dacă întâlnește o coliziune.

După actualizarea mișcării, se verifică starea inamicului pentru alte acțiuni specifice, cum ar fi atacul (ATTACK) sau rănirea (HURT).

Actualizarea animației:

Metoda updateAnimationTick() este responsabilă de actualizarea indexului de animație al inamicului în funcție de viteza de animație (animationSpeed).

Se incrementează animationTick și se verifică dacă a depășit viteza de animație.

Dacă animationIndex depășește numărul de imagini de animație disponibile pentru starea și tipul de inamic, se revine la prima imagine și se gestionează schimbarea stării inamicului în cazul în care este necesar.

Schimbarea direcției de mers:

Metoda changeDir() este folosită pentru a schimba direcția de mers a inamicului atunci când întâlnește o coliziune.

Dacă inamicul se mișcă în dreapta (right) și întâlnește o coliziune, direcția de mers devine left.

Dacă inamicul se mișcă în stânga (left) și întâlnește o coliziune, direcția de mers devine right.

Aceasta este o descriere detaliată a modului în care inamicul se mișcă în joc, conform codului furnizat. Principalele acțiuni sunt gestionate în metoda update(int[][] lvlData, Player player), care actualizează poziția inamicului și starea acestuia în funcție de datele nivelului și interacțiunea cu jucătorul.

Bibliografie: Pentru redactarea acestei teze s-au utilizat următoarele resursede inspirație:

Inspirație alegere tipologie de game: https://www.friv.com/new.html

Creare shițe: https://www.canva.com/design/DAFdkdQs2Io/25Q-hFVrwbMUZpM3tzQXJg/edit

https://www.photopea.com/

Sprite: https://craftpix.net/product/tiny-monsters-pixel-art-pack/

https://craftpix.net/product/fantasy-platformer-game-kit-pixel-art/