Postolache Mălina
Grupa 1208B

Go Zglobi

1. Proiectarea contextului

- *Tipul și tematica jocului:*

Jocul "Go Zglobi" este unul de acțiune, iar stilul acestuia este single player.

- *Povestea jocului:*

Jocul pune în centru un pui de pinguin, Zglobi, care locuiește în Antartica. Fiind în căutare de hrană, acesta s-a rătăcit de familia sa. Scopul lui Zglobi este de a ajunge în siguranță acasă, la colonia sa, în timp ce înfrânge toți inamicii întâlniți pe drum. Acest lucru este posibil doar prin finalizarea nivelelor și câștigarea jocului.

Fiind neinițiat în ceea ce privește autoapărarea și supraviețuirea în cele mai grele situații ale vieții, pinguinul are de parcurs un traseu inițiatic. În primă fază, Zglobi crede că supraviețuirea se rezumă doar la procurarea hrănii, a peștișorilor, însă este mai mult de atât. Pinguinul nu știe ce înseamă răul, că nu toate viețuitoarelor din traseul pe care îl are de străbătut sunt prietenoase.

Pentru prima dată în viață intră în contact cu răul, care poartă numele Snow, ce este un monstru al zăpezii. Având resurse, reprezentate de un număr egal cu șase vieți, Zglobi nu poate fi învins de acesta de la primul contact, însă îi scade semnificativ din energie(vieți). Pentru a fi înfrâns este nevoie, în mod evident, de șase interacțiunii cu Snow.

Cu cât înaintează mai mult, îi apar în drum repere că este pe drumul cel bun, reprezentate de cufere cu comori. Cu toate că este mai

aproape de casă, epuizarea ajunge la cote maxime, iar traseul devine mai complicat și plin de obstacole. Zglobi are nevoie să colecteze bănuți pentru satisfacerea nevoilor fiziologice, iar în schimbul acestora primește mâncarea lui preferată, peștișorii, pe care îi găsește mereu în fața cufărului magic. Cu cât strânge mai repede toți bănuții, cu atât ajunge mai repede la hrană, cufărul cu comori, ce este portalul spre înaintarea eficientă în drum, respectiv la casa lui.

Dacă Zglobi este cumpătat în alegeri, ambițios și își folosește în mod eficient atât atuurile, cât și resursele, el va ajunge în siguranță la colonia sa.

2. Proiectarea sistemului

- *Reguli principale:*

Jucătorul trebuie să acumuleze trei bănuți și să mănânce peștișori (ce sunt în același număr ca și bănuții), în schimbul acestora, ca să avanseze la nivelul următor. În funcție de nivel și de dificultatea acestuia, ai un anumit număr de inamici și șase vieți, care pot să scadă sau nu. Scopul jocului este de a înfrânge toți inamicii pentru a ajunge Zglobi la colonia sa, ce îl așteaptă acasă .

- *Mecanici de joc:*

Pentru a pune pauză jocului se apasă tasta “P”, iar pentru a seta volumul melodiei de fundal și a sunetelor date de interacțiunea cu diferite obiecte și inamici, se folosește tasta “ESC”. Există și optiunea de a accesa și vedea datele curente referitoare la joc, atât nivelul și viețile curente, cât și numărul total de bănuți, peștișori și nivele din joc, prin apăsarea tastei C.

În situația nefavorabilă, a coliziunii protagonistului cu inamicul, apare un text pe ecran, în care Snow îl avertizează si amenință pe Zgobi să se ferească de el. Pentru a scăpa pentru moment atât de el, cât și de mustrarea acestuia, e nevoie să se apese tasta “ENTER”.

- *Tastele de control:*

Ca și controale, jucătorul are următorul set prestabilit:

Action	Player
Press to move up	Up-arrow
Press to move right	Right-arrow
Press to move left	Left-arrow

3. Proiectarea conținutului

- *Schițarea caracterelor:*

Zglobi este protagonistul jocului. Este un pinguin jucăuș ce dorește să ajungă la colonia din care provine. Are abilitatea de a se apăra de pericole, de a își mări viteza, de a colecta bănuți și de a mânca peștișori..

Snow este inamicul acestui joc, un monstru înfometat și care îi dorește răul lui Zglobi. La o simplă coliziune cu acesta îi transmite diverse mesaje și îi “mănâncă” o viață.

- *Interacțiunile jucătorului:*

Pe parcursul jocului player-ul intră în contact cu poțiuni magice (care îi măresc viteza), peștisori, bănuți și inamici.

4. Proiectarea nivelurilor

Jocul are trei nivele de dificultate, determinate atât de poziționarea bucăților de gheață, cât și de existența și numărul de inamici. Fiecare nivel are propria hartă.

- *Primul nivel:*

Pentru accesarea nivelului următor trebuie colectat un număr de trei bănuți si de trei peștișori, care reprezintă hrana vitală a pinguinuilui. Când avem o coliziune cu un bănuț, contorul crește cu o valoare, iar când intră în coliziune cu un peștișor scade cu o valoare. Nu există posibilitatea trecerii la nivelul superior dacă această condiție nu este satisfăcută. Finalizarea nivelului se efectuează în momentul ajungerii la cufărul magic.

- *Al doilea nivel:*

La acest nivel apar în joc și monștrii (inamicii). Playerul are șase vieți, reprezentate print rei inimi, o inimă “plină” reprezentând doua vieți. Daca se realizează coliziunea cu un inamic, viața playerului scade. Nivelul se finalizează după modelul corespunzător primului nivel.

- *Al treilea nivel:*

La ultimul nivel avem în plus, față de cele anterioare, un număr mult mai mare de inamici și o hartă mult mai compicață si greu de parcurs, cu multe capcane. De asemenea, numărul de vieți este mai mici, Zglobi având doar trei. Nivelul se finalizează în urma colectării a trei bănuți si de trei peștișori și a ajungerii la casă.

5. Proiectarea interfeței cu utilizatorul

- Stările jocului sunt următoarele:

Starea de meniu (Title State)- imediat după ce am dat run la Game, jocul intră implicit în această stare. Prin intermediul ei putem accesa starea pentru New Game, Load Game și Exit.

Starea New Game - în această stare începem un nou joc, de la primul nivel.

Starea Load Game – prin intermediul acesteia continuăm jocul, din moment și nivelul la care am rămas în jocul anterior.

Starea de Exit - înseamnă abandonarea jocului.

Pentru a se face tranziția între stări folosim tastele de up și down.

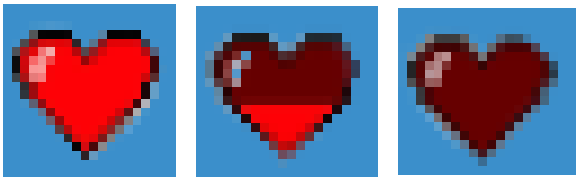
6. Sprites

- *Sprite Sheet-ul jucătorului*

- *Sprite Sheet-ul pentru inamici*

- *pentru peștișori*

- *pentru vieți*



- *pentru bănuți*

- *pentru licoare/poțiunea magică*

- *pentru casă*

- *pentru cufăr*

- *pentru gheță*



➢ *pentru zid*

➢ *pentru apă*

7.Descriere algoritmi utlizați

În clasa **dataStorage** am declarat toate variabilile pe care le vrem salvate, cum ar fi statusul jucătorului, mapa curentă și obiectele de pe mapă.

În clasa **saveLoad** avem doua funcții:

- save (punem într-o varibilă de tip dataStorage informațiile curente, iar pe aceasta o apelăm/folosim în cadrul funcției teleport și checkEvevent, ce fac parte din clasa EventHandler) și

-load(punem în variabila, ce deține informațiile curente, datele din variabila de tip dataStorage, iar pe aceasta o apelăm în KeyHandler, în momentul apăsării butonului de Load Game.

În clasa **Entity** avem datele despre inamic și personajul principal.

În **Player**, clasă ce moștenește Entity, avem mai multe funcții importante pentru joc, dintre care reamintim InteractMonster (ce indică scăderea vieților în momentul interacțiunii cu inamicul) și PickUpObject (care indica schimbările din joc la interacțiunea cu un obiect de pe hartă).

Clasa "Player" este responsabilă de gestionarea obiectelor, inventarului și interacțiunilor jucătorului cu acestea.

Variabila "hasbanuti" indică câți banuți a colectat jucătorul până în acel moment. Variabilele "screenX" și "screenY" reprezintă coordonatele de afișare a jucătorului pe ecran. Există și alte variabile asociate stării și statisticilor jucătorului, cum ar fi nivelul, viața, monedele etc.

Metoda "update()" este apelată în fiecare cadru și verifică tastatura pentru a detecta mișcarea jucătorului. Ea actualizează poziția jucătorului în funcție de direcție și verifică coliziunile cu obiecte, entități și evenimente.

Metoda "pickupObject()" gestionează interacțiunile jucătorului cu obiectele din joc, cum ar fi colectarea banuților sau mâncarea unui peștișor.

Metoda "interactmonster()" gestionează interacțiunile jucătorului cu inamicul și declanșează afișarea dialogului.

Metoda "restorelife()" reîncarcă viața jucătorului la nivelul maxim.

În **Enemy**, clasă ce moștenește Entity, reamintim funcțiile ce setează cum se mișcă inamicul și cum se dispune dialogul în momentul coliziunii.

Deplasarea inamicilor este determinată în metoda setaction(). Acolo se alege o direcție aleatoriu și se setează în variabila direction, în funcție de numărul generat aleatoriu.

-Dacă numărul generat aleatoriu este mai mic sau egal cu 25, direcția va fi up/sus.

-Dacă numărul generat aleatoriu este mai mare decât 25 și mai mic sau egal cu 50, direcția va fi down/jos.

-Dacă numărul generat aleatoriu este mai mare decât 50 și mai mic sau egal cu 75, direcția va fi left/stânga.

-Dacă numărul generat aleatoriu este mai mare decât 75 și mai mic sau egal cu 100, direcția va fi right/dreapta.

Această selecție aleatorie se face la fiecare 150 de iterații, ceea ce corespunde la aproximativ 2 secunde.

În clasa **Asset_Setter** setăm obiectele și monștrii ,în funcție de indexul mapei.

În **CollisionChecker** facem funcții diferite de coliziuni, pentru obiecte, monștrii și tile-uri, iar pe acestea le apelăm în funcția update, din clasa Player.

Metoda "checkTile" primește o entitate ca argument și verifică coliziunile acelei entități cu dalele jocului. Ea calculează coordonatele laturilor dreptunghiului solid al entității în raport cu coordonatele globale ale jocului și apoi le transformă în indici de linii și coloane ale dalelor. Folosind indicii de dale, verifică dacă dalele

respective au coliziune și setează starea de coliziune a entității corespunzătoare.

Metoda "checkObject" primește o entitate și un boolean "player" ca argumente și verifică coliziunile acelei entități cu obiectele din joc. Această metodă parcurge un array de obiecte și verifică dacă obiectul este nul sau nu. Dacă obiectul nu este nul, atunci calculează și actualizează poziția dreptunghiului solid al entității și obiectului și verifică dacă cele două dreptunghiuri se intersectează. Dacă se intersectează și obiectul are coliziune, atunci setează starea de coliziune a entității și, dacă entitatea este jucătorul (player = true), înregistrează indexul obiectului lovit în variabila "index".

Metoda "checkEntity" primește o entitate și o matrice de entități (target) ca argumente și verifică coliziunile între entitatea dată și matricea de entități. Această metodă funcționează similar cu "checkObject", dar în loc să parcurgă un array de obiecte, parcurge o matrice de entități. Verifică dacă entitatea țintă nu este nulă și, în caz afirmativ, calculează și actualizează poziția dreptunghiului solid al entității și entității țintă. Apoi, verifică dacă cele două dreptunghiuri se intersectează și, în cazul în care nu este entitatea însăși, setează starea de coliziune a entității și înregistrează indexul entității lovită în variabila "index".

Metoda "checkPlayer" verifică dacă entitatea specificată(inamicul) și jucătorul se intersectează sau colizionează în joc. Se actualizează poziția solidArea a entității specificate și a jucătorului, luând în considerare poziția lor curentă și dimensiunile zonei solide. Apoi,se simulează noua poziție a entității după mișcare, în funcție de direcția specificată și se verifică dacă zona solidă a entității și zona solidă a jucătorului se intersectează. Dacă cele două dreptunghiuri se ating, înseamnă că entitatea și jucătorul colizionează. Ca urmare, se activează marcajul de coliziune (collisionOn) pentru entitate și se scade o viață din viața jucătorului.

Se resetează valorile poziției solidArea la valorile inițiale, pentru atât entitatea specificată, cât și jucătorul ca să se asigure că zona solidă revine la dimensiunile/ poziția inițială, astfel încât verificarea coliziunilor să se desfășoare corect.

În clasa **EventHandler** se gestionează verificarea și declanșarea evenimentelor în joc, precum teleportarea jucătorului la diferite poziții. Aceasta de ocupă cu manipularea obiectelor de tip EventRect.

Clasa **EventRect** e utilizată pentru a defini acțiuni, în funcție de interacțiunea jucătorullui cu obiectele de acest tip.

KeyHandler e folosită pentru gestionarea evenimentelor tastaturii și controlează acțiunile jucătorului în funcție de starea jocului (playstate, pausestate, dialoguestate, optionstate, gameOverState, titleState).

Clasa **Sound** facilitează încărcarea, redarea și controlul volumului pentru fișierele audio utilizate în joc.

UI are rolul de a gestiona interfața utilizatorului și conține diverse funcționalități pentru afișarea mesajelor, timpului de joc, starea jocului și altele. Clasa conține metode pentru afișarea ecranului de titlu, ecranului de pauză, ecranului de dialog, ecranului de opțiuni, ecranului de încheierea jocului. De asemenea, conține metode pentru afișarea vieților jucătorului și a altor elemente vizuale.

Clasa **SuperObjec**t descrie un obiect într-un joc si conține variabile, metode pentru gestionarea și desenarea obiectelor în cadrul jocului. **OBJ_Banut**, **OBJ_Cufar**, **OBJ_Heart**, **OBJ_House**, **OBJ_Licoare, OBJ_Pestisor** o moștenesc.

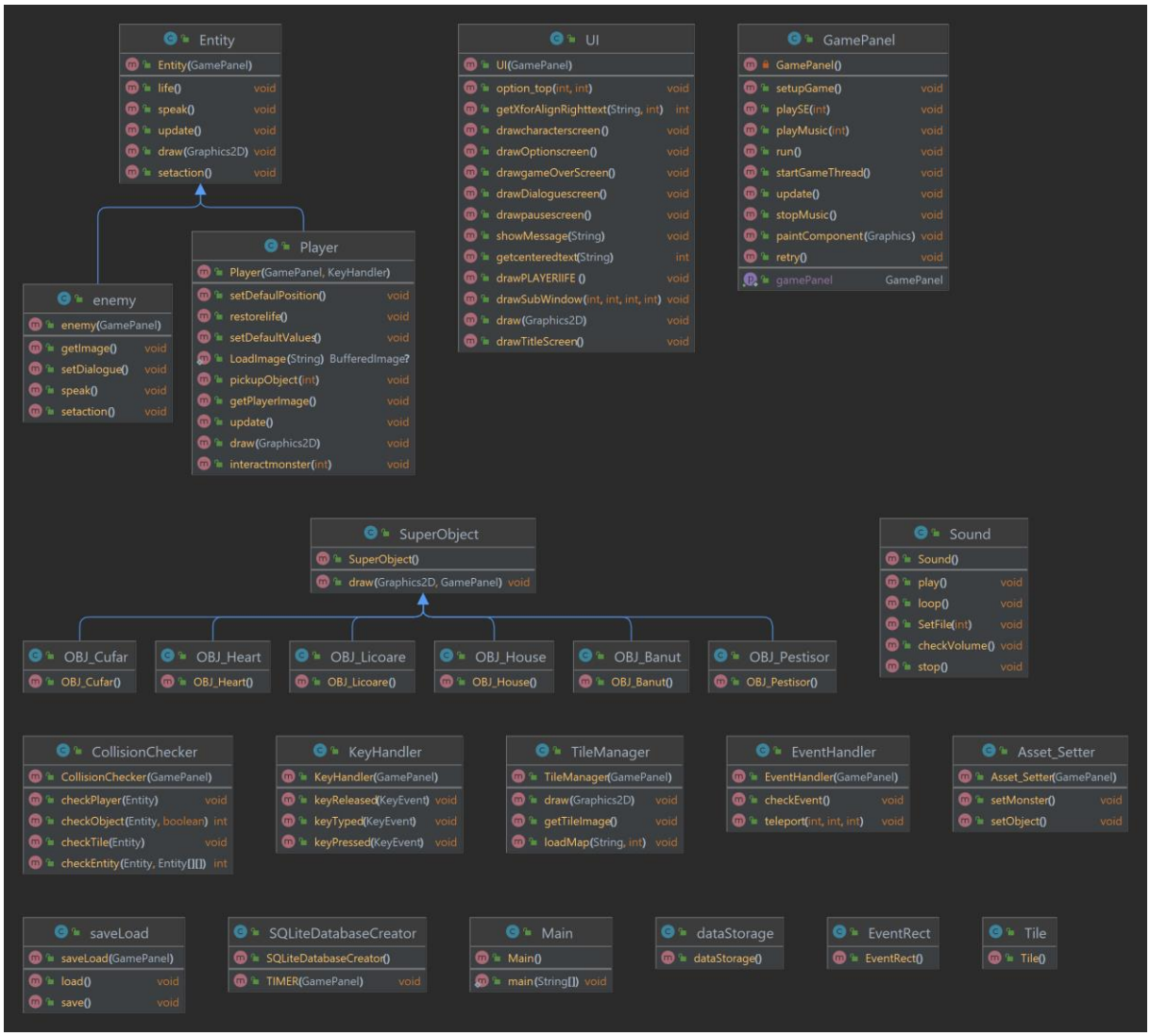
Tile este o clasă simplă pentru gestionarea tipului de tile-uri (în mod evident) și pe care o moștenește **TileManeger**. A doua clasa menționată conține metode pentru încărcarea si desenarea hărții jocului, care este făcută sub forma de matrice (de dimensiunea 50x50) într-un fișier txt.

Clasa **SqliteDataBaseCreator** are o metodă numită Timer, care se ocupă de crearea și gestionarea unei baze de date SQLite. În metoda Timer se stabilește o conexiune la baza de date, iar valoarea specifică a timpului de joc este inserată în coloana TIMER.

Clasa **GamePanel** reprezintă panoul principal al jocului și conține logica principală a acestuia,inclusiv gestionarea elementelor de joc, actualizarea și desenarea acestora, gestionarea stării jocului și a evenimentelor.

Clasa **Main** reprezintă punctul de intrare în aplicație și se ocupă de crearea și configurarea ferestrei jocului, obținerea instanței GamePanel și inițializarea și pornirea jocului

8.Diagrama UML



9. Bibliografie

https://spelunky.fyi/
https://craftpix.net/