Ministerul Educaţiei al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul ISA

**Proiect de an**

**Disciplina : Analiza Proiectarea și Programarea Orientată pe Obiecte**

**Tema:** Elaborarea unui joc 2D in Unity3D

**A efectuat:** st. gr. TI-142 Chirica Alexandru

**A verificat: lector asistent Gavrisco Alexandru**

**Chișinău 2017**

Cuprins:

[Introducere 4](#_Toc484536068)

[1 De ce Unity3D? 5](#_Toc484536069)

[2 Modelarea funcțională a sistemului 6](#_Toc484536070)

[2.1 Modelarea comportamentului sistemului 6](#_Toc484536071)

[2.2 Reprezentarea particularităților temporale în cadrul diagramelor de secvențe/colaborare 7](#_Toc484536072)

[2.3 Colaborare 8](#_Toc484536073)

[3 Proiectarea sistemului 10](#_Toc484536074)

[3.1 Proiectarea logică a sistemului. Diagrama de componente 10](#_Toc484536075)

[3.2 Modelul structural static al sistemului proiectat. Diagrama de stări 10](#_Toc484536076)

[3.3 Modelul structural static al sistemului proiectat. Diagrama de clase 12](#_Toc484536077)

[3.4 Elaborarea diagramei de plasare 18](#_Toc484536078)

[3.5 Elaborarea diagramelor de activități 19](#_Toc484536079)

[4. Elementele de bază a proiectului 20](#_Toc484536080)

[4.1 Scenele și Figurile 20](#_Toc484536081)

[4.2 Animația 22](#_Toc484536082)

[4.3 Coliziunea 24](#_Toc484536083)

[5 Principii OOP 25](#_Toc484536084)

[Concluzii 27](#_Toc484536085)

[Anexa A Coduri sursă 28](#_Toc484536086)

[A.1 BossHealthManager 28](#_Toc484536087)

[A.2 BossPatrol 28](#_Toc484536088)

[A.3 BossWall 29](#_Toc484536089)

[A.4 CameraControll 30](#_Toc484536090)

[A.5 CheckPoint 30](#_Toc484536091)

[A.6 CoinPickup 30](#_Toc484536092)

[A.7 DestroyBlockOnContact 31](#_Toc484536093)

[A.8 DestroyFinishedParticle 31](#_Toc484536094)

[A.9 DestroyObjectOverTime 32](#_Toc484536095)

[A.10 EnemyHealthManager 32](#_Toc484536096)

[A.11 EnemyPatrol 32](#_Toc484536097)

[A.12 EnemyStarController 33](#_Toc484536098)

[A.13 FlyerEnemyMove 34](#_Toc484536099)

[A.13 HealthManager 35](#_Toc484536100)

[A.14 HurtEnemyOnContact 36](#_Toc484536101)

[A.15 HurtPlayerOnContact 37](#_Toc484536102)

[A.16 KillPlayer 37](#_Toc484536103)

[A.17 LadderZone 38](#_Toc484536104)

[A.18 LevelLoader 38](#_Toc484536105)

[A.19 LevelManager 39](#_Toc484536106)

[A.20 LevelSelectManager 40](#_Toc484536107)

[A.21 LifeManager 41](#_Toc484536108)

[A.22 LifePickup 42](#_Toc484536109)

[A.23 MainMenu 42](#_Toc484536110)

[A.24 MovingPlatform 43](#_Toc484536111)

[A.25 NinjaStarController 44](#_Toc484536112)

[A.26 Parallax 45](#_Toc484536113)

[A.27 PauseMenu 46](#_Toc484536114)

[A.28 PlayerController 47](#_Toc484536115)

[A.29 ScoreManager 49](#_Toc484536116)

[A.30 ShootAtPlayerInRange 50](#_Toc484536117)

[Anexa B Screenshot-uri 52](#_Toc484536118)

# Introducere

Jocurile de computer constituie o categorie de aplicații software care este destinată distracției, unele din ele pot fi chiar și educaționale. Încă din copilărie, invățăm să comunicăm, să interacționăm cu lumea ce ne inconjoară prin intermediul jocului. Începînd cu prima etapă de formare a modului de gîndire, întîlnim noi modalități de a ne adopta lanțului de provocări pe care viața le pregătește. În primul rînd, atunci cînd suntem puși în fața unor situații reale,cotidiene, metoda prin care mintea unui copil le ințelege este jocul.

Această practică este deopotrivă o sursă de calm şi relaxare, dar şi o sursă de stimulare a creierului şi a corpului. Jocul este descris adeseori ca modalitatea de petrecere a timpului cea mai vioaie, deşi de multe ori îl luăm în serios şi putem uita în totalitate de el.

Un studiu similar, desfasurat la York University, a descoperit că jocurile pe calculator pregătesc creierul pentru activități diverse ce presupun o coordonare mîini-ochi (ochii văd, mîinile execută).

Un studiu a demonstrat la dislexici îmbunătățirea înțelegerii de lectură în urma sesiunilor de jocuri grele pe acțiune. Motivul, cercetătorii cred, este faptul că jocurile au schimbat în mod constant mediile care necesita concentrare intensă.

În ziua de azi crearea jocurilor 2D-3D este facilitată datorită limbajelor de programare superioare(C++,C#, Java) și motoarelor grafice (Unity3D,Unreal Engine) care ușurează calea spre dezvoltarea acestora.

Un motor grafic este un sistem conceput pentru crearea și dezvoltarea de jocuri video. Există mai multe motoare de joc, care sunt proiectate să funcționeze pe console de jocuri video și calculatoare personale. Funcționalitatea de bază oferită de obicei de un motor grafic include un motor de randare (engleză renderer) pentru grafică 2D sau 3D, un motor de fizică sau de detectare a coliziunilor (și răspunsul la coliziune), sunet, scripting, animație, inteligență artificială, în rețea, streaming, memorie de management, suport de localizare. Dezvoltatorul este predispus să aleagă motorul care îi convine pentru a dezvolta jocuri de înaltă calitate.

Scopul aplicației create este deopotrivă o sursă de calm și relaxare în timpul liber, dar în același timp este o sursă de stimulare a atenției, reacției.

Jocul a fost creat cu ajutorul motorului grafic Unity3D, și limbajului de programare orientat pe obiecte C#.

# 1 De ce Unity3D?

Unity3D - un instrument pentru dezvoltarea aplicațiilor și jocurilor 2D-3D. Aplicațiile create în Unity3D funcționează sub sistemele de operare Windows, OS X, Windows Phone, Android, Apple iOS, Linux, precum și console de jocuri, Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One.

Unity înseamnă o tehnologie unică, convenabilă și extrem de puternică, reprezentînd combinație de caracteristici extraordinare și funcționalitate. Motorul de joc a fost dezvoltat în limbajul C/C ++, și este capabil de a sprijini cod scris în C # sau Javascript.

Unity3D generează o expansiune rapidă pe piața jocurilor la nivel mondial, ocupînd mai mult de 45% din ponderea pe piața mondială, care este de aproximativ de 3 ori mai mare decât cel mai apropiat competitor. așa cum este reprezentat în figura 1.1.

Fara îndoială, motorul Unity este software-ul dominant pentru dezvoltarea de jocuri din lume. Cu Unity3D putem crea orice fel de 2D și 3D-joc. Procesul de creare este ușor, rezultatul este optimizat profund și frumos, și îl putem implementa pe alte platforme cu un singur clic. Serviciul Unity3D integrat este capabil de a accelera procesul de dezvoltare, pentru a optimiza jocul și de a comunica cu publicul țintă.

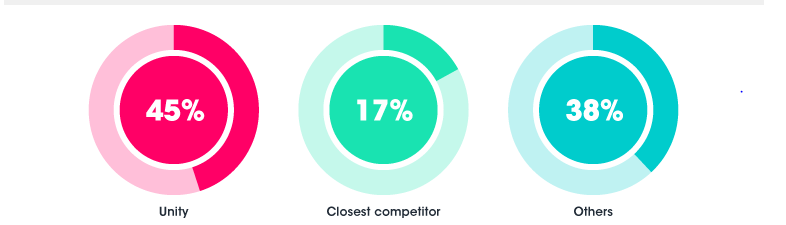


Figura 1.1 **-** Ponderea pe piața mondială a motoarelor de crearea jocurilor

# 2 Modelarea funcțională a sistemului

Scopul realizării acestei lucrări este de a implementa un mini joc 2D din categoria *Action/Arcade*  unde va fi prezent un personaj care va trebui să parcurgă din punctul **A** în punctul **B** evitând un șir de obstacole totodată colectând elemente colectabile care oferă personajului diferite caracteristice. Jocul va dispune de 4 nivele , fiecare nivel se deosebește unul de celălalt .

Ca obiective principale se pot menționa obținerea aptitudinilor și cunoștințelor necesare pentru realizarea sarcinii lucrării, și încadrării in lumea jocurilor.

# 2.1 Modelarea comportamentului sistemului

Diagramele cazurilor de utilizare (use case diagram) descriu destinația funcțională a sistemului sau cu alte cuvinte descrie ceea ce sistemul va executa în procesul său de funcționare. Acestea reprezintă un model inițial conceptual al unui sistem în procesul de proiectare şi exploatare.

În figura 2.1 se definesc principalele funcționalități ale userului odată cu lansarea aplicației. Utilizatorul poate accesa New Game,Load Game,QuitGame,PauseMenu, precum și utilizează TouchControale

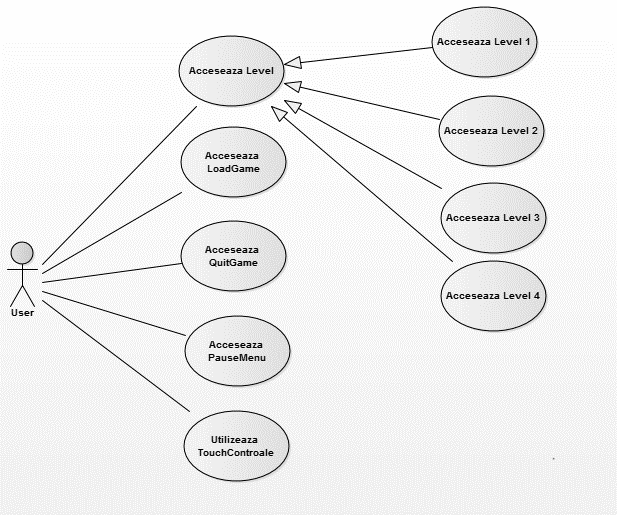


Figura 2.1- Diagrama Use Case, funcționalitățile predispuse utilizatorului

În figura nr 2.2 sunt descrise caracteristicile de bază a erouluui principal(Wisp). Acestea sunt evitarea obstacolelor, colectarea monedelor,efectuare săriturilor,obținerea unui scor,atacarea inamicilor.

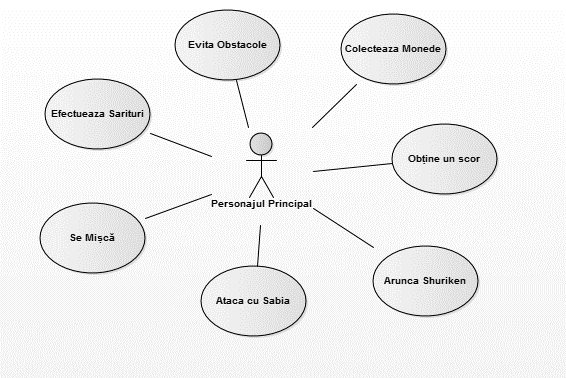


Figura 2.2- Diagrama use case, caracteristicile eroului principal

# 2.2 Reprezentarea particularităților temporale în cadrul diagramelor de secvențe/colaborare

Diagramele de secven ilustreaz interaciunile dintre obiecte sau actori si obiecte din punct de vedere temporal. În figura 2.2.1 este reprezentat metoda de obținere a scorului. Eroul principal întîlnește în preajma sa monede(obiecte) pe care le ridică prin metoda CoinPickUp(), după care acestea sunt destruse, și adaugă puncte la scorul personajului . O altă metodă de a obține scor este distrugerea inamicilor prin metoda HurtOnEnemyContact(), care la fel va apela metoda AddPointToScore() adăugînd puncte la scorul personajului.

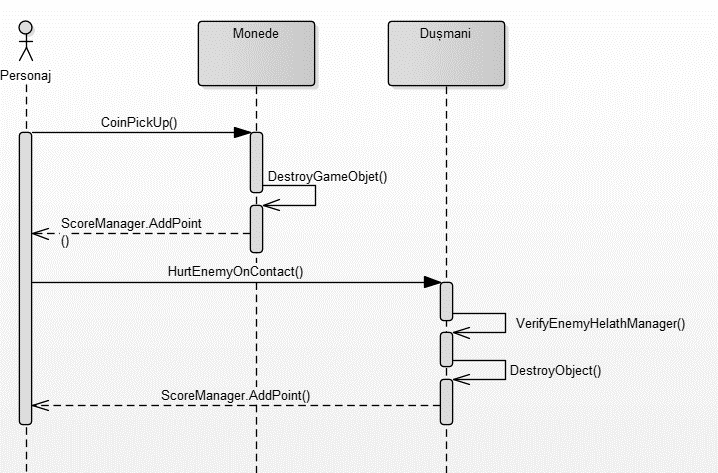


Figura 2.2.1 Adăugarea punctelor la scor

Următorul exemplu va reprezenta mișcarea Personajului doar în cazul în care el se află pe teren precum și accesarea nivelelor la interacțiunea cu ușa de intrare/ieșire. Acest exemplu se observă in figura 2.2.2.

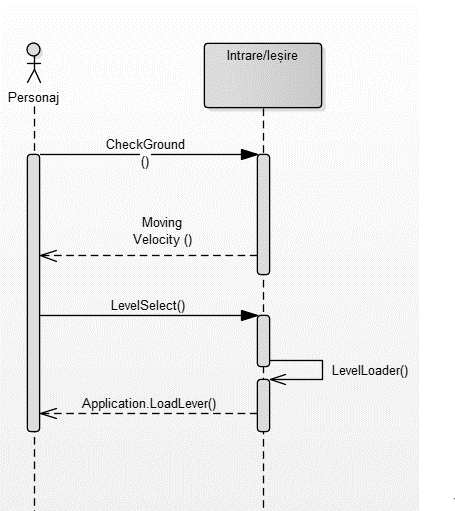


Figura 2.2.2 – Mișcarea si trecerea de la un nivel la altul

# 2.3 Colaborare

Particularitatea principală a diagramei de colaborare constă în posibilitatea de a reprezenta grafic nu numai consecutivitatea colaborării dar şi toate relaţiile structurale între obiecte. Spre deosebire de diagrama de secvenţă în diagrama de colaborare sunt reprezentate relaţiile între obiecte care sunt importante pentru colaborare.

Pentru atingerea unui scop sau pentru realizarea unui serviciu comportamentul unui sistem poate fi descris la nivelul obiectelor care fac schimb de mesaje, având importanță reflectarea legăturilor structurale ale obiectelor aparte. Diagrama de colaborare reflectă un fel de reprezentare statică a unui sistem ca totalitate de obiecte dependente.

În figura următoare este reprezentat colaborarea dintre erou si blocurile cu care interacționeaza și le poate distruge.

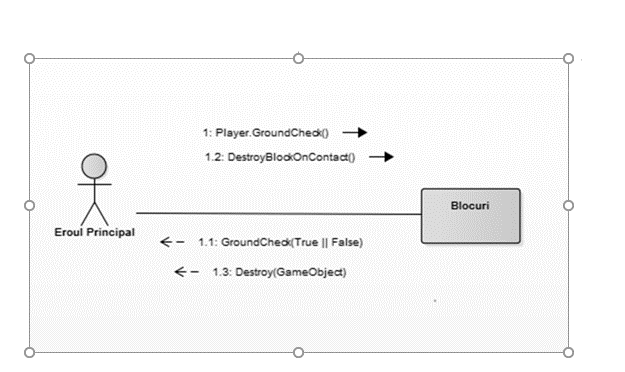


Figura 2.3.1 – Interacțiunea dintre erou și blocuri

Un alt tip de colaborare există între erou și dușmanii săi. Atunci cind eroul întilnește dușmanii , aceștia în mod automat încearcă să-l atace, la interacțiunea cu armele dușmanilor eroul pierde din scorul vieții, la rîndul său la interacțiune cu dușmanii , eroul este aruncat în spate. La atingerea inamicilor cu arma eroului aceștia sunt distruși, adăugînd puncte la scorul eroului(figura 2.3.4) în caz contrar eroul se dizolva în mai multe particule și joaca e finisată .

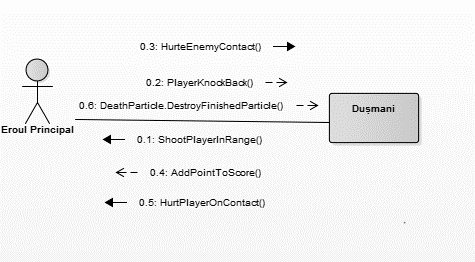


Figura 2.3.4 – Interacțiune erou-dușmani

# 3 Proiectarea sistemului

# 3.1 Proiectarea logică a sistemului. Diagrama de componente

Diagramele de componente sunt utile pentru a reprezenta structura fizică a componentelor sistemului și interdependența acestor componente. Scopul acestei diagrame este de a descrie componentele utilizate pentru oferirea gamei de funcționalități, la fel și vizualizarea componentelor fizice ale sistemului.

Pentru realizarea funcționalităților de bază, aplicația interacționează cu o serie de componente care sunt reprezentate în figura 3.1.1. O particularitate importantă în cadrul aplicației este rularea acesteia, care nu va avea loc fără interacțiunea cu fișierile de tip script , cu obiectele care le-am creat și cu care userul care va utilzia aplicația va putea manipula, precum și scenele pentru trecerea de la un nivel la altul.

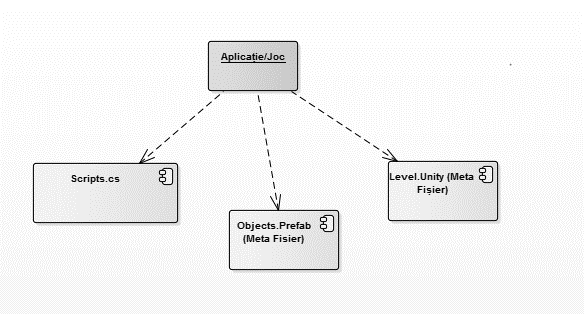


Figura 3.1.1- Diagrama de componente, principalele componente ale sistemului

# 3.2 Modelul structural static al sistemului proiectat. Diagrama de stări

O diagrama de stări modelează viața unui obiect prin stările sale și schimbările de stare care au loc pe parcursul vieții. Schimbările de stare sunt determinate de evenimente. O diagramă de stări reprezintă un automat cu stări finite. Diagramele de interacțiune modelează interacțiunile dintre obiecte. Diagramele de stări modelează efectul acestor interacțiuni asupra stării interne a fiecărui obiect.În figura 3.2.1 este reprezentată digrama de stări a monedelor întilnite în joc . Monedele sunt obiecte ce influențează la scorul personajului atunci cînd acestea interacționează. Deci Monedele vor fi prezente atîta timp cît personajul nu a interacționat cu ele

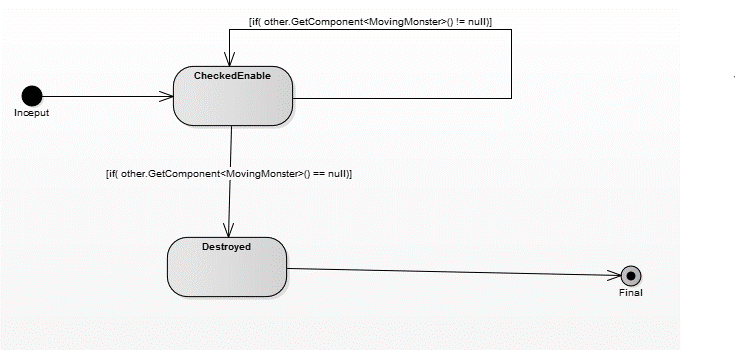


Figura 3.2.1 –Diagrama de stări a monedelor

Pe lîngă monede perosnajul mai poate obține scor și distrugînd dușmanii. În figura 3.2.2 este reprezentată diagrama de stări a dușmanilor,care se află în stare de miscare atîta timp cît în radiusul lor nu este vazut personajul,apoi trec în starea ofensivă și sunt distruși atunci cind viața este egală cu 0.

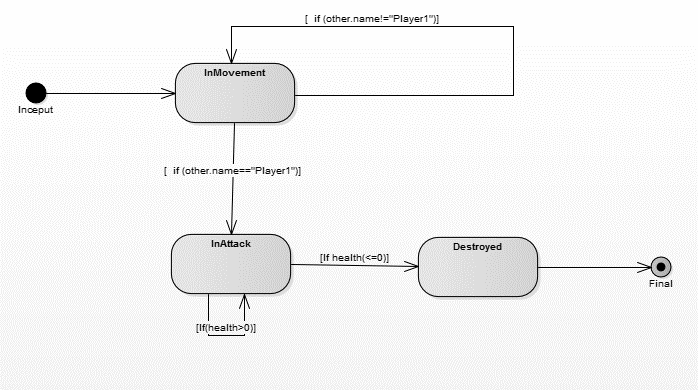


Figura 3.2.2 –Diagrama de stări a dușmanilor

# 3.3 Modelul structural static al sistemului proiectat. Diagrama de clase

În diagrama de clase din figurile 3.2.5 se descrie structura ierarhică a claselor și moștenirea proprietăților și comportamentul claselor părinte, având proprietățile și comportamentul său propriu, pe care nu le au clasa părinte. Clasa **PlayerController**  are un rol esențial în cadrul acestei ierarhii, cu ajutorul ei ,utilizatorul poate controla eroul principal și îl pune în acțiune. Mai jos sunt reprezentate cele mai importante acțiuni realizate de către eroul principal.

if(Input.GetKeyDown(KeyCode.Space) && grounded)

{

// GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D>().velocity.x, jumpHeight);

Jump();

}

if (Input.GetKeyDown(KeyCode.Space) && !doubleJumped && ! grounded)

{

//GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D>().velocity.x, jumpHeight);

Jump();

doubleJumped = true;

}

moveVelocity = 0f;

moveVelocity = moveSpeed \* Input.GetAxisRaw("Horizontal");

if (Input.GetKey(KeyCode.D))

{

// GetComponent<Rigidbody2D>().velocity = new Vector2(moveSpeed,GetComponent<Rigidbody2D>().velocity.y);

moveVelocity = moveSpeed;

}

if (Input.GetKey(KeyCode.A))

{ //GetComponent<Rigidbody2D>().velocity = new Vector2(-moveSpeed, GetComponent<Rigidbody2D>().velocity.y);

moveVelocity = -moveSpeed;

În figura 3.3.1 și 3.3.2 sunt reprezentate trecerea personajului dintr-o poziție în alta,săritura și mișcare.



Figura 3.3.1- Săritura în înălțime a eroului principal



Figura 3.3.2- Mișcarea eroului

În timpul mișcărilor,personajul se poate pierde din limitele ecranului, pentru astfel de cazuri este creată clasa **CameraControl**l care îl va urmări pe erou pînă la fine , apelînd funcția *FindObjectOfType()*, și *transform.position()* care va schimba poziția camerei pe axa X și Y.

void Start () {

player = FindObjectOfType<MovingMonster>();

isFollowing = true;

}

void Update () {

if(isFollowing)

{ transform.position= new Vector3(player.transform.position.x+xoffset,

player.transform.position.y+yOffset, transform.position.z);

}}

În cazul aparițiilor obstacolelor în timpul jocului, eroul are nevoie de implementarea anumitor acțiuni, precum ar fi aruncarea shurikenilor sau lupta cu sabia .Deci a fost creată clasa, **NinjaStarController** și **HurtEnemyOnContact** cu ajutorul caruia player-ul va putea trece peste orice obstacol.În codul de mai jos este reprezentată clasa **HurtOnEnemyContact** și principalele atribute și metode ale acesteia.

public class HurtEnemyOnContact : MonoBehaviour {

public int damageToGive;

public float bounceEnemy;

private Rigidbody2D myrigidbody2D;

void Start () {

myrigidbody2D = transform.parent.GetComponent<Rigidbody2D>();

}

void OnTriggerEnter2D(Collider2D other)

{if (other.tag =="Enemy")

{ other.GetComponent<EnemyHelathManager>().giveDamage(damageToGive);

myrigidbody2D.velocity = new Vector2(myrigidbody2D.velocity.x, bounceEnemy);

}

Efectul cel mai mare asupra pășirii peste obstacole o joacă atributul *damageToGive ,* cu ajutorul lui stabilim forța în valori numerice cu care eroul atacă, și apelînd metoda *giveDamage*() se transmite valoarea numerică către obiectul cu Tag „ Enemy”.

Un rol important în mișcarea personajului îl joaca numărul de vieți, deoarece nu putem mișca ceea ce nu există, deci mișcarea personajului depinde si de numărul de vieți de care dispune personajul. Clasa **healthManager** împreuna cu atributul *playerHealth* informează numărul de vieți atașați personajului.

public class HealthManager : MonoBehaviour {

public static int playerHealth;

Text text;

public bool isDead;

public int maxPlayerHealth;

private LevelManager levelManager;

void Start(){

playerHealth = maxPlayerHealth;

levelManager = FindObjectOfType<LevelManager>();

isDead = false;

}

Așa cum se observă la inițierea jocului Player-ul are viața maximală ,ceea ce se observă și în codul de mai sus atribuind atributului playerHealth, maxPlayerHealth.

În figura 3.3.3 vor fi afișate toate legăturile dintre principalele clase ale proiectului dat.

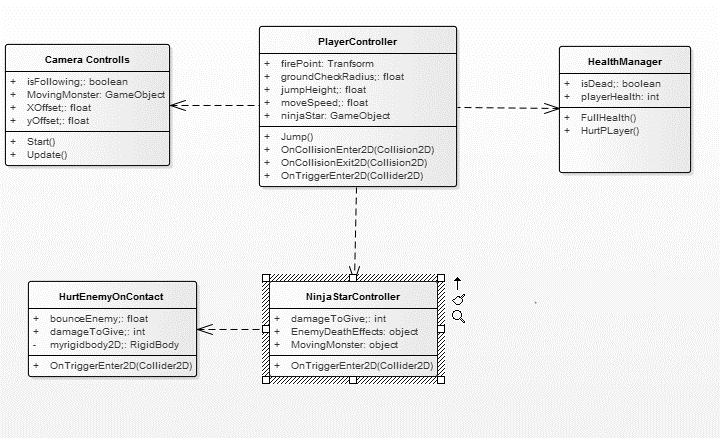


Figura 3.3.3– Relațiile PlayerController și alte clase

Cum și s-a menționat mai sus ,scopul jocului este de a obține un anumit scor și de a trece prin toate nivelele. Obținerea scorului poate avea loc prin strîngerea monedelor și distrugerea inamicilor. Clasa Score Manager va genera pe ecran scorul obținut de către erou. Deci clasa **ScoreManager** depinde în mare parte de clasa **CoinPickUp** (strîngerea monedelor) și clasa **HurtEnemyOnContact** .

Așa cum obținem scor , așa și îl putem pierde atunci cînd eroul este doborît, deci clasa **HealthManager** și **LevelManager** ne va informa dacă pierdem sau nu din scor. Mai jos este afișată clasa **CoinPickUp**

public class CoinPickUp : MonoBehaviour {

public int pointsToAdd;

void OnTriggerEnter2D (Collider2D other)

{ if (other.GetComponent<MovingMonster>() == null)

ScoreManager.AddPoints(pointsToAdd)}}

Așa cum se observă metoda OnTriggerEnter2D apelează metoda AddPoints al clasei ScoreManager pentru a transmite valoarea numerica obținută în urma interacționării cu monedele. Scorul inițial este 0 așa cum este reprezentat în figura 3.3.4 .



Figura 3.3.4 – Scorul inițial

În figura de mai sus ,se observă prezența celor 3 monede , care vor influența asupra scorului eroului în timpul interacțiunii iar în figura 3.3.5 , mărirea scorului în timpul strîngerii monedelor.



Figura 3.3.5 – Obținerea scorului cu ajutorul monedelor

Clasa **LevelManager** va genera puncte de penalizare în caz cînd eroul este distrus,și aceste puncte la rîndul său depind în mare parte de atributul *playerHealth* din clasa **HealthManager**.

public class HealthManager : MonoBehaviour{

void Update()

{if (playerHealth <= 0 && !isDead)

{ playerHealth = 0;

levelManager.RespawnPlayer();

isDead = true; }

text.text = "" + playerHealth;

}

În codul de mai sus , este menționat faptul că atunci cînd viața personajului este mai mică sau egală cu 0 , este apelată metoda RespawnPlayer al clasei **LevelManager**. Ulterior clasa **LevelManager** apelează metoda AddScore a clasei **ScoreManager** adăugînd ca parametru puncte de penalizare,și-l va întoarce în viață la poziția **CheckPoint** pe Wisp.

public class LevelManager {

public GameObject currentCheckPoint;

private MovingMonster player;

public HealthManager healthManager;

void Start () {

player = FindObjectOfType<MovingMonster>();

camera = FindObjectOfType<CameraControls>();

healthManager = FindObjectOfType<HealthManager>();

}

public IEnumerator RespawnPlayerCo()

{ Instantiate(deathParticle, player.transform.position, player.transform.rotation);

player.enabled=false;

player.GetComponent<Renderer>().enabled = false;

camera.isFollowing = false;

ScoreManager.AddPoints(-pointPenaltyOnDeath);}

În figura 3.3.6 este reprezentată diagrama de clasă a obținerii scorului .

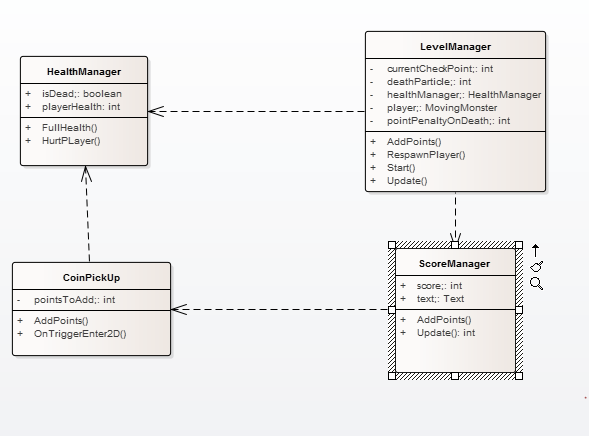


Figura 3.3.6 – Relațiile LevelManger și alte clase

# 3.4 Elaborarea diagramei de plasare

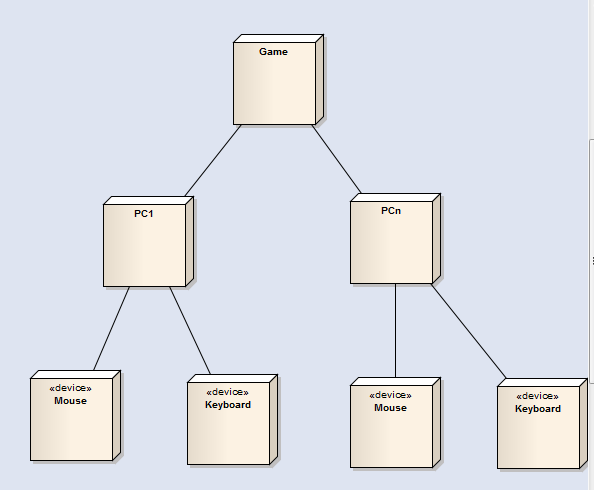


Figura 3.3.7 – Diagrama de amplasare a sistemului

În figura 3.3.7 este rerezentată diagrama de desfașurare în care se descrie accesul la server Joc pe nivele. Nivelul cel mai de jos reprezinta dispozitivele cu ajutorul carora se introduce infosrmația la nivelul 2 care reprezintă niște procesoare care au capacitatea de calcul, cum ar fi PC

# 3.5 Elaborarea diagramelor de activități

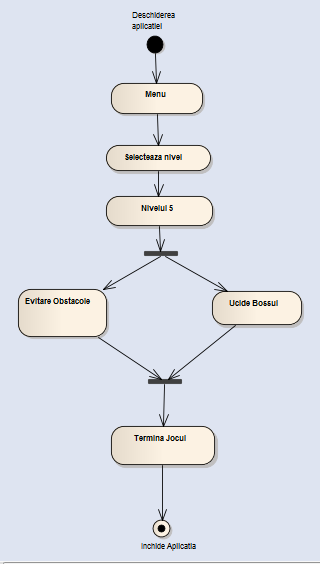


Figura 3.3.8 – Diagrama de activitate a nivelului 5

Pentru a putea Termina nivelul 5 jucatorul trebuie sa execute niste activitati care sunt reprezentate in Figura 3.3.8 , odata cu alegerea nivelului 5 jucatorul nimereste in camera boss-ului din care nu poate iesi pana nu evita obstacolele din acea camera si pana ce nu ucide Boss-ul , dupa ce Bossul este ucis este deschis accesul la usa unde jucatorul Finalizeaza jocul

# 4. Elementele de bază a proiectului

Cum și a fost menționat mai sus scopul proiectului dat , este elaborarea unui joc 2D. Astfel pentru a-l aduce la bun sfîrșit, îl inițializăm cu toate elementele necesare.

Din elementele necesare fac parte :

* scena și figurile;
* animația;
* coliziunea;
* scripting .

# 4.1 Scenele și Figurile

Scenele conțin obiecte ale jocului. Ele pot fi folosite pentru a crea un meniu principal, la nivel individual, și orice altceva. Fiecare fișier Scene este unic. În fiecare scenă, plasăm mediile, obstacole, si decorațiuni, în esență, proiectarăm și construim jocul pe bucăți. Proiectul în cauză conține 5 scene : meniul și cele 4 nivele. Meniul este format din 3 butoane : NewGame,LoadGame,QuitGame. În figura 4.1.1 este reprezentată o scenă a proiectului nostru.

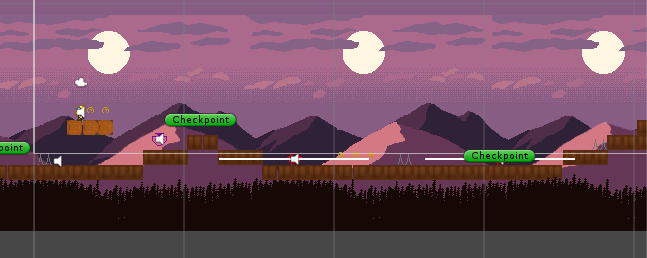


Figura 4.1.1 –Exemplu de scenă

Este foarte important de reținut , că scenele sunt unice deci nu interacționeaza automat între ele, atunci cînd sărim de la o scena la alta , nu uităm de 2 parametri foarte importanți , numele Scenei , și arhivarea acestora în submeniul Build Scene , unde fiecărei scene îi vom atribui un număr de prioritate,pentru a face legătură între ele și ca Unity Editorul să cunoască care e scena principală. În figura 4.1 sunt afișate prioritățile fiecărei scene din Submeniul Build.

Obiectele sunt elementele pe care le creeăm pentru ca user-ul să interacționeze cu ele în cadrul jocului. Eroul principal (Wisp) este un obiect tip imagine,format (png) , extras cu ajutorul Sprite Editor, un editor de transformare a imaginii în obiect Unity.Pentru a extrage imaginea și a o serializa utilizăm elemental de tăiere a imaginii , și atribuim nume obiectului nou format , în cazul nostru Player1 așa cum este afișat în figura 4.1.2.

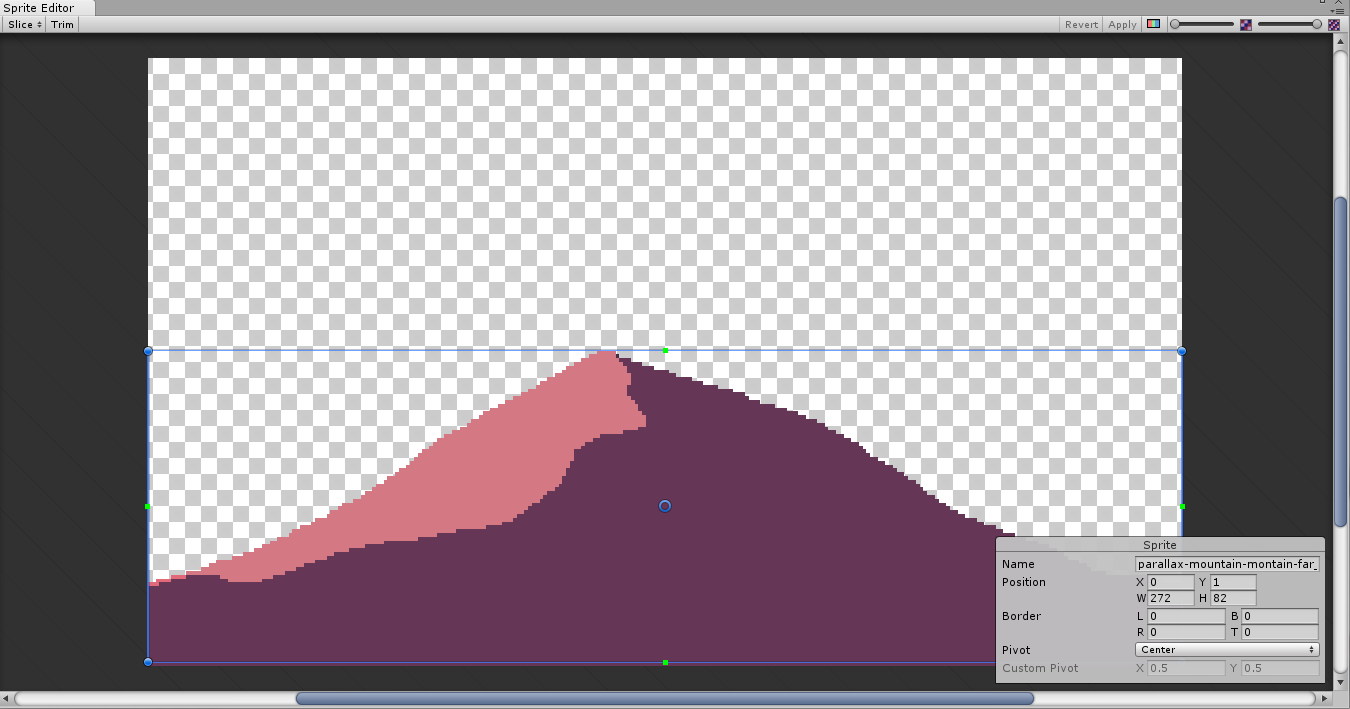


Figura 4.1.2 – Sprite Editor

Alte elemente utilizate în cadrul jocului sunt :

* inamicii(Enemies) sunt acelea obiecte care vor crea piedici eroului Wisp;
* Spikes , elemente capcană;
* blocuri , terenul pe care se va deplasa Wisp;
* arme ,obiectele cu care va ataca Wisp compuse din NinjaStar și Sword;
* Boss ,principalul dușman a lui Wisp.

În figura 4.1.3 de mai jos vor fi afișate toate elementele secundare utilizate în cadrul jocului.

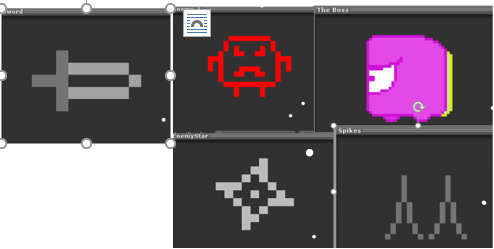


Figura 4.1.3 – Elementele secundare ale jocului

# 4.2 Animația

Testând joaca vom observa că accesînd tasta „L”,eroul începe a ataca cu sabia, apoi sabia iar dispare, la fel și dușmanii , în timpul mișcărilor din dreapta în stînga își flutură mîinile și aruncă shuriken-uri. Toate acestea se datorează animației. A anima ceva inseamnă, literal, a-l aduce la viață, a-i da viață. Animația, în accepțiunea uzuală, cuprinde toate modificările care au un efect vizual. Efectele vizuale pot fi de diferite feluri: poziții care variază în timp (motion dynamics - dinamica mișcării), modificări ale formei, culorilor, transparenței, structurii și a texturii suprafețelor unui obiect (update dynamics - dinamica infațișării)

Motorul Unity3D , dispune de editorul Animație, unde am folosit un sprite de imagini la un anumit interval de timp . Unity va crea un fișier Anim , în care va înregistra schimbările efectuate.În figura 4.2.1 și 4.2.2 putem observa că la un anumit interval de timp și anume într-o secundă , eroul principal va realiza 4 acțiuni : ascunde sabia, afișează sabia, menține sabia întro poziție și ascunde iar sabia .

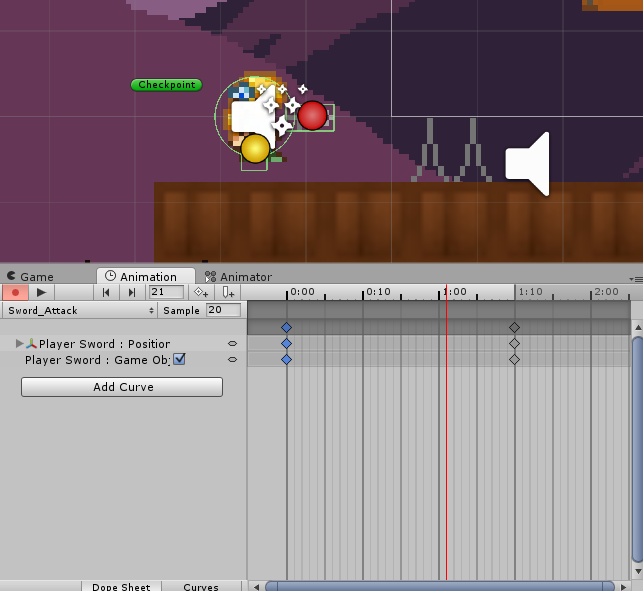


Figura 4.2.1 – Menținerea sabiei într-o poziție

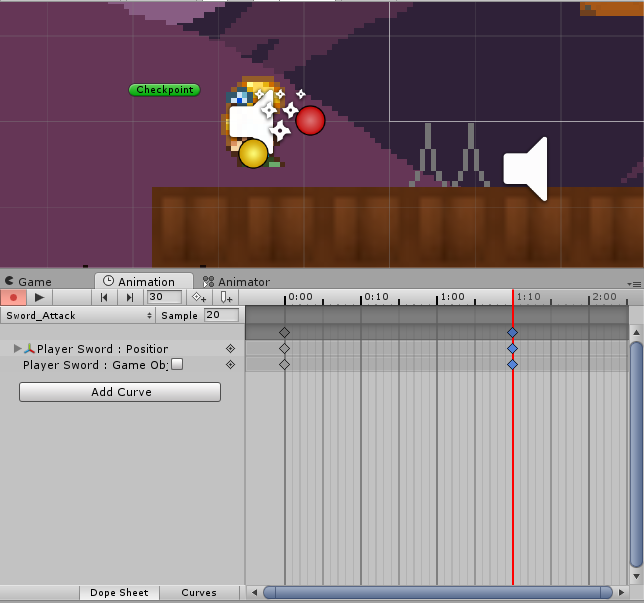


Figura 4.2.2 – Ascunderea sabiei

Pe lîngă înregistrarea efectuată ,Unity mai dispune și de Animator, o interfață de control al mecanismului de animație , reprezentat în figura 4.2.3.

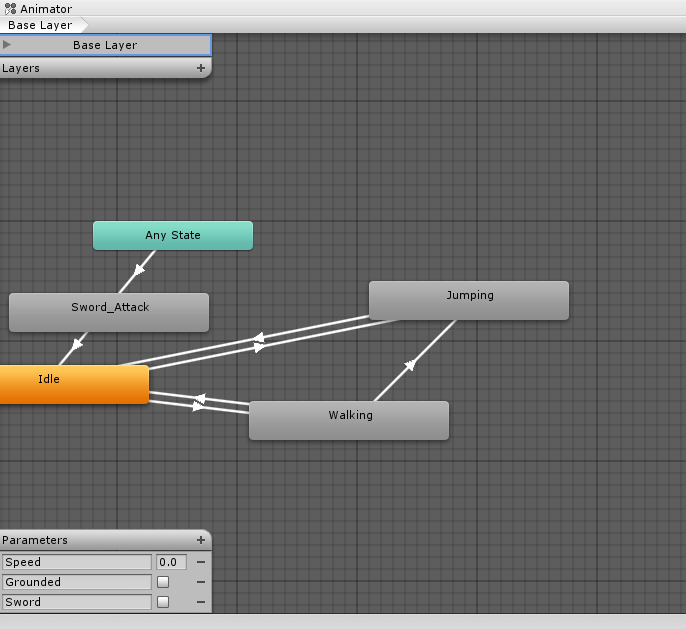


Figura 4.2.3– Animator

Așa cum observăm avem 4 dreptunghiuri , unul legat de celălalt prin referințe. Dreptunghiul Entry raspunde pentru animația la începutul pornirii jocului. Idle1 , animație în timpul mișcărilor, SwordAnimation1 , animația ce corespunde afișării și ascunderii sabiei eroului principal , AnyState animații opționale.

Dar asta nu e de ajuns pentru a porni animațiile să lucreze , Unity3D mai cere și atașarea componentelor animației personajului , precum și anunțarea pornirii acesteia prin setarea unei variabile isBool prin True sau False atunci cînd tastăm sau nu tasta “L”, așa cum vedem în codul de mai jos .

void Start(){

anim = GetComponent<Animator>(); }

if (anim.GetBool("Sword1")) {

anim.SetBool("Sword1", false); }

if (Input.GetKey(KeyCode.L))

{ anim.SetBool("Sword1", true); }

# 4.3 Coliziunea

Coliziunea reprezintă întîlnirea particulelor sau corpurilor în care fiecare exercită o forță asupra celuilalt, provocând schimbul de energie sau impuls . Unity3D dispune de obiecte *Collider* de diferite forme (*Box Collider, Circle Collider,* etc) *.* Cu o poziționare atentă și dimensionare, collider-urile combinate pot ocupa obiectul în întregime.

Atunci când au loc coliziuni, motorul numește funcții cu nume specifice cu privire la orice script atașat la obiectele implicate. Avem posibilitatea să plasăm orice cod pe care dorim în aceste funcții pentru a răspunde la evenimentul de coliziune. De exemplu, să se distrugă un bloc, atunci cînd interacționăm cu sabia.

La prima actualizare a coliziunii,utilizăm funcția *OnCollisionEnter*. În timpul actualizărilor în care contactul dintre obiecte se menține pe o perioada mai lungă folosim *OnCollisionStay* și în cele din urmă, *OnCollisionExit* indică faptul că un contact a fost rupt.

# 5 Principii OOP

Programarea orientată pe obiect (Programare Orientată Obiectual) este unul din cei mai importanți pași făcuți în evoluția limbajelor de programare spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai natural ființei umane. Astfel unitațile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât modul de lucru al calculatorului. Deși tehnica se numește "Programare Orientată Obiectual", conceptul de bază al ei este Clasa. Clasa, pe lângă faptul că abstractizează foarte mult analiza/sinteza problemei, are proprietatea de generalitate, ea desemnând o mulțime de obiecte care împart o serie de proprietăți.

Programarea orientată pe obiecte se bazează, sau este comusă și din anumite principii.

Principiile OOP :

* abstractizarea ;
* moștenirea ;
* încapsularea ;
* polimorfismul ;
* compoziție ;
* agregare .

În cadrul proiectului nostru am utilizat Încapsularea pentru fiecare clasa declarată. Încapsularea reprezintă ascunderea de informații: Asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv), doar metodele proprii ale obiectului pot accesa starea acestuia. În codul de mai jos sunt declarate atributele clasei **PlayerController** , utilizînd Încapsularea.

public class PlayerController : MonoBehaviour {

public float moveSpeed;

private float moveVelocity;

public float jumpHeight;

public Transform groundCheck; //pentru FixedUpdate

public float groundCheckRadius; //pentru FixedUpdate

public LayerMask whatIsGround; //pentru FixedUpdate

private bool grounded; //pentru FixedUpdate }

În cadrul unui obiect, codul și datele pot fi private sau public. Cînd sunt private, ele sunt vizibile și accesibile doar în interiorul obiectului. În cazul public, celelalte părți ale programului le pot utiliza .

În cazul nostru Unity Editor ne permite să utilizăm elementele publice direct din bara de navigare așa cum vedem în figura 5.1 .

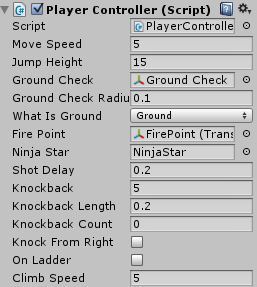


Figura 5.1 – Elementele publice ale clasei.

Pentru a utiliza metodele Unity pentru fiecare clasă aparte, am utilizat principiul Moștenirii , moștenind Clasa **MonoBehaviorScript**, clasa de bază de unde derivă toate scripturile.

Foarte des utilizînd o clasă avem nevoie să interacționăm cu obiectul altei clase, sau să extragem componentele acestuia , deci folosim principiul , relația de compoziție(hasA).

În codul de mai jos voi reprezenta clasa CameraController, în care vom declara un obiect al clasei **PlayerController** , îi vom găsi tipul prin metoda *FindObjectOfType ,* și îl vom utiliza în metoda Update() ca un parametru al funcției *transform.position* .

using UnityEngine;

using System.Collections;

public class CameraController : MonoBehaviour {

public PlayerController player;

public bool isFollowing;

public float xOffset;

public float yOffset;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController> ();

isFollowing = true;

}

void Update () {

if (isFollowing)

transform.position = new Vector3 (player.transform.position.x + xOffset, player.transform.position.y + yOffset, transform.position.z);

}

}

# Concluzii

Realizarea aplicațiilor utilizând principiile programării orientate pe obiecte are o serie de avantaje importante, atîi și pentru lumea jocurilor . Aceste principii permit o proiectare mai facilă . În ceea ce priveşte proiectarea, se facilitează descompunerea problemelor complexe în subprobleme simple, care pot fi uşor modelate cu ajutorul obiectelor (variabilele vor descrie proprietăţile obiectelor modelate şi metodele acţiunile lor). Efectuând acest proiect de curs am studiat un Editor nou, sub numele de Unity3D Editor care m-a ajutat la crearea unui joc simplu 2D. El oferă un șir de funcționalități gata de utilizarea iar dacă acestea nu ne convin , putem implementa funcționalitățile personale. Pe lângă faptul că oferă un șir vast de funcționalități , mai are și o documentație bună și ușoară pentru începătorii în industria dată. În documentația dată se poate de găsit răspunsuri la majoritatea întrebărilor iar dacă acestea nu sunt suficientemai sunt și exemple oferite de către dezvoltatorii acestui editor sau chiar de alți utilizatori care au creat jocuri utilizând acest mecanism.

# Anexa A Coduri sursă

# A.1 BossHealthManager

using UnityEngine;

using System.Collections;

public class BossHealthManager : MonoBehaviour {

public int enemyHealth;

public GameObject deathEffect;

public int pointOnDeath;

public GameObject bossPrefab;

public float minSize;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (enemyHealth <= 0)

{

Instantiate(deathEffect, transform.position, transform.rotation);

ScoreManager.AddPoints(pointOnDeath);

if(transform.localScale.y > minSize)

{

GameObject clone1 = Instantiate(bossPrefab, new Vector3 (transform.position.x + 0.5f , transform.position.y , transform.position.z),transform.rotation) as GameObject;

GameObject clone2 = Instantiate(bossPrefab, new Vector3 (transform.position.x - 0.5f , transform.position.y , transform.position.z),transform.rotation) as GameObject;

clone1.transform.localScale = new Vector3(transform.localScale.y \* 0.5f, transform.localScale.y \* 0.5f , transform.localScale.z);

clone1.GetComponent<BossHealthManager>().enemyHealth = 10;

clone2.transform.localScale = new Vector3(transform.localScale.y \* 0.5f, transform.localScale.y \* 0.5f , transform.localScale.z);

clone2.GetComponent<BossHealthManager>().enemyHealth = 10;

}

Destroy(gameObject);

}

}

public void giveDamage(int damageToGive)

{

enemyHealth -= damageToGive;

audio.Play (); // adaugam efectul de audio

}

}

# A.2 BossPatrol

using UnityEngine;

using System.Collections;

public class BossPatrol : MonoBehaviour {

public float moveSpeed;

public bool moveRight;

public Transform wallCheck; //pentru FixedUpdate

public float wallCheckRadius; //pentru FixedUpdate

public LayerMask whatIsWall; //pentru FixedUpdate

private bool hittingWall; //pentru FixedUpdate

private bool notAtEdge;

public Transform edgeCheck;

private float ySize;

// Use this for initialization

void Start () {

ySize = transform.localScale.y;

}

// Update is called once per frame

void Update () {

hittingWall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius, whatIsWall);

notAtEdge = Physics2D.OverlapCircle(edgeCheck.position, wallCheckRadius, whatIsWall);

if (hittingWall || !notAtEdge)

moveRight = !moveRight;

if (moveRight)

{

transform.localScale = new Vector3 (-ySize,transform.localScale.y,transform.localScale.z);

rigidbody2D.velocity = new Vector2 (moveSpeed, rigidbody2D.velocity.y);

} else {

transform.localScale = new Vector3 (ySize,transform.localScale.y,transform.localScale.z);

rigidbody2D.velocity = new Vector2 (-moveSpeed, rigidbody2D.velocity.y);

}

}

}

# A.3 BossWall

using UnityEngine;

using System.Collections;

public class BossWall : MonoBehaviour {

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if(FindObjectOfType<BossHealthManager>())

{

return;

}

Destroy (gameObject);

}

}

# A.4 CameraControll

using UnityEngine;

using System.Collections;

public class CameraController : MonoBehaviour {

public PlayerController player;

public bool isFollowing;

public float xOffset;

public float yOffset;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController> ();

isFollowing = true;

}

// Update is called once per frame

void Update () {

if (isFollowing)

transform.position = new Vector3 (player.transform.position.x + xOffset, player.transform.position.y + yOffset, transform.position.z);

}

}

# A.5 CheckPoint

using UnityEngine;

using System.Collections;

public class Checkpoint : MonoBehaviour {

public LevelManager levelManager;

// Use this for initialization

void Start () {

levelManager = FindObjectOfType<LevelManager> ();

}

// Update is called once per frame

void Update () {

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

levelManager.currentCheckPoint = gameObject;

Debug.Log("Activated Checkpoint " + transform.position);

}

}

}

# A.6 CoinPickup

using UnityEngine;

using System.Collections;

public class CoinPickup : MonoBehaviour {

public int pointsToAdd;

public AudioSource coinSoundEffect;

void OnTriggerEnter2D (Collider2D other)

{

if (other.GetComponent<PlayerController> () == null)

return;

ScoreManager.AddPoints (pointsToAdd);

coinSoundEffect.Play ();

Destroy (gameObject);

}

}

# A.7 DestroyBlockOnContact

using UnityEngine;

using System.Collections;

public class DestroyBlockOnContact : MonoBehaviour {

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Ground")

{

Destroy (other.gameObject);

}

}

}

# A.8 DestroyFinishedParticle

using UnityEngine;

using System.Collections;

//Distrugem obiectele verzi si rosii Death si Respawn de respawn dupa ceea ce murim , sa optimizam sistemul

public class DestroyFinishedParticle : MonoBehaviour {

private ParticleSystem thisParticleSystem;

// Use this for initialization

void Start () {

thisParticleSystem = GetComponent<ParticleSystem> ();

}

// Update is called once per frame

void Update () {

if (thisParticleSystem.isPlaying)

return;

Destroy (gameObject);

}

void OnBecameInvisible()

{

Destroy (gameObject);

}

}

# A.9 DestroyObjectOverTime

using UnityEngine;

using System.Collections;

//Distruge obiectele create de cutitele ninja ,adica direct cutitele le distruge dupa ce au fost aruncate

public class DestroyObjectOverTime : MonoBehaviour {

public float lifetime;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

lifetime -= Time.deltaTime;

if (lifetime < 0)

{

Destroy(gameObject);

}

}

}

# A.10 EnemyHealthManager

using UnityEngine;

using System.Collections;

public class EnemyHealthManager : MonoBehaviour {

public int enemyHealth;

public GameObject deathEffect;

public int pointOnDeath;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (enemyHealth <= 0)

{

Instantiate(deathEffect, transform.position, transform.rotation);

ScoreManager.AddPoints(pointOnDeath);

Destroy(gameObject);

}

}

public void giveDamage(int damageToGive)

{

enemyHealth -= damageToGive;

audio.Play (); // adaugam efectul de audio

}

}

# A.11 EnemyPatrol

using UnityEngine;

using System.Collections;

public class EnemyPatrol : MonoBehaviour {

public float moveSpeed;

public bool moveRight;

public Transform wallCheck; //pentru FixedUpdate

public float wallCheckRadius; //pentru FixedUpdate

public LayerMask whatIsWall; //pentru FixedUpdate

private bool hittingWall; //pentru FixedUpdate

private bool notAtEdge;

public Transform edgeCheck;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

hittingWall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius, whatIsWall);

notAtEdge = Physics2D.OverlapCircle(edgeCheck.position, wallCheckRadius, whatIsWall);

if (hittingWall || !notAtEdge)

moveRight = !moveRight;

if (moveRight)

{

transform.localScale = new Vector3 (-1f,1f,1f);

rigidbody2D.velocity = new Vector2 (moveSpeed, rigidbody2D.velocity.y);

} else {

transform.localScale = new Vector3 (1f,1f,1f);

rigidbody2D.velocity = new Vector2 (-moveSpeed, rigidbody2D.velocity.y);

}

}

}

# A.12 EnemyStarController

using UnityEngine;

using System.Collections;

public class EnemyStarController : MonoBehaviour {

public float speed;

public PlayerController player;

//public GameObject enemyDeathEffect;

public GameObject impactEffect;

//public int pointsForKill;

public float rotationSpeed; //rotatia la cutite ninja

public int damageToGive;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController>();

if (player.transform.position.x < transform.position.x)

{

speed = -speed;

rotationSpeed = -rotationSpeed; //rotatia la cutitele ninja

}

}

// Update is called once per frame

void Update () {

rigidbody2D.velocity = new Vector2 (speed, rigidbody2D.velocity.y);

rigidbody2D.angularVelocity = rotationSpeed;

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

//Instantiate(enemyDeathEffect, other.transform.position , other.transform.rotation);

//Destroy(other.gameObject);

//ScoreManager.AddPoints (pointsForKill);

HealthManager.HurtPlayer(damageToGive);

}

Instantiate (impactEffect, transform.position, transform.rotation);

Destroy (gameObject);

}

}

# A.13 FlyerEnemyMove

using UnityEngine;

using System.Collections;

public class FlyerEnemyMove : MonoBehaviour {

private PlayerController thePlayer;

public float moveSpeed;

public float playerRange;

public LayerMask playerLayer;

public bool playerInRange;

public bool facingAway;

public bool followOnLockAway;

// Use this for initialization

void Start () {

thePlayer = FindObjectOfType<PlayerController> ();

}

// Update is called once per frame

void Update () {

playerInRange = Physics2D.OverlapCircle (transform.position, playerRange, playerLayer); // Verifica daca playerul este in Range

// te urmareste doar daca mergi inapoi

if(!followOnLockAway)

{

if (playerInRange) // te urmareste in orice directie

{

transform.position = Vector3.MoveTowards (transform.position, thePlayer.transform.position, moveSpeed \* Time.deltaTime);

return;

}

}

if ((thePlayer.transform.position.x < transform.position.x && thePlayer.transform.localScale.x < 0) || (thePlayer.transform.position.x > transform.position.x && thePlayer.transform.localScale.x > 0))

{

facingAway = true;

} else {

facingAway = false ;

}

if (playerInRange && facingAway)

{

transform.position = Vector3.MoveTowards (transform.position, thePlayer.transform.position, moveSpeed \* Time.deltaTime);

return;

}

}

void OnDrawGizmosSelected()

{

Gizmos.DrawSphere (transform.position, playerRange);

}

}

# A.13 HealthManager

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class HealthManager : MonoBehaviour {

public int maxPlayerHealth;

public static int playerHealth;

Text text;

private LevelManager levelManager;

//Stergem efectu ala de sange cand moare de mai multe ori

public bool isDead;

private LifeManager lifeSystem; // De la LifeManage

// Use this for initialization

void Start () {

text = GetComponent<Text>();

//playerHealth = maxPlayerHealth;

playerHealth = PlayerPrefs.GetInt ("PlayerCurrentHealth");// salveaza si face loading la vieti dupa ce mori sau cand treci nivelul

levelManager = FindObjectOfType<LevelManager>();

lifeSystem = FindObjectOfType<LifeManager> (); // De la LifeManager

isDead = false;

}

// Update is called once per frame

void Update () {

if (playerHealth <= 0 && !isDead)

{

playerHealth = 0;

levelManager.RespawnPlayer();

lifeSystem.TakeLife();

isDead = true; //tergem efectu ala de sange cand moare de mai multe ori

}

text.text = "" + playerHealth;

}

public static void HurtPlayer(int damageToGive)

{

playerHealth -= damageToGive;

PlayerPrefs.SetInt ("PlayerCurrentHealth", playerHealth); // salveaza si face loading la vieti dupa ce mori sau cand treci nivelul

}

public void FullHealth()

{

//playerHealth = maxPlayerHealth;

playerHealth = PlayerPrefs.GetInt ("PlayerMaxHealth");

PlayerPrefs.SetInt ("PlayerCurrentHealth", playerHealth); // salveaza si face loading la vieti dupa ce mori sau cand treci nivelul

}

}

# A.14 HurtEnemyOnContact

using UnityEngine;

using System.Collections;

public class HurtEnemyOnContact : MonoBehaviour {

public int damageToGive;

public float bounceOnEnemy;

private Rigidbody2D rigidbody2D;

// Use this for initialization

void Start () {

rigidbody2D = transform.parent.GetComponent<Rigidbody2D> ();

}

// Update is called once per frame

void Update () {

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Enemy")

{

other.GetComponent<EnemyHealthManager> ().giveDamage (damageToGive);

rigidbody2D.velocity = new Vector2 (rigidbody2D.velocity.x, bounceOnEnemy);

}

}

}

# A.15 HurtPlayerOnContact

using UnityEngine;

using System.Collections;

public class HurtPlayerOnContact : MonoBehaviour {

public int damageToFive;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

HealthManager.HurtPlayer(damageToFive);

other.audio.Play(); //efectul cand se atinge de alt enemy

var player = other.GetComponent<PlayerController>();

player.knockbackCount = player.knockbackLength;

if(other.transform.position.x < transform.position.x)

player.knockFromRight = true;

else

player.knockFromRight = false;

}

}

}

# A.16 KillPlayer

using UnityEngine;

using System.Collections;

public class KillPlayer : MonoBehaviour {

public LevelManager levelManager;

// Use this for initialization

void Start () {

levelManager = FindObjectOfType<LevelManager> ();

}

// Update is called once per frame

void Update () {

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

levelManager.RespawnPlayer();

}

}

}

# A.17 LadderZone

using UnityEngine;

using System.Collections;

public class LadderZone : MonoBehaviour {

private PlayerController thePlayer;

// Use this for initialization

void Start () {

thePlayer = FindObjectOfType<PlayerController> ();

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

thePlayer.onLadder = true;

}

}

void OnTriggerExit2D(Collider2D other)

{

if (other.name == "Player")

{

thePlayer.onLadder = false;

}

}

}

# A.18 LevelLoader

using UnityEngine;

using System.Collections;

public class LevelLoader : MonoBehaviour {

private bool playerInZone;

public string levelToLoad;

public string levelTag;

// Use this for initialization

void Start () {

playerInZone = false;

}

// Update is called once per frame

void Update () {

if (Input.GetKeyDown (KeyCode.W) && playerInZone)

{

LoadLevel();

}

}

public void LoadLevel()

{

PlayerPrefs.SetInt (levelTag,1);

Application.LoadLevel(levelToLoad);

}

void OnTriggerEnter2D(Collider2D other )

{

if (other.name == "Player")

{

playerInZone = true;

}

}

void OnTriggerExit2D(Collider2D other )

{

if (other.name == "Player")

{

playerInZone = false;

}

}

}

# A.19 LevelManager

using UnityEngine;

using System.Collections;

public class LevelManager : MonoBehaviour {

public GameObject currentCheckPoint;

private PlayerController player;

public GameObject deathParticle; //Animatia Rosie de Sange

public GameObject respawnParticle; //Animatia Verde

//Stergem punctele dupa ce mori

public int pointPenaltyOnDeath;

public float respawnDelay; // reswpawnul la ANimatia rosie si verde

private float gravityStore;

public HealthManager healthManager; // Am adaugat cand am immplimentat Health Sistem la player

private CameraController camera;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController>();

camera = FindObjectOfType<CameraController> ();

healthManager = FindObjectOfType<HealthManager> (); // Am adaugat cand am immplimentat Health Sistem la player

}

// Update is called once per frame

void Update () {

}

public void RespawnPlayer(){

StartCoroutine("ReswpawnPlayerCo");

}

public IEnumerator ReswpawnPlayerCo()

{

Instantiate (deathParticle, player.transform.position, player.transform.rotation);

player.enabled = false; //dezactiva miscarea dupa ce moare, il facem invizibil

player.renderer.enabled = false; //dezactiva miscarea dupa ce moare, il facem invizibil

camera.isFollowing = false;

//gravityStore = player.rigidbody2D.gravityScale; //adaugam gravitatia la player.rigidbody2D.gravityScale = 0f;

//player.rigidbody2D.gravityScale = 0f; //Oprim camera cand playerul more cazand jos

//player.rigidbody2D.velocity = Vector2.zero; //stopeaza camera dupa ce mori

ScoreManager.AddPoints (-pointPenaltyOnDeath);//Stergem punctele dupa ce murim

Debug.Log ("Player Respawn");

yield return new WaitForSeconds (respawnDelay);

//player.rigidbody2D.gravityScale = gravityStore; //adaugam gravitatia la player.rigidbody2D.gravityScale = 0f;

player.knockbackCount = 0; //Sa nu sara atunci cand isi face respawn asta dupa ce am implimetat saritura ca sa omoara monstrii

player.enabled = true; //activam miscarea dupa ce moare

player.renderer.enabled = true; //activam miscarea dupa ce moare

healthManager.FullHealth ();

healthManager.isDead = false;

camera.isFollowing = true;

player.transform.position = currentCheckPoint.transform.position;

Instantiate (respawnParticle, currentCheckPoint.transform.position, currentCheckPoint.transform.rotation);

}

}

# A.20 LevelSelectManager

using UnityEngine;

using System.Collections;

public class LevelSelectManager : MonoBehaviour {

public string[] levelTags;

public GameObject[] locks;

public bool[] levelUnlocked;

public int positionSelector;

public float distanceBelowLock;

public string[] levelName;

public float moveSpeed;

private bool isPressed;

void Start () {

for (int i = 0; i < levelTags.Length; i++)

{

if(PlayerPrefs.GetInt(levelTags[i]) == null)

{

levelUnlocked[i] = false ;

} else if (PlayerPrefs.GetInt(levelTags[i]) == 0)

{

levelUnlocked[i] = false ;

}else

{

levelUnlocked[i] = true ;

}

if(levelUnlocked[i])

{

locks[i].SetActive(false);

}

}

transform.position = locks [positionSelector].transform.position + new Vector3 (0, distanceBelowLock, 0);

}

void Update (){

}

}

# A.21 LifeManager

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class LifeManager : MonoBehaviour {

//public int startingLives;

private int lifeCounter;

private Text theText;

public GameObject gameOverScreen;

public PlayerController player;

public string mainMenu;

public float waitAfterGameOver;

// Use this for initialization

void Start () {

theText = GetComponent<Text>();

//lifeCounter = startingLives;

lifeCounter = PlayerPrefs.GetInt ("PlayerCurrentLives");

player = FindObjectOfType<PlayerController> ();

}

// Update is called once per frame

void Update () {

if (lifeCounter < 0)

{

gameOverScreen.SetActive(true);

player.gameObject.SetActive(false);

}

theText.text = "x " + lifeCounter;

if (gameOverScreen.activeSelf)

{

waitAfterGameOver -=Time.deltaTime;

}

if (waitAfterGameOver < 0)

{

Application.LoadLevel(mainMenu);

}

}

public void GiveLife()

{

lifeCounter++;

PlayerPrefs.SetInt ("PlayerCurrentLives", lifeCounter);

}

public void TakeLife()

{

lifeCounter--;

PlayerPrefs.SetInt ("PlayerCurrentLives", lifeCounter);

}

}

# A.22 LifePickup

using UnityEngine;

using System.Collections;

public class LifePickup : MonoBehaviour {

private LifeManager lifeSystem;

// Use this for initialization

void Start () {

lifeSystem = FindObjectOfType<LifeManager> ();

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.name == "Player")

{

lifeSystem.GiveLife();

Destroy(gameObject);

}

}

}

# A.23 MainMenu

using UnityEngine;

using System.Collections;

public class MainMenu : MonoBehaviour {

public string startLevel;

public string levelSelect;

public int playerLives;

public int playerHealth;

public string level1Tag;

public void newGame()

{

PlayerPrefs.SetInt ("PlayerCurrentLives", playerLives);

PlayerPrefs.SetInt ("CurrentPlayerScore", 0);

PlayerPrefs.SetInt ("PlayerCurrentHealth", playerHealth);

PlayerPrefs.SetInt ("PlayerMaxHealth", playerHealth);

PlayerPrefs.SetInt (level1Tag, 1);

Application.LoadLevel (startLevel);

}

public void LevelSelect()

{

PlayerPrefs.SetInt ("PlayerCurrentLives", playerLives);

PlayerPrefs.SetInt ("CurrentPlayerScore", 0);

PlayerPrefs.SetInt ("PlayerCurrentHealth", playerHealth);

PlayerPrefs.SetInt ("PlayerMaxHealth", playerHealth);

PlayerPrefs.SetInt (level1Tag, 1);

Application.LoadLevel (levelSelect);

}

public void QuitGame()

{

Application.Quit ();

}

}

# A.24 MovingPlatform

using UnityEngine;

using System.Collections;

public class MovingPlatform : MonoBehaviour {

public GameObject platform;

public float moveSpeed;

public Transform currentPoint;

public Transform[] points;

public int pointSelection;

// Use this for initialization

void Start () {

currentPoint = points [pointSelection];

}

// Update is called once per frame

void Update () {

platform.transform.position = Vector3.MoveTowards (platform.transform.position, currentPoint.position, Time.deltaTime \* moveSpeed);

if (platform.transform.position == currentPoint.position)

{

pointSelection++;

//Noi nu avem punctul 2 adica mai mare ca elementul 1 , si cum trece de elementul 0 si 1 el repede se intoarce la elementul 0

if(pointSelection == points.Length)

{

pointSelection = 0;

}

currentPoint = points[pointSelection];

}

}

}

# A.25 NinjaStarController

using UnityEngine;

using System.Collections;

public class NinjaStarController : MonoBehaviour {

public float speed;

public PlayerController player;

public GameObject enemyDeathEffect;

public GameObject impactEffect;

public int pointsForKill;

public float rotationSpeed; //rotatia la cutite ninja

public int damageToGive;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController>();

if (player.transform.localScale.x < 0)

{

speed = -speed;

rotationSpeed = -rotationSpeed; //rotatia la cutitele ninja

}

}

// Update is called once per frame

void Update () {

rigidbody2D.velocity = new Vector2 (speed, rigidbody2D.velocity.y);

rigidbody2D.angularVelocity = rotationSpeed;

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Enemy")

{

//Instantiate(enemyDeathEffect, other.transform.position , other.transform.rotation);

//Destroy(other.gameObject);

//ScoreManager.AddPoints (pointsForKill);

other.GetComponent<EnemyHealthManager>().giveDamage(damageToGive);//cat damage o sa dea cutitele in monstri

}

if(other.tag == "Boss")

{

other.GetComponent<BossHealthManager>().giveDamage(damageToGive);

}

Instantiate (impactEffect, transform.position, transform.rotation);

Destroy (gameObject);

}

}

# A.26 Parallax

using UnityEngine;

using System.Collections;

public class Parallax : MonoBehaviour {

public Transform[] backgrounds;

private float[] parallaxScales;

public float smoothing;

private Transform cam;

private Vector3 previousCamPos;

// Use this for initialization

void Start () {

cam = Camera.main.transform;

previousCamPos = cam.position;

parallaxScales = new float[backgrounds.Length];

for (int i = 0; i < backgrounds.Length; i++)

{

parallaxScales[i] = backgrounds[i].position.z \* -1 ;

}

}

// Update is called once per frame

void LateUpdate () {

for (int i = 0; i< backgrounds.Length; i++)

{

float parallax = (previousCamPos.x - cam.position.x) \* parallaxScales[i];

float backgroundTargetPosX = backgrounds[i].position.x + parallax ;

Vector3 backgroundTargetPos = new Vector3(backgroundTargetPosX, backgrounds[i].position.y , backgrounds[i].position.z);

backgrounds[i].position = Vector3.Lerp ( backgrounds[i].position,backgroundTargetPos , smoothing \* Time.deltaTime);

}

previousCamPos = cam.position;

}

}

# A.27 PauseMenu

using UnityEngine;

using System.Collections;

public class PauseMenu : MonoBehaviour {

public string levelSelect;

public string mainMenu;

public bool isPaused;

public GameObject pauseMenuCanvas;

// Update is called once per frame

void Update () {

if (isPaused)

{

pauseMenuCanvas.SetActive (true);

Time.timeScale = 0f; // pauzeaza jocul

} else {

pauseMenuCanvas.SetActive(false);

Time.timeScale = 1f; //reia jocul

}

if (Input.GetKeyDown (KeyCode.Escape))

{

isPaused = !isPaused;

}

}

public void Resume()

{

isPaused = false;

}

public void LevelSelect()

{

Application.LoadLevel (levelSelect);

}

public void Quit()

{

Application.LoadLevel (mainMenu);

}

}

# A.28 PlayerController

using UnityEngine;

using System.Collections;

public class PlayerController : MonoBehaviour {

public float moveSpeed;

private float moveVelocity;

public float jumpHeight;

public Transform groundCheck; //pentru FixedUpdate

public float groundCheckRadius; //pentru FixedUpdate

public LayerMask whatIsGround; //pentru FixedUpdate

private bool grounded; //pentru FixedUpdate

private bool doubleJumped;

private Animator anim; // Animatia

public Transform firePoint;

public GameObject ninjaStar;

public float shotDelay;

private float shotDelayCounter;

//contactu cu monstrii , sa nu stea pe loc si sa se opreasca unu pe altul

public float knockback;

public float knockbackLength;

public float knockbackCount;

public bool knockFromRight;

//pentru Usa

public bool onLadder;

public float climbSpeed;

private float climbVelocity;

private float gravityStore;

// Use this for initialization

void Start () {

anim = GetComponent<Animator> ();

gravityStore = rigidbody2D.gravityScale;

}

//Pentru a nu sari la vesnicie atunci cand tastam space

void FixedUpdate(){

grounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround);

}

// Update is called once per frame

void Update () {

if (grounded)

doubleJumped = false;

anim.SetBool ("Grounded", grounded); // Animatia , cand tastezi space sa nu se miste

//pentru a sari

if(Input.GetKeyDown(KeyCode.Space) && grounded)

{

//rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x,jumpHeight);

Jump();

}

//pentru a sari de 2 ori

if(Input.GetKeyDown(KeyCode.Space) && !doubleJumped && !grounded)

{

//rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x,jumpHeight);

Jump();

doubleJumped = true;

}

//pentru a stopa oprire dupa ce mori

//moveVelocity = 0f; //Am comentat dupa ce am optimizat controlul functiilor comentate mai jos

//Pentru a se misca la Dreapta

//if(Input.GetKey(KeyCode.D))

//{

//rigidbody2D.velocity = new Vector2(moveSpeed,rigidbody2D.velocity.y);

//moveVelocity = moveSpeed;

//}

//Pentru a se misca la Stanga

//if(Input.GetKey(KeyCode.A))

//{

//rigidbody2D.velocity = new Vector2(-moveSpeed,rigidbody2D.velocity.y);

//moveVelocity = -moveSpeed;

//}

//Optimizam controlul functiilor comentate mai sus

moveVelocity = moveSpeed \* Input.GetAxisRaw ("Horizontal");

if (knockbackCount <= 0)

{//pentru a stopa oprire dupa ce mori

rigidbody2D.velocity = new Vector2 (moveVelocity, rigidbody2D.velocity.y);

} else {

if(knockFromRight)

rigidbody2D.velocity = new Vector2(-knockback , knockback);

if(!knockFromRight)

rigidbody2D.velocity = new Vector2(knockback , knockback);

knockbackCount -= Time.deltaTime;

}

anim.SetFloat ("Speed", Mathf.Abs(rigidbody2D.velocity.x));

//vederea ca in oglinda atunci cand te misti cu animatia

if (rigidbody2D.velocity.x > 0)

transform.localScale = new Vector3 (1f, 1f, 1f);

else if(rigidbody2D.velocity.x < 0)

transform.localScale = new Vector3 (-1f, 1f, 1f);

//Cutite Ninja

if (Input.GetButtonDown ("Fire1"))

{

Instantiate(ninjaStar, firePoint.position , firePoint.rotation);

shotDelayCounter = shotDelay;

}

if (Input.GetButtonDown ("Fire1"))

{

shotDelayCounter -= Time.deltaTime;

if(shotDelayCounter <= 0 )

{

shotDelayCounter = shotDelay;

Instantiate(ninjaStar, firePoint.position , firePoint.rotation);

}

}

if(anim.GetBool("Sword"))

anim.SetBool("Sword", false);

if (Input.GetButtonDown("Fire2"))

{

anim.SetBool("Sword", true);

}

if (onLadder)

{

rigidbody2D.gravityScale = 0f;

climbVelocity = climbSpeed \* Input.GetAxisRaw("Vertical");

rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x , climbVelocity);

}

if (!onLadder)

{

rigidbody2D.gravityScale = gravityStore;

}

}

public void Jump()

{

rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x,jumpHeight);

}

//playerul sa se miste odata cu platforma miscatoare

void OnCollisionEnter2D(Collision2D other)

{

if (other.transform.tag == "MovingPlatform")

{

transform.parent = other.transform;

}

}

// playerul nu se mai misca odata cu platforma atunci cand sare de pe ea

void OnCollisionExit2D(Collision2D other)

{

if (other.transform.tag == "MovingPlatform")

{

transform.parent = null;

}

}

}

# A.29 ScoreManager

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class ScoreManager : MonoBehaviour {

public static int score;

Text text;

void Start()

{

text = GetComponent<Text> ();

//score = 0;

score = PlayerPrefs.GetInt ("CurrentPlayerScore");

}

void Update()

{

if (score < 0)

score = 0;

text.text = "" + score;

}

public static void AddPoints (int pointsToAdd)

{

score += pointsToAdd;

PlayerPrefs.SetInt ("CurrentPlayerScore", score);

}

public static void Reset()

{

score = 0;

PlayerPrefs.SetInt ("CurrentPlayerScore", score);

}

}

# A.30 ShootAtPlayerInRange

using UnityEngine;

using System.Collections;

public class ShootAtPlayerInRange : MonoBehaviour {

public float playerRange;

public GameObject enemyStar;

public PlayerController player;

public Transform launchPoint;

public float waitBetweenShots;

private float shotCounter;

// Use this for initialization

void Start () {

player = FindObjectOfType<PlayerController> ();

shotCounter = waitBetweenShots;

}

// Update is called once per frame

void Update () {

Debug.DrawLine (new Vector3 (transform.position.x - playerRange, transform.position.y, transform.position.z), new Vector3 (transform.position.x + playerRange, transform.position.y, transform.position.z));

shotCounter -= Time.deltaTime;

if (transform.localScale.x < 0 && player.transform.position.x > transform.position.x && player.transform.position.x < transform.position.x + playerRange && shotCounter < 0 )

{

Instantiate(enemyStar,launchPoint.position , launchPoint.rotation);

shotCounter = waitBetweenShots;

}

if (transform.localScale.x > 0 && player.transform.position.x < transform.position.x && player.transform.position.x > transform.position.x - playerRange && shotCounter < 0)

{

Instantiate(enemyStar,launchPoint.position , launchPoint.rotation);

shotCounter = waitBetweenShots;

}

}

}

# Anexa B Screenshot-uri



Figura B.1- Meniul Jocului



Figura B.2- Interacțiunea cu obiecte



Figura B.3- Mișcarea eroului



Figura B.3- Săritură în prăpastie



Figura B.4- Lupa dintre erou si dușmanii săi