



UNIVERSITATEA DIN  
BUCUREŞTI



FACULTATEA DE  
MATEMATICA ŞI  
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

# JOC 2D TOP-DOWN SHOOTER: APOCALYPSE DEFENDERS

Absolvent  
Ionescu Alexandru

Coordonator științific  
Conf. dr. Claudia Mureșan

Bucureşti, iunie-iulie 2023

## Rezumat

Lumea jocurilor video este un refugiu foarte popular în societatea actuală, o nouă comunitate în care fiecare persoană este liberă să facă aproape orice își dorește, cât timp aceste lucruri îi aduc fericire, fără să fie judecată pentru preferințele sau abilitățile sale. Aceste lucruri sunt valabile pentru jucători, dar și pentru creatorii de jocuri, care sunt liberi să creeze orice își imaginează. Astfel, am decis să realizez un joc care, prin unele aspecte, precum cel vizual sau al unor anumite mecanici, reamintește de jocurile jucate în copilărie, care m-au introdus în această lume ce m-a învățat atât de multe și mi-a adus atâtea amintiri plăcute. În această lucrare voi prezenta procesul de creație al unui joc de tip 2D top-down shooter. Jocul este realizat în game engine-ul Unity și prelucrarea este făcută cu script-uri în C#, folosind în mare parte principii de programare orientată pe obiecte, folosind mediul de dezvoltare Microsoft Visual Studio.

Lucrarea prezintă în detaliu tot procesul prin care se dezvoltă un joc, de la conceptualizarea unei idei, la crearea unui workflow și al unei strategii de lucru, la crearea mediului de joc, la gestionarea scripturilor și integrarea acestora cu obiectele, la gândirea și implementarea mecanicilor de joc, la procesul creativ de realizare a interfeței grafice și a părții vizuale a jocului, până la combinarea tuturor acestor lucruri pentru a realiza un joc complet, a-l testa și a rezolva orice alte probleme apărute, pentru a avea un produs finit de care sunt mulțumit. Toate aceste procese au la bază concepte academice care au fost dobândite pe parcursul facultății: programarea orientată pe obiecte, folosirea unui game engine, folosirea unui mediu de dezvoltare pentru cod, folosirea și înțelegerea documentației relevante, debugging și troubleshooting, chiar și scrierea acestui document, toate acestea au fost cruciale pentru realizarea acestui proiect. Prin intermediul acestor aspecte, am demonstrat abilitățile și cunoștințele dobândite în domeniul dezvoltării de jocuri și am creat o experiență interactivă și captivantă pentru jucători.

În concluzie, acest proiect aduce o contribuție valoroasă în domeniul dezvoltării de jocuri și evidențiază abilitățile și cunoștințele dobândite în implementarea unui joc de tip 2D top-down shooter. Jocul dezvoltat oferă o experiență captivantă și explorează concepte academice relevante, reprezentând o sursă de inspirație și învățare pentru alți dezvoltatori și cercetători în domeniul jocurilor.

## Abstract

The world of video games is a highly popular refuge in today's society, a new community where each person is free to do as they please, in their own way, as long as it brings them happiness, without being judged for their preferences or abilities. This applies not only to players but also to game developers, who are free to create whatever they imagine. Thus, I decided to create a game that, in some aspects, such as visually or with certain mechanics, reminds me of the games I played in my childhood, but also honours their legacy for me, which introduced me to this world that taught me so much and brought me so many great memories. In this paper, I will present the process of creating a 2D top-down shooter game. The game is developed in the Unity game engine, and the coding is done using scripts in C#, primarily using object-oriented programming principles, utilizing the Microsoft Visual Studio development environment.

The paper provides a detailed overview of the entire game development process, from conceptualizing an idea, to creating a workflow and work strategy, building the game environment, managing and integrating scripts with objects, designing and implementing game mechanics, the creative process of creating the visual interface and graphical elements of the game, leading to combining all these elements to create a complete game, testing it, and solving any issues that may arise, ultimately making a finished product that I am proud of. All these processes are based on academic concepts that have been acquired throughout my studies, including object-oriented programming, using and configuring a game engine, utilizing a development environment for coding, understanding and utilizing relevant documentation, debugging and troubleshooting, and even the process of writing this document. All of these elements have been crucial in the making of this project. Through these aspects, I have demonstrated the skills and knowledge acquired in the field of game development and have created an interactive and captivating experience for players.

In conclusion, this academic project makes a valuable contribution to the field of game development and highlights the skills and knowledge gained in implementing a 2D top-down shooter game. The game offers an engaging experience and explores relevant academic concepts, serving as a source of inspiration and learning for other developers and researchers in the field of games.

# Cuprins

<b>1 Introducere</b>	<b>6</b>
1.1 Motivație . . . . .	6
1.2 Prezentare generală a proiectului . . . . .	6
1.2.1 Povestea jocului . . . . .	6
1.2.2 Tematica jocului . . . . .	7
1.2.3 Descrierea pe scurt a jocului . . . . .	7
1.3 Domenii abordate și aplicații folosite . . . . .	8
1.3.1 Dezvoltare joc - Unity . . . . .	8
1.3.2 Programare orientată pe obiecte - Microsoft Visual Studio (C#) . . . . .	9
1.3.3 Design grafic - Photoshop . . . . .	9
1.4 Scopul lucrării . . . . .	9
1.5 Structura lucrării . . . . .	10
<b>2 Preliminarii</b>	<b>11</b>
2.1 Definire Termeni . . . . .	11
2.1.1 Funcțiile predefinite din Unity . . . . .	12
2.2 Diagramă UML design nivel . . . . .	13
2.3 Workflow . . . . .	14
2.4 Resurse folosite preluate de pe internet . . . . .	14
2.4.1 Pathfinding - A* . . . . .	14
2.4.2 Modele caractere și design niveluri . . . . .	15
2.4.3 Interfața grafică . . . . .	16
2.4.4 Powerup-uri . . . . .	17
<b>3 Implementarea jocului</b>	<b>18</b>
3.1 Player . . . . .	18
3.1.1 Caracteristici . . . . .	18
3.1.2 Mișcarea . . . . .	19
3.1.3 Folosirea armelor . . . . .	19
3.1.4 Bara de viață . . . . .	20
3.2 Arme . . . . .	21

3.2.1	Caracteristici . . . . .	21
3.2.2	Tragerea . . . . .	21
3.2.3	Reîncărcarea armei . . . . .	23
3.3	Glonțul . . . . .	23
3.4	Inamici . . . . .	24
3.4.1	Spawner Inamici . . . . .	24
3.4.2	Damage și eliminarea inamicilor . . . . .	25
3.4.3	Zombie . . . . .	26
3.4.4	Robot . . . . .	27
3.4.5	Asasinul . . . . .	28
3.4.6	Bara de viață . . . . .	29
3.5	Powerup-uri . . . . .	29
3.5.1	Heal Powerup . . . . .	31
3.5.2	Damage Multiplier Powerup . . . . .	31
3.5.3	Speed Powerup . . . . .	32
3.5.4	Ammo Powerup . . . . .	32
3.6	Game Manager, Audio Manager, Game Controller . . . . .	33
3.7	Metodele de joc - Story și Endless . . . . .	39
3.8	Interfața grafică . . . . .	39
3.8.1	Meniul principal . . . . .	39
3.8.2	Interfața grafică în nivel . . . . .	45
3.8.3	Meniul de pauză din nivel . . . . .	47
3.8.4	Meniul de victorie și înfrângere din nivel . . . . .	48
3.8.5	Panourile cu text informativ și poveste . . . . .	50
3.9	Scalabilitatea jocului . . . . .	51
3.10	Modificările făcute în editorul Unity . . . . .	51
<b>4</b>	<b>Concluzii</b>	<b>56</b>
4.1	Posibile dezvoltări ulterioare . . . . .	56
<b>Bibliografie</b>		<b>58</b>

# Capitolul 1

## Introducere

### 1.1 Motivație

Jocurile au o importanță aparte pentru mine. La doar câțiva ani, am învățat alfabetul jucându-mă un joc făcut de tatăl meu și de un coleg de-al lui și privind tastatura, pentru a învăța controalele pentru alte jocuri. Crescând, am petrecut nenumărate ore împreună cu tatăl meu jucându-ne jocuri precum Age of Empires sau Starcraft. Pe lângă amintirile frumoase, jocurile m-au învățat să lucrez în echipă cu alte persoane, să munesc pentru a îmi îmbunătăți abilitățile sau să îmi fac planuri înainte să acționez. Am ales, pentru proiectul de licență, să creez un joc care îmi aduce aminte de acele jocuri care se pot numi deja "retro", în care aspectul simplu și plăcut este important, însă nu este atracția principală, ci plăcerea de a-l juca. Crearea unui joc implică o gamă largă de competențe, iar managementul proiectului este extrem de important, deoarece majoritatea componentelor într-un joc funcționează împreună, astfel că este necesar ca toate aceste componente să funcționeze corespunzător chiar și pentru a putea dezvolta jocul. Dezvoltarea unui joc îți permite să îți dezvolți și să îți arăți abilitățile tehnice, creativitatea și organizarea într-un mod practic și aplicat.

### 1.2 Prezentare generală a proiectului

#### 1.2.1 Povestea jocului

Ne aflăm în viitor, iar cei mai mari răufăcători ai omenirii au inventat o armă pe care nimeni nu știe cum să o contracareze. Cu multe sacrificii, o agenție secretă creată cu un singur scop, numită *Apocalypse Defenders*, reușește să fure secretele care pot duce la distrugerea acestei arme, însă aceasta este decimată în acest proces. Ultimul agent rămas are cea mai importantă misiune a lumii, să evadeze cu secretele care pot duce la înfrângerea răufăcătorilor, însă misiunea lui este foarte dificilă. Dușmanii săi au găsit casa pe care agenții o foloseau ca bază de operațiuni și i-au eliminat toți camarazii, iar acum îl

caută și pe acesta. Este încolțit în una din camere și trebuie să se lupte pentru a traversa fiecare cameră, până să iasă din casă și din curte până la locul de evacuare stabilit cu salvatorii săi, care nu știu în ce situație extremă se află acesta și nu pot verifica, pentru că nu risca să fie deconspirati. Agentul trebuie să evadeze, toată omenirea se bazează pe el.

### 1.2.2 Tematica jocului

Acest proiect reprezintă un joc de tip 2D Top-Down Shooter. Cum acțiunea jocului se desfășoară în viitor, tema jocului trebuie să reflecte acest lucru. Astfel, pe lângă interfața cu aspect futurist, inamicii trimiși de răufăcătorii care sunt atât de puternici încât pot distrugе lumea trebuie să ilustreze puterea acestora. Astfel, inamicii sunt roboti, zombi și asasini care sunt clonați de acești răufăcători, și pot apărea oriunde. Acești inamici pot lovi jucătorul de aproape sau pot trage după acesta cu arme puternice. Am dorit ca acest joc să aibă aspect și design simplu, cu un stil "retro" care să semene cu jocurile pe care le jucam când eram copil, care aveau de multe ori și o tematică asemănătoare: top down shooting. Acest stil ar da mai multă atenție mecanicilor propriu-zise și al jucatului jocului, făcând jocul mai captivant.

### 1.2.3 Descrierea pe scurt a jocului

Acest joc se poate juca în două moduri: modul *story*, în care se desfășoară povestea descrisă mai sus, pe niveluri, precum și în modul endless, unde jucătorul este în același nivel la infinit și singurul scop e să învingă cât mai mulți adversari pentru a face un scor cât mai mare și un timp record. Pentru mai multă varietate și strategii diferite, jucătorul poate alege caracterul preferat, fiecare având avantaje și dezavantaje.

La pornirea jocului, utilizatorul este întâmpinat de meniul principal, unde poate închide jocul sau poate comuta la meniurile de opțiuni sau de joc. În meniul de opțiuni, poate modifica setări grafice sau audio, iar în meniul de joc poate selecta din mai multe variante pentru caracterul pe care dorește să îl folosească, adică aspectul vizual și caracteristicile de atac, viață, viteza de mișcare ale agentului secret. Fiecare caracter are avantaje și dezavantaje, iar jucătorul poate să își gândească strategia în funcție de detaliile personajului ales. Pentru a învinge inamicii, jucătorul are la dispoziție 4 arme: pumnii, cu care nu poate lovi inamicii, însă care îi dau viteza de mișcare mult mai mare față de celelalte arme; pistolul, care are un număr infinit de gloanțe, însă nu lovește inamicii foarte tare; o armă automată, care trage foarte repede și lovește puțin mai tare decât pistolul, însă are gloanțe limitate și jucătorul este mai lent atunci când o folosește și pușca, aceasta lovind foarte tare, însă doar de la distanțe mici, cu număr mic, limitat de gloanțe și cu o viteza de mișcare a jucătorului foarte mică. În meniul de joc se poate alege și modul de joc, unul dintre cele două descrise mai devreme: cel de poveste (*story*) și cel infinit (*endless*).

Dacă alege modul endless, jucătorul este trimis direct în nivelul infinit, iar dacă alege

modul story, este trimis în meniul de alegere a nivelului, unde poate selecta nivelul pe care vrea să îl joace, dintre cele deblocate. Inițial, este deblocat doar nivelul 1, iar următoarele sunt deblocate după câștigarea nivelului precedent. Pentru a câștiga un nivel din poveste, jucătorul are unul din două obiective în fiecare nivel: să distrugă un număr dat de inamici sau să reziste un timp dat inamicilor care apar pe toată durata acestui nivel. Controalele sunt următoarele: mișcarea se realizează cu tastele W,A,S,D sau cu săgețile, iar trasul cu arma cu click stânga pe mouse. În timpul jocului, utilizatorul are o interfață grafică în care poate observaarma selectată, precum și celealte arme, timpul scurs de la începerea nivelului și scorul curent. La scor se adaugă un punct pentru fiecare inamic eliminat. Deasupra propriului caracter, dar și deasupra inamicilor, se află câte o bară de viață, care ilustrează în mod vizual procentul de puncte de viață rămase pentru acel caracter. De asemenea, la apăsarea tastei *Escape*, jucătorul poate accesa un meniu de pauză, unde află detalii despre caracter, precum viața curentă și viața maximă, viteza de mișcare și un multiplicator al *damage*-ului, care multiplică puterea armelor, *damage*-ul reprezentând numărul de puncte de viață scăzute inamicului lovit. În acest meniu de pauză se mai află și un buton de reîncepere a nivelului, un buton pentru returnarea la meniul principal, și un buton pentru accesarea setărilor, care sunt aceleași cu cele din meniul principal. La finalizarea nivelului, prin victorie sau înfrângere, dacă jucătorul este distrus de inamici, apare câte un panou specific fiecărei situații, ambele având câte un buton de reîncepere a nivelului actual și un buton de returnare la meniul principal, iar panoul de victorie mai are și butonul de trecere la următorul nivel.

## 1.3 Domenii abordate și aplicații folosite

### 1.3.1 Dezvoltare joc - Unity

Unity este o platformă de dezvoltare a jocurilor cu o gamă largă de funcționalități. În cadrul proiectului, l-am utilizat pentru următoarele:

- Crearea mediului de joc: am construit și configurat obiectele și nivelurile din joc, plasându-le, adăugându-le componente și scripturi în C#, setând relațiile între componente sau inițializând unele variabile.
- Gestionarea scripturilor: Majoritatea lucrurilor din joc se întâmplă deoarece sunt programate în scripturile care sunt atașate în Unity obiectelor. Unity permite o integrare strânsă cu Microsoft Visual Studio, unde am dezvoltat scripturile.
- Implementarea mecanicilor de joc: Am implementat mecanicile specifice, cum ar fi mișcarea, trasul cu arma, coliziunile dintre anumite obiecte, precum dintre gloanțe și caractere, inamicii, jucătorul, sistemul de scor, nivelurile, cronometrarea timpului și altele, descrise extensiv în capitolul 2.1.

- Integrarea resurselor: Folosind Unity, am importat și gestionat resursele folosite la crearea jocului, acestea fiind atât realizate de mine, cât și de alte persoane, însă toate resursele externe folosite au o licență care permite folosirea lor. Folosind aceste resurse, precum și funcționalitățile enumerate anterior, am creat atmosfera și experiența dorită în joc.

### **1.3.2 Programare orientată pe obiecte - Microsoft Visual Studio (C#)**

Microsoft Visual Studio este un mediu de dezvoltare integrat (IDE) puternic pentru programarea în limbajul C#. Am folosit Microsoft Visual Studio pentru scrierea și structurarea codului, folosindu-mă de acesta pentru a crea clase și obiecte pentru diferite aspecte ale jocului, cum ar fi inamicii, gloanțele, dar și obiecte care nu sunt vizibile în joc, folosite pentru a plasa noi inamici pe hartă, a cronometra timpul de la începerea unui nivel, sau modificarea setărilor în joc. Folosind integrarea cu Unity, am putut schimba și sincroniza cu ușurință toate scripturile folosite în acest joc.

### **1.3.3 Design grafic - Photoshop**

Adobe Photoshop este un software de editare grafică foarte popular și puternic, pe care l-am utilizat pentru a crea anumite grafici ale jocului, pentru a crea o schiță a design-ului meniului și interfeței grafice din joc, dar și a combinației acesteia cu restul jocului, asigurând astfel o armonie vizuală atunci când jocul a fost implementat. Am mai folosit această aplicație și pentru a face mici ajustări unor imagini.

## **1.4 Scopul lucrării**

Scopurile principale au fost experiența utilizatorului și impactul asupra jucătorului: să cum am povestit și în secțiunea despre motivație, acest joc a fost conceptualizat considerând anumite emoții și amintiri, astfel că am dorit ca experiența și impactul pentru jucător să fie asemănătoare cu cele pe care le-am avut la crearea acestui joc, un joc care ar putea fi considerat o abordare nouă a jocurilor copilăriei, având însă un concept și aspect mai modern, însă care aduc totuși nostalgie.

Un alt scop este demonstrarea abilităților și cunoștințelor. Am creat această lucrare, în care au fost însumate o multitudine de tipuri de cunoștințe și abilități dobândite pe tot parcursul academic, de la programare orientată și dezvoltarea jocurilor, la redactarea unui document, la planificarea în avans și la organizarea unui workflow.

## **1.5 Structura lucrării**

Lucrarea este structurată în următoarele capitole: Capitolul 1 - Introducere, care explică jocul în mod simplu și conceptual, Capitolul 2 - Preliminarii, în care vor fi explicate anumite terminologii și funcții de bază, folosite pe tot parcursul dezvoltării și resursele folosite care au fost preluate de pe internet, cu licență care permite acest lucru, Capitolul 3 - Implementarea jocului, în care vor fi explicate în detaliu toate caracteristicile jocului și implementarea acestora și Capitolul 4 - Concluzii.

# Capitolul 2

## Preliminarii

### 2.1 Definire Termeni

*Spawnare*, sau *Instanțiere* reprezintă procesul de creare a unei instanțe sau copii a unui obiect. *TimeScale* reprezintă ritmul de trecere a timpului. Pentru 0, timpul este oprit, pentru 1, timpul se scurge normal. Toate acțiunile care implică mișcare sau forțe se scalează cu timpul folosind acest parametru, astfel că atunci când este 0, aceste acțiuni sunt puse pe pauză [17]. *Frame*, sau *cadră*, reprezintă o imagine, o multitudine de cadre derulate pe secundă formând senzația de mișcare și de video. O funcție de tipul corutină *IEnumerator*, care returnează folosind *yield return null* face ca această corutină să revină în acest punct în cadrul următor, iar apoi să înceapă din nou funcția. *Viață* unui caracter se referă la numărul de puncte de viață pe care acesta le are. Aceste puncte sunt scăzute atunci când caracterul este lovit de un inamic sau de glonțul tras de inamic, numărul de puncte de viață pierdute reprezentând *damage-ul* dat de acel inamic. Acest *damage* poate fi multiplicat cu ajutorul unor bonusuri, numite *damage multiplier*. Aceste bonusuri, care sunt de mai multe tipuri, nu doar pentru multiplicarea damage-ului, sunt numite în acest joc și *buff-uri*, aceste *buff-uri* fiind primite ori la ridicarea unor obiecte de pe hartă, numite *powerup-uri*. Câțiva termeni legați de motorul de joc Unity, folosiți pe parcursul lucrării, sunt: *Layer*, care reprezintă un mod de a separa obiectele din joc pe ”straturi”, modificând astfel modul cum interacționează și cum se văd în joc; *PlayerPrefs* reprezintă o clasă care stochează anumite valori simple date, precum numere sau texte, între sesiunile de joc [15], fiind utilă pentru a salva lucruri simple, precum setările jocului preferate de utilizator, caracterul selectat sau ultimul nivel deblocat; *Canvas* reprezintă un spațiu pe care se adaugă obiectele din cadrul interfeței grafice [11]. Alți termeni folosiți pe parcursul lucrării sunt *highscore*, ce reprezintă scorul record stabilit de jucător într-un anumit nivel, iar *pathfinding* reprezintă un algoritm pentru găsirea unui drum, de preferință cât mai scurt, evitând obstacolele, de la inamici la jucător, pentru mișcarea inamicilor.

### 2.1.1 Funcțiile predefinite din Unity

Acste funcții fac parte din clasa MonoBehaviour [13], care conține funcțiile de bază, unele dintre acestea fiind folosite în toate scripturile din joc.

#### **Awake, Start, Update, FixedUpdate**

Funcția Awake este apelată atunci când obiectul la care scriptul este atașat începe să existe, ori când scena este încărcată, ori când acesta este instantiat.

Funcția Start este apelată după finalizarea funcțiilor Awake, tot atunci când existența obiectului începe.

Funcția Update este apelată în fiecare frame al execuției, iar funcția FixedUpdate este apelată la un număr fixat de frame-uri.

#### **OnTriggerEnter2D, OnCollisionEnter2D, OnBecomeInvisible**

Funcțiile OnTriggerEnter2D și OnCollisionEnter2D sunt apelate atunci când obiectul intră în contact cu un alt obiect. Aceste obiecte trebuie să aibă *collidere* atașate, care sunt componente care adaugă coliziuni și detecția coliziunilor. Aceste collidere pot fi sau nu *trigger*; dacă collider-ul este trigger, aceste obiecte nu se vor mai lovi, ci vor trece unul prin celălalt. Funcția OnTriggerEnter2D detectează când un alt obiect intră în trigger collider-ul atașat obiectului de care aparține și script-ul, iar funcția OnCollisionEnter2D detectează când un alt obiect intră în coliziune cu collider-ul obiectului de care este aparține și script-ul.

Funcția OnBecomeInvisible este apelată atunci când obiectul la care scriptul este atașat nu mai este în câmpul vizual al unei camere. În acest proiect, este folosită o singură cameră, toată acțiunea se întâmplă în câmpul vizual al acesteia, iar funcția aceasta este folosită pentru a distruge obiectele care ies din câmpul vizual al camerei, întrucât acestea ar încetini jocul degeaba.

## 2.2 Diagramă UML design nivel

Pentru a vizualiza modul de funcționare a unui nivel din povestea jocului, am realizat o diagramă UML care descrie tot procesul prin care un utilizator poate trece pe parcursul unui nivel.

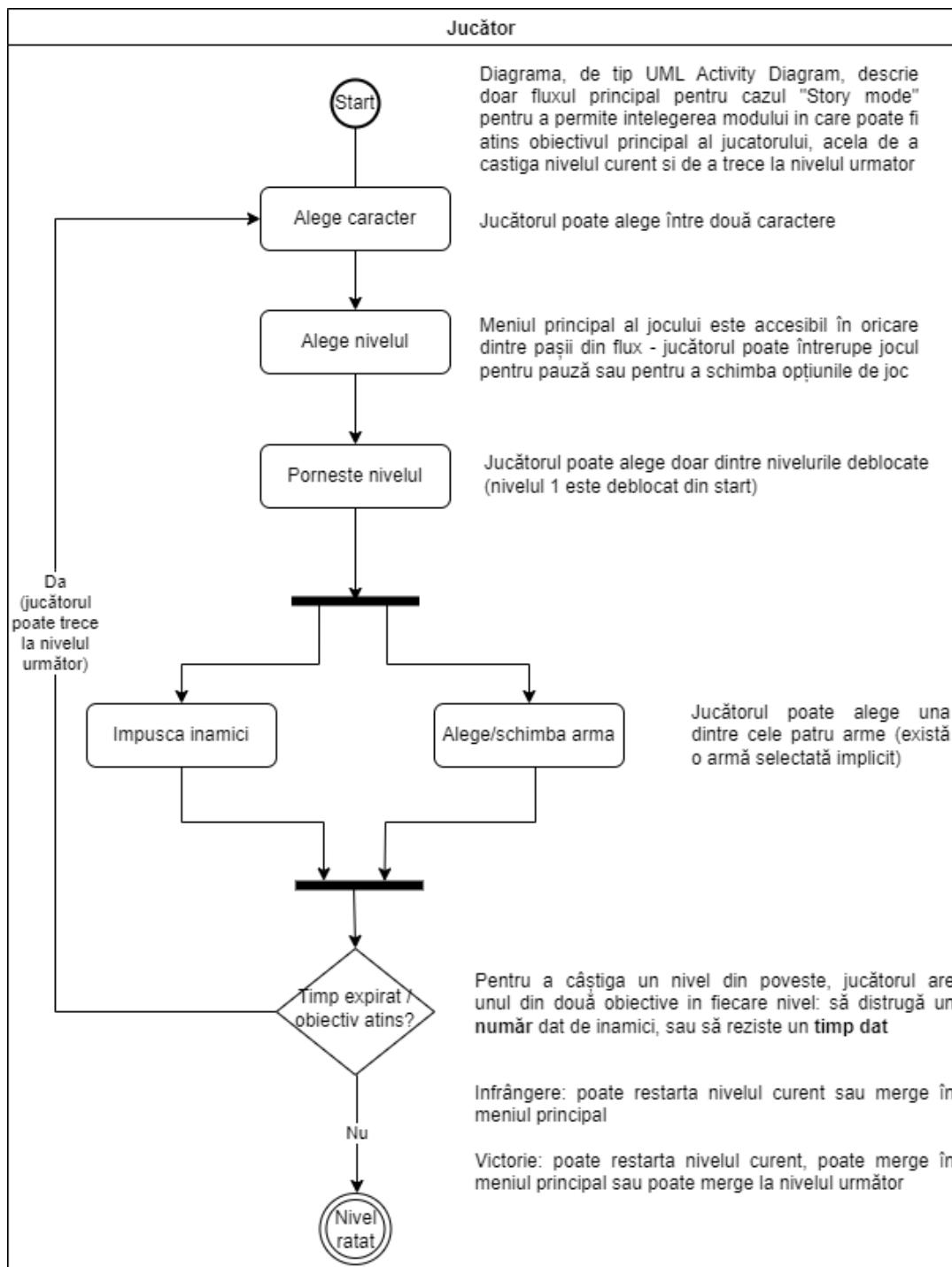


Figura 2.2.0.1: Diagramă de design a unui nivel din poveste

## 2.3 Workflow

Cunoscând interdependența dintre componentele unui joc, am considerat foarte importantă o ordine a rezolvării task-urilor, dar și consemnarea tuturor acestor task-uri, întrucât dacă uți rezolvarea unora dintre ele, nimic nu ar funcționa, inclusiv task-urile rezolvate cu succes. Astfel, stabilirea unui workflow, pentru a stabili o ordine a rezolvării problemelor și obiectivelor următoare, dar și consemnarea celor deja rezolvate într-o ordine clară a fost un pas foarte important, care a ajutat foarte mult în realizarea acestui proiect. Am folosit mai multe board-uri Trello, împărțite pe secțiuni relevante, precum task-urile în curs de rezolvare, cele rezolvate, cele următoare, clasificate pe categorii ca *interfață grafică, jucător, inamici, arme, bug-uri*. Având în vedere succesul acestei metode pentru dezvoltarea jocului, am folosit-o și pentru realizarea acestei documentații.



Figura 2.3.0.1: Câteva panouri din dezvoltarea jocului

## 2.4 Resurse folosite preluate de pe internet

### 2.4.1 Pathfinding - A\*

Pentru ca inamicii să poată urmări jucătorul în mod eficient și realistic, ocolind obstacole și găsind cel mai scurt drum spre acesta, am folosit o librărie [5] specială pentru pathfinding, care folosește metoda *Astar* pentru a găsi cel mai scurt drum către obiectiv. Decizia de a alege A\* ca algoritm de pathfinding a fost făcută deoarece acesta este considerat unul dintre cele mai bune soluții pentru a rezolva această problemă, dar este și foarte bine optimizat. Acest algoritm funcționează astfel: pentru găsirea unui drum, algoritmul A\* examinează succesiv cele mai promițătoare locații neexplorate pe care le-a văzut. Când o locație este explorată, algoritmul se încheie dacă aceasta este destinația; altfel, memorează toți vecinii acelei locații pentru explorarea ulterioară [3].

#### Folosirea algoritmului

Am creat un GameObject gol, pe care am atașat scriptul Pathfinder din librărie, iar în setările acestuia am modificat suprafața grid-ului pentru a conține toată suprafața de joc. Obstacolele de pe hartă sunt toate într-un layer, care este trecut în setările acestui script. Algoritmul astfel determină locurile accesibile și inaccesibile de pe hartă. Zonele albastre sunt accesibile, iar cele cu pătratul roșu sunt inaccesibile, așa cum se vede în figura următoare.



Figura 2.4.1.1: Nivelul cu gridul A\*

Pe fiecare GameObject de tipul inamic sunt atașate 3 script-uri din librăria Astar:

- Seeker: Acest script se ocupă de găsirea traseului optim spre destinație în fiecare cadru;
- AIPath: Acest script urmărește traseul găsit de Seeker;
- AIDestinationSetter: Acest script setează destinația ca un obiect specificat.

## 2.4.2 Modele caractere și design niveluri

Pentru design-ul caracterelor și al nivelurilor am folosit un pachet de asset-uri [7] făcut pentru jocuri de tip 2D Top-down shooter, care conține mai multe modele de caractere și multe obiecte pentru design-ul nivelurilor care reflectă povestea și fac acțiunea mai interesantă.

Cum obiectivul jucătorului în acest joc este să evadeze din casă în viață, nivelurile se vor desfășura în diverse locuri ale casei și în curtea acesteia, iar agentul își face loc prin diferitele tipuri de inamici trimiși de răufăcători pentru a-i nu lăsa să scape din casă cu secretele care vor duce la înfrângerea lor și salvarea lumii.

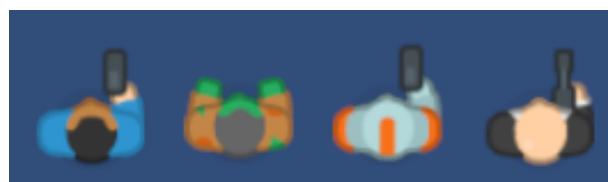


Figura 2.4.2.1: Caractere

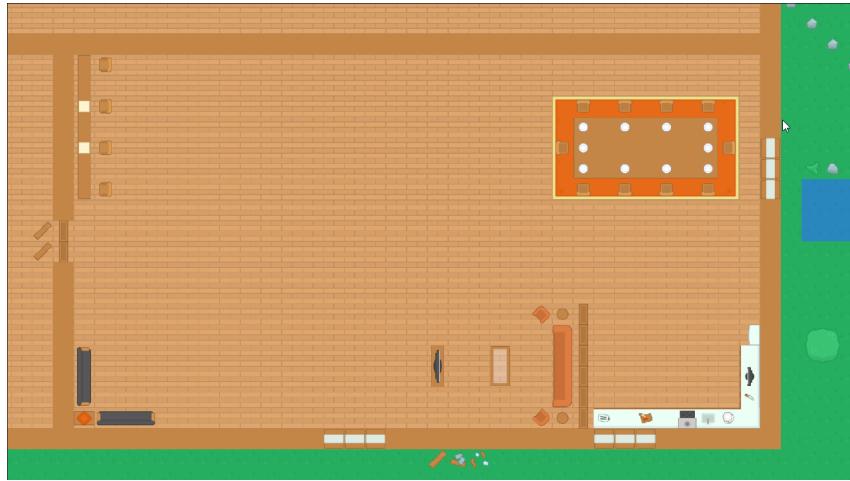


Figura 2.4.2.2: Design nivel

### 2.4.3 Interfață grafică

Cum jocul se desfășoară într-o lume futuristă, interfața grafică trebuie să reflecte acest lucru, în acest caz prin alegerea fontului și a combinațiilor de culori. Aceasta trebuie să fie totuși ușor de folosit și înțeles, cu un layout asemănător cu cele cunoscute de majoritatea oamenilor și cu un aspect plăcut. În meniul de pauză, atunci când jucătorul este într-un nivel, acesta poate vedea și statistici despre personajul său, precum viteza de mișcare, viața curentă și viața maximă, multiplicatorul de damage, dar și timpul scurs de la începerea nivelului. În modul de niveluri, unde este o poveste liniară, acest cronometru este util pentru încercarea de a completa un nivel cât mai rapid, dar și pentru a identifica rata de spawnare a inamicilor, pentru a putea crea o strategie de luptă în acel nivel. Acest pachet grafic [8] se potrivește cu viziunea mea legată de interfața grafică în acest joc.

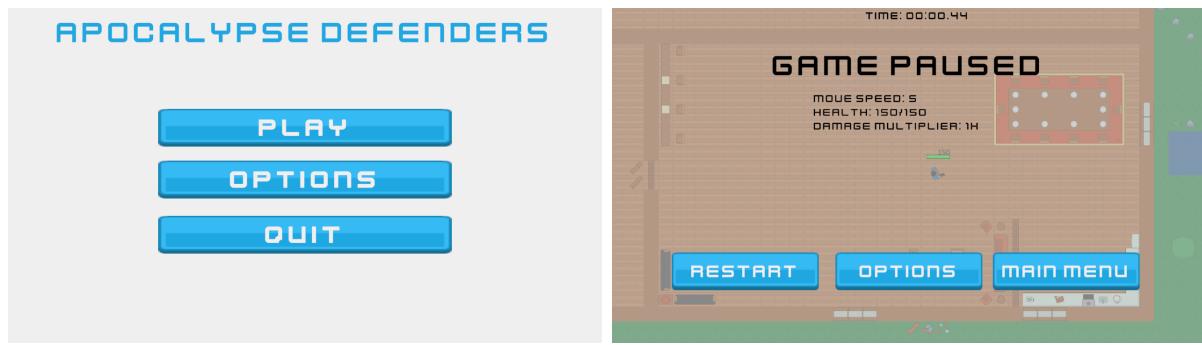


Figura 2.4.3.1: Interfața grafică în meniul principal și în meniul de pauză

Interfața grafică din timpul nivelului, cu selectarea armelor, folosește ca fundal imagini pentru butoane din pachetul de interfață grafică [8], iar pistolul, arma automată și pușca sunt preluate din alt pachet [10], imaginea pumnului fiind preluată din altă sursă [9].



Figura 2.4.3.2: Interfață grafică în joc

#### 2.4.4 Powerup-uri

La un număr de inamici eliminați, pe hartă apar obiecte de tip power-up, care atunci când sunt folosite de jucător prin intrarea în contact cu ele, acesta primește anumite ajutoare: muniție în plus, viață, viteză de mișcare pentru o perioadă, putere mai mare a armelor pentru o anumită perioadă. Acestea sunt reprezentate prin imagini care fac ușor de înțeles efectul folosirii fiecărui și au fost preluate de la același autor al imaginilor din secțiunile precedente [6]. Powerup-urile din următoarea figură sunt următoarele, în ordine de la stânga la dreapta: viață, viteză, damage, gloanțe.



Figura 2.4.4.1: Imaginele folosite pentru powerup-uri

# Capitolul 3

## Implementarea jocului

În acest capitol vor fi detaliate codurile folosite în acest joc. Nu vor fi prezentate părțile care se repetă, precum inițializări de variabile, inițializări ale altor obiecte prin căutare sau în editor.

```
void Start()
{
    weapon.parent = GameObject.FindGameObjectWithTag("EnemyBullets");
    player = GameObject.FindGameObjectWithTag("Player");
    playerScoreManager = GameObject.FindGameObjectWithTag("GameController") // 
        .GetComponent<GameController>(); // GameController
    gameObject.GetComponent<AIDestinationSetter>().target = player.transform;
    currentHealth = maxHealth;
    healthBar.SetMaxHealth(maxHealth);
}
```

Figura 3.0.0.1: Inițializarea unui obiect Enemy2

### 3.1 Player

Jucătorul controlează un GameObject de tipul Player, care are atașate un Collider2D, un Rigidbody2D și script-ul PlayerController.

#### 3.1.1 Caracteristici

Player-ul are mai multe variabile care îl definesc:

- Health și maxHealth: reprezintă punctele de viață curente și punctele de viață maxime;
- moveSpeed și defaultSpeed: reprezintă viteza de mișcare curentă a caracterului și viteza de bază a caracterului selectat;

- damageMultiplier: reprezintă un multiplicator al damage-ului produs de armele caracterului;
- weaponsEnabled: indică ce arme pot fi folosite de caracter.

### 3.1.2 Mișcarea

Pentru mișcarea caracterului, se preiau input-urile pe axa orizontală și verticală, care sunt preluate de la apăsarea tastelor W, A, S, D, respectiv săgețile sus, stânga, jos, dreapta. Folosind vectorul pe 2 dimensiuni generat de aceste input-uri, putem folosi funcția MovePosition pentru a modifica poziția caracterului. Adăugăm la poziția actuală, care este tot un vector pe 2 dimensiuni, vectorul de mișcare normalizat, scalat cu viteza de mișcare definită pentru caracter și cu timpul, iar astfel caracterul se mișcă.

Pentru rotația caracterului, se preiau într-un vector pe 2 dimensiuni, mousePosition, coordonatele din plan, acestea fiind convertite din coordonatele folosite de cameră folosind funcția ScreenToWorldPoint. Vectorul mousePosition este folosit pentru a calcula alt vector pe 2 dimensiuni, aimDirection, prin scăderea vectorului de poziție a caracterului din vectorul mousePosition. Astfel, prin scăderea celor 2 vectori, obținem un vector care arată de la unul către celalalt. Folosind vectorul aimDirection și funcția Atan2 [14], care returnează un unghi în radiani, transformat mai apoi în grade scăzut cu un offset de 90 de grade pentru a se potrivi cu poziția de start a caracterului. Acest concept de mișcare a fost inspirat de un videoclip de pe internet [2].

```
movement.x = Input.GetAxisRaw("Horizontal");
movement.y = Input.GetAxisRaw("Vertical");
moveDirection.x = movement.x;
moveDirection.y = movement.y;
mousePosition = cam.ScreenToWorldPoint(Input.mousePosition);
rb.MovePosition(rb.position + moveDirection.normalized * moveSpeed
                * Time.deltaTime);
Vector2 aimDirection = mousePosition - rb.position;
float aimAngle = Mathf.Atan2(aimDirection.y, aimDirection.x)
    * Mathf.Rad2Deg - 90f;
rb.rotation = aimAngle;
```

Figura 3.1.2.1: Codul pentru mișcare și rotație

### 3.1.3 Folosirea armelor

În acest proiect, jucătorul poate folosi 4 arme, în funcție de caracterul ales. Aceste arme sunt:

1. Pumni: când aceștia sunt selectați, inamicii nu pot fi atacați, însă jucătorul se mișcă mult mai rapid când folosește pumnii.

2. Pistol: acesta are gloanțe infinite și este echilibrat și ca viteză de tras și ca damage dat inamicilor.
3. SMG: acesta este un pistol-mitralieră, deci trage mai repede decât pistolul, iar damage-ul este doar puțin mai mare. Este singura armă care are foc automat.
4. Pușcă: aceasta este o armă bună pentru inamicii care sunt foarte aproape de jucător, având damage foarte mare, însă trage lent și face și jucătorul să se miște foarte lent cât o folosește. Gloanțele dispar după o rază, deoarece o pușcă este o armă eficientă doar la apropiere.

Acstea pot fi selectate cu butoanele 1, 2, 3, 4. Caracteristicile armelor vor fi descrise în detaliu în capitolul dedicat lor, însă jucătorul mai are și o variabilă *damageMultiplier*, care modifică damage-ul dat de arme ca procent.

### 3.1.4 Bara de viață

Pentru a putea să cunoască câtă viață mai are caracterul jucătorului, acesta are mereu o bară de viață deasupra caracterului și numărul de puncte de viață rămase deasupra acestei bare. Această bară scade de la dreapta la stânga, și își schimbă culoarea în funcție de viață rămasă:

- verde, între 100% și 50% din viață;
- galben, între 50% și 20% din viață;
- roșu, sub 20% din viață.

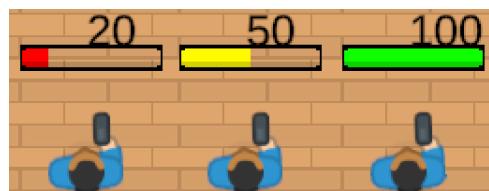


Figura 3.1.4.1: Bara de viață a jucătorului

Pentru a schimba culoarea în funcție de viață rămasă, schimbăm parametrul *color* cu funcția *Evaluate* din clasa *Gradient*, care schimbă culoarea în funcție de procentul de viață rămasă, calculat prin împărțirea vieții curente la viață maximă. Apoi, umplerea acesteia se face prin setarea procentului de umplere cu procentul de viață rămasă.

```
fill.color = gradient.Evaluate(time: currentHp / maxHp);
fill.fillAmount = currentHp / maxHp;
```

Figura 3.1.4.2: Codul pentru bara de viață

## 3.2 Arme

### 3.2.1 Caracteristici

O armă este definită de mai multe variabile:

- fireForce: reprezintă forța cu care gloanțele sunt trase din armă;
- firerate, reloadTime: reprezintă rata de tras a armei, respectiv timpul pentru reîncărcarea armei;
- magAmmo, ammoCapacity, maxAmmo: reprezintă numărul de gloanțe în încărcătorul armei, respectiv capacitatea unui încărcător și numărul maxim de gloanțe;
- infiniteAmmo: variabila care definește dacă arma are număr infinit de gloanțe;
- damage: reprezintă cate puncte de viață îi sunt scăzute celui lovit de glonț;
- hasRange: dacă este adevărat, gloanțele dispar după o perioadă de timp.

### 3.2.2 Tragerea

Înainte de a trage glonțul, se verifică mai multe condiții:

- Dacă arma nu este în proces de reîncărcare;
- Dacă nu s-a tras un glonț în mai puțin timp decât rata de tragere;
- Dacă în încărcător există gloanțe;
- Dacă arma nu are un număr infinit de gloanțe, se scade un glonț din încărcător;
- Dacăarma nu este pușcă.

Pentru a nu putea trage mai repede decât se dorește, se folosește variabila *canShoot*, care este adevarată atunci când arma poate trage, adică poate instanția un glonț și falsă cât se scurge timpul precizat în variabila *firerate*, care reprezintă timpul dintre tragerea gloanțelor. După ce se trage un glonț, *canShoot* devine fals și se apelează corutina *FireWait*, care așteaptă *firerate* secunde, iar apoi face variabila *canShoot* adevarată. Continutul *if-else*-ului este descris în continuare.

```
public void Fire()
{
    if (reloading == false)
    {
        if(canShoot)
        {
            if (magAmmo > 0)
            {
                if (infiniteAmmo == false)
                    magAmmo--;

                if (isShotgun == false)...
                else...
                canShoot = false;
                StartCoroutine(routine:FireWait(firerate));
            }
        }
    }
}

IEnumerator FireWait(float time)
{
    yield return new WaitForSeconds(time);
    canShoot = true;
}
```

Figura 3.2.2.1: Funcția pentru tras un glonț și corutina pentru așteptat între trageri

Pentru a trage un glonț, se creează un GameObject de tipul *Bullet*, căruia îi sunt modificate mai multe atrbute:

- Poziția de plecare a glonțului devine capătul armei;
- Pentru pușcă, variabila *hasRange* este *true*, pentru celelalte arme este *false*;
- Damage-ul dat de glonț este setat ca produsul dintre variabilele *damage* și *damageMultiplier*.

La final, se adaugă o forță glonțului, care este tras astfel.

```
GameObject bullet = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation,
parent.transform);
bullet.GetComponent<Bullet>().startingPosition = firePoint.position;
bullet.GetComponent<Bullet>().hasRange = true;
bullet.GetComponent<Bullet>().defaultDamage = damage * damageMultiplier;
bullet.GetComponent().AddForce(~firePoint.up * fireForce, ForceMode2D.Impulse);
```

Figura 3.2.2.2: Tragere glonț

În cazul în carearma este pușcă, se trag 3 gloanțe, unul drept spre țintă, și două cu un offset de 5, respectiv -5 grade, folosindu-se următorul cod în loc de ultima linie de cod din figura 3.2.2.2.

```
bullet2.GetComponent<Rigidbody2D>().AddForce(~Quaternion.Euler(x: 0f, y: 0f, z: 5f)
* firePoint.up * fireForce, ForceMode2D.Impulse);
```

Figura 3.2.2.3: Tragere glonț pușcă

### 3.2.3 Reîncărcarea armei

Pentru reîncărcarea armei, se verifică în primul rând dacă arma are un număr infinit de gloanțe. Dacă nu are, se verifică dacă mai sunt destule gloanțe pentru un încărcător complet, iar dacă nu sunt destule, toate gloanțele rămase sunt mutate în încărcător. Dacă mai erau destule gloanțe pentru un încărcător, gloanțele rămase sunt calculate ca diferența dintre gloanțele totale și numărul de gloanțe introduse în încărcător. În final, se apelează o funcție care așteaptă un timp dat, reloadTime, înainte ca reîncărcarea armei să fie finalizată, pentru a adăuga realism.

```
if (ammoCapacity > totalAmmo)
    magAmmo = totalAmmo;

else
{
    totalAmmo = totalAmmo - ammoCapacity + magAmmo;
    magAmmo = ammoCapacity;
}
StartCoroutine(routine: ReloadWait(reloadTime));
```

Figura 3.2.3.1: Cod reîncărcare armă

Pe toată perioada câtarma este reîncărcată, jucătorul nu poate schimba arma sau trage, iar vizual, imaginea caracterului se schimbă pe parcursul reîncărcării, pentru a da un semnal vizual clar pentru perioada câtarma este reîncărcată, iar după finalizarea reîncărcării armei, imaginea caracterului se întoarce la cea în carearma este îndreptată spre inamici.



Figura 3.2.3.2: Imagini reîncărcare arme

## 3.3 Glonțul

Un glonț este definit de variabilele:

- defaultDamage și damage: damage-ul pe care glonțul îl dă în mod implicit, respectiv damage-ul pe care glonțul îl dă.
- despawnDistance: distanța fată de punctul de plecare pe care trebuie să o parcurgă glonțul până să dispară; variabila folosită de gloanțele de pușcă;
- startingPosition: vector care reține poziția de plecare a glonțului;

Pentru a verifica dacă glonțul a lovit un inamic sau un obstacol, se folosesc funcțiile OnTriggerEnter2D și OnCollisionEnter2D. Dacă glonțul este tras de jucător, atunci când intră în coliziune cu un inamic, accesează scriptul inamicului respectiv și apelează o funcție care îi dă damage-ul definit pe glonț inamicului. Dacă glonțul este tras de un inamic, se procedează la fel pentru a îi da damage jucătorului. După aceste lucruri, glonțul este distrus.

Glonțul poate să fie distrus și atunci când nu lovește un inamic: dacă lovește un obstacol, dacă devine invizibil sau dacă gloanțele sunt trase de o pușcă, caz în care au range, adică dispar după o distanță parcursă.

```
void Update()
{
    float dist = Vector3.Distance(a.startingPosition,
        b.gameObject.transform.position);
    if(dist>=despawnDistance && hasRange)
        Destroy(gameObject);
}

// Unity Message | 0 references
private void OnBecameInvisible()
{
    Destroy(gameObject);
}

// Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    Debug.Log(collision.gameObject.tag);
    if (collision.gameObject.CompareTag("Obstacle"))
    {
        Destroy(gameObject);
    }
}
```

Figura 3.3.0.1: Distrugere gloanțe

## 3.4 Inamici

### 3.4.1 Spawner Inamici

Inamicii sunt spawnați pe hartă folosind un obiect gol, la care este atașat script-ul de instanțiere al inamicilor. În acesta sunt specificate ca variabile coordonatele limită, x-ul minim și maxim, y-ul minim și maxim, și următoarele variabile:

- spawnRate: reprezintă rata de apariție a inamicilor;
- enemyMaxNumber, enemiesSpawned: numărul de inamici care trebuie instanțiați în acest nivel, este -1 dacă nivelul este unul pe timp, respectiv numărul de inamici instanțiați pana în acel moment;
- allEnemiesSpawned: variabilă booleană care este true atunci când toți inamicii au fost instanțiați, dacă este cazul. Este una dintre condiții, este folosită pentru a verifica dacă nivelul a fost finalizat.

- enemyPrefab: obiectele inamicilor sunt adăugate în aceste variabile.

Codul de mai jos este în funcția Update, pentru a se verifica în fiecare cadru dacă condițiile pentru a spawna un inamic sunt întrunite: dacă a trecut destul timp de la ultima instanțiere. Folosind coordonatele limită, generăm la întâmplare un set de coordonate, iar folosind grid-ul generat de obiectul A\*, dacă poziția de spawn este într-un loc considerat obstacol pe grid, găsim cea mai apropiată locație care nu este obstacol, iar în acel loc instanțiem inamicul. Pentru a alege tipul de inamic, se alege un număr la întâmplare, iar fiecare inamic are anumite numere care îl instantiază, unii având astfel șanse mai mari de apariție decât ceilalți. Pentru inamicii cu arme, trebuie atribuită arma la acel inamic, pentru a o putea folosi. De asemenea, spawnerul este distrus dacă player-ul este distrus, adică dacă pierde, pentru a evita anumite erori legate de atribuirile cu obiecte care nu mai există.

```

if(player.IsDestroyed())
    gameObject.SetActive(false);
timer += Time.deltaTime;
if(timer > spawnRate &&
   (enemiesSpawned < enemyMaxNumber || enemyMaxNumber == -1))
{
    timer = 0;
    enemiesSpawned++;
    SpawnEnemy();
}
if(enemiesSpawned >= enemyMaxNumber)
    allEnemiesSpawned = true;

```

Figura 3.4.1.1: Codul pentru instanțierea inamicilor

### 3.4.2 Damage și eliminarea inamicilor

Toți inamicii urmăresc jucătorul folosind pathfinding-ul A\* descris în secțiunea 2.4.1. Imediat ce aceștia sunt spawnati pe hartă, ei primesc jucătorul ca țintă și încep urmărirea. Aceștia au mai multe variabile în comun:

- currentHealth, maxHealth: reprezintă viața curentă, respectiv viața maximă;
- moveSpeed: reprezintă viteza de miscare;
- pointAdded: o variabilă de tip bool, care este folosită pentru a verifica dacă a fost adăugat un punct la scorul jucătorului atunci când inamicul a fost eliminat.

Pentru a primi damage, atunci când este lovit de un glonț, glonțul accesează scriptul inamicului lovit de acesta și apelează funcția *TakeDamage*, cu damage-ul specificat în variabilele glonțului. Astfel, fiecare glonț are un damage specific care nu poate fi schimbat odată ce este tras. Am preferat această metodă deoarece puteau exista bug-uri atunci când se folosea și un buff de tipul *damage multiplier*, sau când se schimbă arma în timp ce glonțul este în zbor, iar damage-ul acestuia să se schimbe cu cel de pe arma nouă.

```

public void TakeDamage(float damage)
{
    currentHealth -= damage;
    healthBar.SetHealth(currentHealth);
}

```

Figura 3.4.2.1: Funcția prin care inamicul primește damage

În funcția *Update*, apelată în fiecare cadru, se verifică dacă inamicul mai are viață rămasă. În caz contrar, viteza de mișcare a inamicului devine 0. Pentru a evita situații neprevăzute, obiectul este distrus, și dacă nu se adăugase un punct la scorul jucătorului, se executa și acest lucru.

```

if (currentHealth <= 0)
{
    movementSpeed = 0.0f;
    Destroy(gameObject.transform.parent.gameObject);
    if (pointAdded == false)
    {
        playerScoreManager?.AddPoint();
        pointAdded = true;
    }
}

```

Figura 3.4.2.2: Eliminarea inamicului

### 3.4.3 Zombie

Acest inamic este mai rapid și are mai multă viață, însă nu are nici o armă, ci se apropie de jucător și îl lovește. Variabilele specifice acestui inamic sunt:

- damage, damageRate: reprezintă damage-ul pe care acesta îl dă jucătorului la fiecare atac, respectiv rata atacurilor, în secunde;
- canHit: variabilă bool care determină dacă a trecut durata de timp între lovitură, durată specificată în variabila damageRate. Dacă este true, va ataca jucătorul atunci când este destul de aproape de el;



Figura 3.4.3.1: Modelul caracterului zombie

Funcția *OnCollisionStay2D* se activează în fiecare cadru în care un alt collider atinge colliderul acestui inamic. O dată la *damageRate* secunde, acesta îi dă damage jucătorului. Acest lucru se face folosind o funcție *IEnumerator* care, după un număr dat de secunde, modifică variabila booleană *canHit* în true și dă damage jucătorului, iar apoi variabila devine false, până la trecerea timpului între lovitură.

```

void OnCollisionStay2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player") && canHit)
    {
        collision.gameObject.GetComponent<PlayerController>().TakeDamage(damage);
        canHit = false;
        StartCoroutine(routine: WaitBetweenHits(damagerate));
    }
}

IEnumerator WaitBetweenHits(float time)
{
    yield return new WaitForSeconds(time);
    canHit = true;
}

```

Figura 3.4.3.2: Funcția prin care dă damage jucătorului

### 3.4.4 Robot

Acest inamic este mai lent și are mai puțină viață, însă are o armă cu care trage de la distanță către jucător. Variabilele specifice sunt:

- weapon: variabilă de tip *Weapon*, unde se adaugă arma atașată inamicului;
- rotationSpeed: viteza cu care inamicul se rotește spre jucător.
- accuracy: variabilă float cu valori cuprinse între 0 și 1; cu cât este mai aproape de 1, cu atât inamicul va trage doar atunci când arma este îndreptată fix înspre jucător. Totuși, acest procent nu arată precizia reală, acesta îmbunătățind precizia doar atunci când jucătorul se mișcă și este la o distanță medie față de inamic.



Figura 3.4.4.1: Modelul caracterului robot

Pentru a folosi arma, inamicul trebuie să ajungă la o distanță specificată în scriptul *AIPath*, care este atașat obiectului. Odată ce ajunge la această distanță, variabila bool *reachedEndOfPath* este true și inamicul poate trage spre jucător. Variabila *dot* este initializată cu rezultatul funcției *Dot* dintre diferența dintre vectorul poziției jucătorului și vectorul poziției inamicului, care reprezintă un vector care arată de la unul către celalalt, și vârful armei inamicului. Rezultatul acestei funcții este 1 dacăarma inamicului este îndreptată spre jucător și -1 dacăarma inamicului este îndreptată opus față de jucător. Atunci când acesta este cu arma îndreptată spre jucător cu un offset dat ca variabila *accuracy*, va trage spre jucător. Rotația se execută în mod asemănător cu cea a jucătorului, însă se folosește funcția *Slerp* pentru a avea o tranziție mai fină între rotația inițială și cea dorită în final. Funcția *Quaternion* returnează o rotație care rotește z grade în jurul axei z, x grade în jurul axei x și y grade în jurul axei y, în această ordine [16].

```

void FixedUpdate()
{
    if (aiPath.reachedEndOfPath)
    {
        float dot = Vector2.Dot(lhs: transform.up.normalized,
                               rhs: (player.transform.position - transform.position).normalized);
        Rotate();
        if (dot > accuracy)
        {
            weapon.Fire();
        }
    }
}

private void Rotate()
{
    Vector2 targetDirection = player.transform.position - transform.position;
    float angle = Mathf.Atan2(targetDirection.y, targetDirection.x) * Mathf.Rad2Deg - 90f;
    Quaternion q = Quaternion.Euler(new Vector3(x: 0, y: 0, z: angle));
    transform.localRotation = Quaternion.Slerp(a: transform.localRotation, b: q, rotationSpeed);
}

```

Figura 3.4.4.2: Rotația inamicului

### 3.4.5 Asasinul

Acesta are aceleși variabile specifice ca robotul, descris în secțiunea anterioară, aceste caractere folosind același script:

- weapon: variabilă de tip *Weapon*, unde se adaugă arma atașată inamicului;
- rotationSpeed: viteza cu care inamicul se rotește spre jucător.
- accuracy: variabilă float cu valori cuprinse între 0 și 1; cu cât este mai aproape de 1, cu atât inamicul va trage doar atunci când arma este îndreptată fix înspre jucător. Totuși, acest procent nu arată precizia reală, acesta îmbunătățind precizia doar atunci când jucătorul se mișcă și este la o distanță medie față de inamic.



Figura 3.4.5.1: Modelul caracterului asasin

Acest inamic pare să fie uman, însă nu se știe clar; acesta este mai puternic decât ceilalți inamici, mai rapid, iar arma acestuia este o pușcă, deci trebuie să se apropie de jucător pentru a îl folosi, însă dacă îl nimerește pe jucător, daunele vor fi mari. Acesta funcționează în același mod ca inamicul robot, descris în secțiunea anterioară, însă în codul armei lui are variabila booleană *isShotgun* adevărată, ceea ce face arma să tragă 3 gloanțe. Celor două gloanțe adiționale le este adăugată aceeași forță care le trimit spre inamic, însă direcția inamicului este modificată cu 5, respectiv -5 grade, pentru a simula tragerea unei arme de acest tip.

```

GameObject bullet = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation,
    parent.transform);
bullet.GetComponent<Bullet>().startingPosition = firePoint.position;
bullet.GetComponent<Bullet>().hasRange = true;
bullet.GetComponent<Rigidbody2D>().AddForce(firePoint.up * fireForce, ForceMode2D.Impulse);
GameObject bullet2 = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation,
    parent.transform);
bullet2.GetComponent<Bullet>().startingPosition = firePoint.position;
bullet2.GetComponent<Bullet>().hasRange = true;
bullet2.GetComponent<Bullet>().defaultDamage = damage * damageMultiplier;
bullet2.GetComponent<Rigidbody2D>().AddForce(Quaternion.Euler(x: 0f, y: 0f, z: 5f) * firePoint.up
    * fireForce, ForceMode2D.Impulse);
GameObject bullet3 = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation,
    parent.transform);
bullet3.GetComponent<Bullet>().startingPosition = firePoint.position;
bullet3.GetComponent<Bullet>().hasRange = true;
bullet3.GetComponent<Bullet>().defaultDamage = damage * damageMultiplier;
bullet3.GetComponent<Rigidbody2D>().AddForce(Quaternion.Euler(x: 0f, y: 0f, z: -5f) * firePoint.up
    * fireForce, ForceMode2D.Impulse);

```

Figura 3.4.5.2: Trasul inamicului 3

### 3.4.6 Bara de viață

Pentru a putea să cîtă viață mai are inamicul, acesta are mereu o bară de viață deasupra lui. Această bară scade de la dreapta la stânga și își schimbă culoarea în funcție de viață rămasă:

- verde, între 100% și 50% din viață;
- galben, între 50% și 20% din viață;
- roșu, sub 20% din viață.

Această bară funcționează la fel cum funcționează bara de viață a jucătorului, descrisă în secțiunea 3.1.4.



Figura 3.4.6.1: Bara de viață a inamicului

## 3.5 Powerup-uri

Powerup-urile reprezintă obiecte care, atunci când sunt ridicate, aduc diferite bonusuri jucătorului. Acestea apar atunci când se atinge multiplul unui scor stabilit la crearea nivelului, care poate fi modificat pentru a schimba dificultatea jocului, deoarece aceste powerup-uri sunt foarte utile și aduc multe beneficii jucătorului, scăzând din dificultate. Acestea sunt spawnate folosind un script în care sunt specificate ca variabile coordonatele limită, x-ul minim și maxim, y-ul minim și maxim și multiplul scorului la care să apară acestea. Codul de mai jos este în funcția *Update*, pentru a se verifica în fiecare cadru dacă

condițiile pentru a instanția un powerup sunt întrunite: dacă scorul curent este divizibil cu numărul dat ca variabilă, dacă scorul este mai mare decât 0, pentru a nu spawna un powerup când scorul este 0 și dacă scorul s-a modificat, folosind o altă variabilă *oldScore* care se modifica atunci când se spawnează un powerup, pentru a nu instanția mai multe powerup-uri la același scor.

Folosind coordonatele limită, generăm la întâmplare un set de coordonate, iar folosind grid-ul generat de obiectul A\*, dacă poziția de apariție este într-un loc considerat obstacol de grid-ul generat de A\*, găsim cea mai apropiată locație care nu este obstacol, iar în acel loc spawnăm powerup-ul. Pentru a alege tipul de powerup, se alege un număr la întâmplare, fiecare număr reprezentând un tip de powerup, care este apoi spawnat. Odată cu apariția unui powerup aleator, mai este instantiat de fiecare dată și un powerup de muniție.

```
if (controller.score % scoreMultiple == 0 && controller.score > 0 && controller.score != oldScore)
{
    hasSpawned = true;
    oldScore = controller.score;
    Vector3 randomSpawnPos = new Vector3(x: Random.Range(x_max, x_min), y: Random.Range(y_max, y_min));
    randomSpawnPos = (Vector3)AstarPath.active.data.graphs[0].GetNearestForce(randomSpawnPos,
        NNConstraint.Default).node.position; // Int3
    int nr = Random.Range(1, 4);
    if (nr == 1)
    {
        Instantiate(powerup1prefab, randomSpawnPos, Quaternion.identity);
```

Figura 3.5.0.1: Spawnarea powerup-urilor

Variabilele din acest script de instanțiere a powerup-urilor sunt:

- scoreMultiple: multiplii de scor la care apare un nou powerup;
- oldScore: scorul precedent, folosit pentru a verifica dacă a apărut un powerup la scorul curent;
- x\_min, x\_max, y\_min, x\_max: coordonatele limită ale instanțierii inamicilor pe câmpul de joc;
- powerupprefab: variabile GameObject în care se adaugă în editor obiectele de tip *Powerup*.

```

void OnTriggerEnter2D(Collider2D collider)
{
    if (collider.CompareTag("Player"))
    {
        Pickup();
    }
}

1 reference
void Pickup()
{
    playerController.AddHealth(health);
    Destroy(gameObject);
}

```

Figura 3.5.0.2: Ridicarea powerup-urilor

Aceste powerup-uri pot fi văzute în figura 2.4.4.1.

### 3.5.1 Heal Powerup

Acesta folosește funcția *OnTriggerEnter2D*, verifică tipul obiectului care a intrat în zona de trigger a acestui powerup, iar dacă acel obiect este jucătorul, se apelează funcția Pickup, care îi adaugă jucătorului viață cât este precizat în variabila *health*. Dacă suma dintre viață curentă și viață maximă a jucătorului este mai mare decât *maxHealth*, viață curentă va deveni *maxHealth*, nu suma celor două.

```

public void AddHealth(float hp)
{
    if (currentHealth + hp > maxHealth)
        currentHealth = maxHealth;
    else
        currentHealth += hp;
}

```

Figura 3.5.1.1: Cod funcție bonus viață

### 3.5.2 Damage Multiplier Powerup

Acesta procedează la fel ca powerupul de heal, doar că are două variabile, *multiplier*, care este numărul cu care damage-ul dat de player este înmulțit și *buffTime*, care reprezintă timpul cât damage-ul este multiplicat. Funcția din script-ul jucătorului care este apelată este *SetDamageMultiplier*. Aceasta apelează un *IEnumerator* care modifică variabila *damageMultiplier* pentru timpul dat; se mai folosește o variabilă auxiliară pentru a memora multiplicatorul inițial.

```

public void SetDamageMultiplier(float multiplier, float time)
{
    StartCoroutine(routine: SetMultiplier(multiplier, time));
}

1 reference
IEnumerator SetMultiplier(float multiplier, float time)
{
    float oldMultiplier = damageMultiplier;
    damageMultiplier = multiplier;
    yield return new WaitForSeconds(time);
    damageMultiplier = oldMultiplier;
}

```

Figura 3.5.2.1: Cod funcții damage multiplier

### 3.5.3 Speed Powerup

Acesta procedează la fel ca powerupul de damage multiplier, cu două variabile, *speed*, care este viteza player-ului pe perioada de activitate a powerup-ului și *speedBuffTime*, care reprezintă timpul cât viteza este multiplicată. Funcția din script-ul jucătorului care este apelată este *SetMoveSpeed*. Aceasta apelează un *IEnumerator* care modifică variabila *moveSpeed* pentru timpul dat; se mai folosește o variabilă auxiliară pentru a memora dacă viteza este modificată sau nu.

```

public void SetMoveSpeed(float speed, float time)
{
    StartCoroutine(routine: SetSpeed(speed, time));
}

1 reference
IEnumerator SetSpeed(float speed, float time)
{
    newSpeed = speed;
    speedBuff = true;
    yield return new WaitForSeconds(time);
    speedBuff = false;
}

```

Figura 3.5.3.1: Cod funcții bonus viteza

### 3.5.4 Ammo Powerup

Acesta reumple stocul de gloanțe al jucătorului: în fiecare armă a jucătorului este reținut numărul inițial de gloanțe din încărcătorul armei și numărul total de gloanțe ale armei. Folosindu-le pe acestea, setăm numărul de gloanțe ale armei ca aceste numere reținute.

```

void OnTriggerEnter2D(Collider2D collider)
{
    if (collider.CompareTag("Player"))
    {
        Pickup();
    }
}
1 reference
void Pickup()
{
    foreach (Weapon wep in playerController.weapons)
    {
        wep.totalAmmo = wep.maxAmmo;
        wep.magAmmo = wep.ammoCapacity;
    }
    Destroy(gameObject);
}

```

Figura 3.5.4.1: Cod powerup adăugare muniție

## 3.6 Game Manager, Audio Manager, Game Controller

Acstea 3 obiecte, fiecare cu script-ul cu același nume atașat, se ocupă cu funcționarea jocului în toate momentele sale - *Game Manager* se ocupă cu memorarea caracterului selectat, memorarea nivelurilor deblocate și transmiterea de informații între *scene*, *Audio Manager* se ocupă cu transmiterea sunetelor pe toată durata rulării jocului; atât muzica de fundal, cât și sunetele din niveluri. Fiecare nivel, precum și cel *endless*, are un *GameController*, care se ocupă cu detaliile specifice fiecărui nivel - numărul de inamici care trebuie instantiați, numărul de inamici deja instantiați, scorul jucătorului, verificarea timpului trecut - dacă a trecut timpul stabilit, nivelul este completat, verificarea condițiilor ca acel nivel să fie completat, executarea pauzei atunci când jucătorul apasă butonul de pauză, apariția ecranelor de victorie și înfrângere, în funcție de caz, și modificarea tuturor lucrurilor necesare din nivel în aceste situații.

### Game Manager

Având în vedere sarcinile acestui obiect, ne dorim să existe o singură instanță a acestuia pe toată rularea jocului. Astfel că, în *Awake* verificăm că există doar un Game Manager, iar dacă deja există unul, acest obiect nu mai este necesar și este distrus. Dacă nu există alt Game Manager, îi adaugăm condiția *DontDestroyOnLoad*, ca acesta să nu fie distrus când schimbam *scena*. Apoi, preluăm numărul ultimului nivel deblocat, acesta fiind salvat, și folosind funcția *AddLevel*, deblocăm toate nivelurile până la acesta.

```

void Awake()
{
    GameObject[] objs = GameObject.FindGameObjectsWithTag("GameManager");

    if (objs.Length > 1)
    {
        Destroy(gameObject);
    }

    DontDestroyOnLoad(gameObject);
    lastLevelUnlocked = PlayerPrefs.GetInt(key: "lastLevelUnlocked");
    if (lastLevelUnlocked > 1)
    {
        for (int i = 2; i <= lastLevelUnlocked; i++)
        {
            AddLevel(i);
        }
    }
}

```

Figura 3.6.0.1: Inițializare GameManager

Acstea două funcții sunt foarte similare, fiind diferit doar contextul în care sunt folosite, astfel că sunt necesare câteva verificări diferite. Se creează o listă cu numerele nivelurilor deja deblocate la care se adaugă noul nivel deblocat, adică ultimul nivel blocat. Dacă toate nivelurile au fost deblocate, se creează o listă goală pentru nivelurile blocate, pentru a nu avea erori. Această adăugare a nivelului în listă se face într-o listă auxiliară, iar la final lista auxiliară este transferată la lista de niveluri deblocate, respectiv la lista de niveluri blocate. *AddCompleteLevel* verifică dacă nivelul completat este ultimul în lista celor deblocate. Dacă da, înseamnă că trebuie deblocat nivelul următor, iar în memorie este salvat noul nivel ca ultimul nivel deblocat. *AddLevel* ia ultimul nivel blocat și îl adaugă în lista celor deblocate.

```

public void AddCompleteLevel(int levelNr)
{
    if (unlockedLevels[unlockedLevels.Length - 1] == levelNr)
    {
        lastLevelUnlocked = levelNr + 1;
        PlayerPrefs.SetInt("lastLevelUnlocked", lastLevelUnlocked);
        int unlockedLength = unlockedLevels.Length + 1;
        int[] aux = new int[unlockedLength];
        for (int i = 0; i < unlockedLevels.Length; i++)
            aux[i] = unlockedLevels[i];
        aux[unlockedLength] = lockedLevels[0];
        unlockedLevels = new int[unlockedLength];
        unlockedLevels = aux;
        if (lockedLevels.Length <= 1)
            lockedLevels = new int[0];
    }
    else
    {
        int lockedLength = lockedLevels.Length - 1;
        aux = new int[lockedLength];
        for (int i = 1; i < lockedLevels.Length; i++)
            aux[i - 1] = lockedLevels[i];
        lockedLevels = new int[lockedLength];
        lockedLevels = aux;
    }
}

public void AddLevel(int levelNr)
{
    int unlockedLength = unlockedLevels.Length + 1;
    int[] aux = new int[unlockedLength];
    for (int i = 0; i < unlockedLevels.Length; i++)
        aux[i] = unlockedLevels[i];
    aux[unlockedLength] = lockedLevels[0];
    unlockedLevels = new int[unlockedLength];
    unlockedLevels = aux;
    if (lockedLevels.Length <= 1)
        lockedLevels = new int[0];
    else
    {
        int lockedLength = lockedLevels.Length - 1;
        aux = new int[lockedLength];
        for (int i = 1; i < lockedLevels.Length; i++)
            aux[i - 1] = lockedLevels[i];
        lockedLevels = new int[lockedLength];
        lockedLevels = aux;
    }
}

```

Figura 3.6.0.2: Cod funcții deblocare niveluri

## Audio Manager

La fel ca Game Manager, este nevoie ca acest obiect să fie unic și să fie activ în permanentă, pentru a se ocupa de toată partea de audio a jocului. Astfel că este instantiat în același fel ca celălalt manager. Aceasta a fost dezvoltat cu inspirație de la un videoclip de pe internet [1].

```
private void Awake()
{
    if (Instance != null)
    {
        Destroy(gameObject);
        return;
    }

    Instance = this;
    DontDestroyOnLoad(gameObject);
    source = GetComponent< AudioSource >();
    copySounds = sounds;
}
```

Figura 3.6.0.3: Inițializare AudioManager

În funcția *Update*, care se apelează în fiecare cadru, se verifică dacă pe sursa audio de muzică se difuzează sunet. Dacă nu, apelăm funcția de redare a sunetelor în mod aleator, *PlayRandom*. Această funcție ia toată lista de melodii disponibile, face o copie a acestora, și redă unul aleator. După ce redă melodia, o șterge din listă și repetă acest lucru până când copia este goală. Acest lucru se repetă la infinit, astfel având muzică de fundal permanent și în mod aleator pentru a nu plăcăti jucătorul. Toată muzica are licență corespunzătoare pentru a putea fi utilizată în acest joc.

```
void Update()
{
    if (copySounds.Length == 0)
    {
        copySounds = sounds;
    }

    if (!source.isPlaying)
    {
        PlayRandom();
    }
}

1 reference
public void PlayRandom()
{
    source.outputAudioMixerGroup = musicMixerGroup;
    Sound s = copySounds[Random.Range(0, copySounds.Length)];
    source.clip = s.clip;
    source.Play();
    copySounds = copySounds.Where(snd : Sound => snd != s).ToArray();
}
```

Figura 3.6.0.4: Cod play muzică fundal

Clasa de tip *Sound* este o clasă creată pentru a putea modifica în editor lista de melodii, ca volum, ca tonalitate sau ca ordine de derulare. Aceasta are un *AudioClip*, care reprezintă clasa din Unity care conține melodia, și mai multe setări, ca volum sau tonalitate, pentru a putea fi echilibrate, astfel încât o melodie să nu fie mult mai tare decât celelalte în joc, de exemplu.

```

public class Sound
{
    public string name;
    public AudioClip clip;
    [Range(0f, 1f)] public float volume;
    [Range(.1f, 3f)] public float pitch;
    [HideInInspector] public AudioSource source;
}

```

Figura 3.6.0.5: Clasa Sound

## Game Controller

Acesta se ocupă cu inițializarea tuturor obiectelor necesare pentru funcționarea unui singur nivel, fiecare nivel având un game controller cu setările specifice. În funcție de tipul de nivel setat în editor pentru nivel, *enemyKillVictory*, eliminarea tuturor inamicilor care se spawnează sau *timedVictory*, victoria după ce jucătorul rezistă o perioadă de timp, spawnerul de inamici primește numărul de inamici care trebuie instantiați, respectiv -1, care îi indică spawnerului că trebuie instantiați un număr infinit de inamici. Apoi, sunt preluate din Game Manager informații despre caracterul selectat de jucător. Informațiile sunt folosite pentru a seta obiectul *Player*. Pentru a activa armele, se verifică numărul de gloanțe astfel: dacă arma are mai mult de 0 gloanțe, jucătorul poate folosi acea armă, iar arma primește numărul de gloanțe conform selecției caracterului. Dacăarma are -1 gloanțe setate, înseamnă că aceasta are un număr infinit de gloanțe, iar dacăarma are 0 gloanțe, înseamnă că jucătorul nu poate folosi acea armă, așa că este dezactivată.

```

if (enemyKillVictory)
{
    if (enemySpawner.allEnemiesSpawned)
    {

        if (GameObject.FindGameObjectsWithTag("Enemy1").Length == 0 &&
            GameObject.FindGameObjectsWithTag("Enemy2").Length == 0)
            enemyKillVictoryAchieved = true;
    }
}

if (timedVictory)
{
    if (timerController.elapsedTime >= secondsToLast)
    {
        if (GameObject.FindGameObjectsWithTag("Enemy1").Length > 0)
        {
            GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy1");
            foreach (var en : GameObject in enemies)
            {
                Destroy(en.transform.parent.gameObject);
            }
        }

        if (GameObject.FindGameObjectsWithTag("Enemy2").Length > 0)
        {
            GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy2");
            foreach (var en : GameObject in enemies)
            {
                Destroy(en.transform.parent.gameObject);
            }
        }
        timedVictoryAchieved = true;
    }
}

if ((enemyKillVictoryAchieved || timedVictoryAchieved) && levelCompleteDone == false)
{
    levelCompleteDone = true;
    gameManager.AddCompleteLevel(gameManager.currentLevelIndex);
    levelCompleteUI.gameObject.SetActive(true);
}

```

Figura 3.6.0.6: Inițializare spawner inamici și jucător

Pentru a verifica dacă un nivel a fost completat, se verifică inițial dacă este victorie prin eliminarea tuturor inamicilor sau prin rezistarea un timp dat. Pentru victoria prin eliminarea inamicilor, se verifică dacă spawnerul de inamici a instantiat toți inamicii, iar dacă a făcut-o, se verifică să nu mai existe în scenă nici un obiect de tipul inamic. Dacă nu mai există, înseamnă că au fost eliberați de jucător, iar nivelul a fost completat. Pentru victoria prin rezistarea unui timp dat, se verifică dacă timpul măsurat de cronometru este mai mare decât timpul necesar pentru victorie. Dacă da, înseamnă că nivelul a fost completat, iar variabila pentru victorie este setată ca *true*. Pentru a nu avea erori, toți inamicii din scenă sunt distruiți.

Dacă au fost împlinite condițiile pentru victorie și variabila booleană *levelCompleteDone* este falsă, folosită pentru a nu repeta procedura de finalizare a nivelului, adaugăm nivelul curent la nivelurile completate și deblocăm următorul nivel apelând funcția *AddCompleteLevel* din Game Manager și afișăm ecranul de nivel completat.

```

if (enemyKillVictory && enemySpawner.allEnemiesSpawned)
{
    if (GameObject.FindGameObjectsWithTag("Enemy1").Length == 0 &&
        GameObject.FindGameObjectsWithTag("Enemy2").Length == 0)
        enemyKillVictoryAchieved = true;
}
if (timedVictory && timerController.elapsedTime >= secondsToLast)
{
    if (GameObject.FindGameObjectsWithTag("Enemy1").Length > 0)
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy1");
        foreach (var en : GameObject in enemies)
            Destroy(en.transform.parent.gameObject);
    }

    if (GameObject.FindGameObjectsWithTag("Enemy2").Length > 0)
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy2");
        foreach (var en : GameObject in enemies)
            Destroy(en.transform.parent.gameObject);
    }
    timedVictoryAchieved = true;
}
if ((enemyKillVictoryAchieved || timedVictoryAchieved) && levelCompleteDone == false)
{
    levelCompleteDone = true;
    gameManager.AddCompleteLevel(gameManager.currentLevelIndex);
    levelCompleteUI.gameObject.SetActive(true);
}

```

Figura 3.6.0.7: Verificare victorie nivel

În acest controller se găsesc și metodele de pauză a jocului și de adăugare a unui punct la scor, deoarece scorul este specific pe nivel. Pentru a pune pauza, jucătorul trebuie să apese tastă *Escape*, iar dacă nivelul nu este deja finalizat, prin victorie sau înfrângere, ecranul de pauza apare, iar timpul este oprit prin modificarea parametrului *timeScale* în 0. Atunci când se apasă din nou aceeași tastă, *timeScale* devine 1, și ecranul de pauză dispare, iar ecranul din joc apare. Metoda de adăugare a unui punct la scor este apelată, iar dacă noul scor este mai mare decât cel mai mare scor înregistrat de jucător în acel nivel, devine highscore-ul.

```

if (Input.GetKeyDown(KeyCode.Escape)
    && !gameOverUI.isActiveAndEnabled)
{
    if (gamePaused == false)
    {
        pauseMenu.gameObject.SetActive(true);
        pauseMenu.Pause();
        gamePaused = true;
    }
    else
    {
        pauseMenu.Unpause();
        pauseMenu.gameObject.SetActive(false);
        gamePaused = false;
    }
}
3 references
public void AddPoint()
{
    score += 1;
    scoreText.text = "Score: " + score.ToString();
    if (score > highscore)
        PlayerPrefs.SetInt("highscore", score);
}

```

Figura 3.6.0.8: Pauză și adăugare puncte în nivel

## Timer Controller

La începerea nivelului, funcția *BeginTimer* este apelată, care apelează *UpdateTimer*. Aceasta adaugă timpul trecut de la ultima apelare și modifică textul vizibil pentru jucător, formatând-ul. *Yield return null* face ca această coruină să revină în acest punct în cadrul următor, iar apoi să înceapă din nou funcția, verificând condiția din *while* [4]. Fără aceasta, funcția ar putea rula de mai multe ori pe cadru și nu numai că ar fi imprecisă, însă ar și bloca alte părți ale jocului din a rula, ceea ce ar produce erori. Datorita while-ului cu condiția ca variabila booleană *timerStarted* să fie adevărată, cronometrul nu va rula atunci când această variabilă este falsă, ceea ce este necesar pentru a nu cronometra atunci când jocul este pe pauză.

```

public void BeginTimer()
{
    timerStarted = true;
    elapsedTime = 0f;
    StartCoroutine(routine: UpdateTimer());
}
1 reference
private IEnumerator UpdateTimer()
{
    while (timerStarted)
    {
        if (!gamePaused)
        {
            elapsedTime += Time.deltaTime;
            timePlaying = TimeSpan.FromSeconds(elapsedTime);
            string timePlayingStr = "Time: " +
                timePlaying.ToString(format: "mm':ss'.fff");
            timeCounter.text = timePlayingStr;
        }
        yield return null;
    }
}

```

Figura 3.6.0.9: Timer controller

## 3.7 Metodele de joc - Story și Endless

Cum a fost descris și în capitolul introductiv, jocul poate fi jucat în două moduri: cel infinit, în care jucătorul încearcă să reziste cât mai mult timp și să acumuleze un scor cât mai mare și cel în care jucătorul parcurge povestea jocului, împărțită pe niveluri. Pentru a da un motiv de a rejuca povestea, în meniu de selectare a modului de joc, la butonul de statistici, se pot observa recordurile de scor și timp pentru tot modul de poveste adunat și pentru nivelul infinit, și, dacă jocul a fost terminat, pentru a putea verifica dacă aceste totaluri de timp și scor sunt pentru toate nivelurile.

## 3.8 Interfața grafică

### 3.8.1 Meniul principal

Pagina inițială a meniului principal conține titlul jocului și trei butoane: butonul *Quit*, pentru închiderea jocului, butonul *Options*, care deschide meniul de opțiuni și butonul *Play*, care deschide meniul de selectare al modului de joc și al caracterului. Setările utilizatorului rămân salvate, astfel că atunci când acesta redeschide jocul sau revine la meniul principal, jocul va fi setat la preferințele utilizatorului. Cum jocul pe niveluri începe cu nivelul 1 deblocat, iar la câștigarea unui nivel se deblochează următorul, progresul jucătorului va fi salvat, pentru a putea continua jocul de la ultimul nivel deblocat.



Figura 3.8.1.1: Meniul principal

### Meniul de opțiuni

Meniul de opțiuni conține, de asemenea, 3 butoane:

- *Graphics options*: activează meniul de opțiuni grafice;
- *Audio options*: activează meniul de opțiuni de sunet;
- *Back*: returnează utilizatorul la meniul principal.



Figura 3.8.1.2: Meniul de opțiuni

### Meniul de opțiuni grafice

În acest meniu se pot găsi setările grafice:

- *Fullscreen*: dacă jocul să fie în modul *Fullscreen* sau nu;
- *Graphics quality*: calitatea grafică a jocului, o calitate mai joasă îmbunătățește performanța jocului, dacă este nevoie;
- *Resolution*: setarea rezoluției jocului. Jocul preia toate opțiunile de rezoluție posibile pentru sistemul utilizatorului și aplică refresh rate-ul calculatorului ales de utilizator pe sistem, iar utilizatorul poate alege dintre aceste opțiuni;
- *Back*: returnează utilizatorul la meniul de opțiuni.



Figura 3.8.1.3: Setările grafice

### Meniul de opțiuni audio

În acest meniu se pot găsi setările audio:

- *Master volume*: modifică volumul tuturor sunetelor din joc;
- *Music volume*: modifică volumul muzicii de fundal din joc;
- *Back*: returnează utilizatorul la meniul de opțiuni.



Figura 3.8.1.4: Setările audio

## Meniul Play

Butonul *Play* duce utilizatorul la meniul de selectare a modului de joc, unde se poate alege modul de joc și se poate selecta caracterul:

- *Endless mode*: încarcă modul de joc endless, în care jucătorul se luptă la infinit cu dușmanii, pentru a face un scor cât mai mare până este înfrânt;
- *Levels mode*: meniul de selectare al nivelului de joc;
- *Choose character*: meniul de selectare al caracterului utilizat în joc;
- *Back*: returnează utilizatorul la meniul principal.



Figura 3.8.1.5: Meniul Play

### Endless mode

La apăsarea butonului *endless mode*, este încărcat nivelul endless, care se desfășoară pe aceeași hartă de joc ca cea de la nivelurile 1-5, cu inamici care apar la infinit, iar scopul jocului este ca jucătorul să reziste cât mai mult timp și să înfrângă cât mai mulți inamici.

## Levels Mode

În acest meniu, jucătorul selectează nivelul de joc pe care vrea să îl joace, iar nivelul selectat este încărcat. La începutul jocului, doar nivelul 1 este deblocat, iar următoarele niveluri sunt deblocate după câștigarea nivelului precedent. Jocul salvează ultimul nivel deblocat de jucător, iar la încărcarea meniului deblochează doar nivelurile disponibile pentru jucător. De asemenea, dacă jucătorul intră într-un nivel, se memorează și numărul nivelului, pentru a putea debloca nivelul următor la câștigarea acestuia și pentru a adăuga nivelul curent la lista celor câștigate. Astfel, jucătorul poate accesa următorul nivel din nivelul curent prin apăsarea unui buton, pentru a nu fi nevoie să se întoarcă după finalizarea fiecărui nivel la acest meniu de selectare a nivelului. Pentru a debloca butoanele de selectare a nivelurilor, acestea sunt adăugate într-o listă care conține toate butoanele de acest tip, iar la deblocarea unui nivel, butonul acelui nivel este activat în meniu. Butonul *Back* returnează utilizatorul la meniul play.



Figura 3.8.1.6: Meniul de selectare nivel

Pentru a accesa un nivel, butoanele sunt configurate sa acceseze funcția *LoadLevel* cu numărul nivelului corespondent butonului, iar pentru actualizarea butoanelor, activăm toate butoanele listate ca deblocate. La deschiderea jocului, sunt blocate toate nivelurile care nu au fost deblocate.

```
void Start()
{
    gameManager = GameObject.FindGameObjectWithTag("GameManager")
        .GetComponent<GameManager>(); // GameManager
    foreach (int i in gameManager.lockedLevels)
    {
        levels[i-1].gameObject.SetActive(false);
    }
}
Unity Message Log references
void Update()
{
    foreach (int i in gameManager.unlockedLevels)
        levels[i-1].gameObject.SetActive(true);
}
0 references
public void LoadLevel(int levelNr)
{
    gameManager.currentLevelIndex = levelNr;
    SceneManager.LoadScene("Level"+levelNr);
}
```

Figura 3.8.1.7: Codul pentru meniul de nivel

## Character selection

În acest meniu, utilizatorul poate selecta caracterul pe care dorește să îl controleze. Îi sunt prezentate aspectul caracterelor, numele și capabilitățile: dacă are mai multă sau mai puțină viață, viteza de mișcare sau damage-ul împotriva inamicilor. Aceasta poate selecta caracterul dorit, iar butonul de sub acel caracter nu va mai putea fi apăsat, iar pe buton va fi scris "SELECTED". Butonul *Back* returnează utilizatorul la meniul *play*. Alegerea caracterului va fi salvată, iar atunci când jocul este repornit caracterul selectat va rămâne acela ales de utilizator.



Figura 3.8.1.8: Meniul pentru selectare caracter

Pentru alegerea caracterelor, fiecare buton accesează o funcție specifică, funcțiile se-  
tând în *GameManager* variabilele specifice caracterului, precum nume, punctele de viață,  
viteza, imaginea caracterului vizibilă în nivel, numărul de gloanțe din arme. Atunci când  
un caracter este selectat, numele acestuia este salvat în *PlayerPrefs*, iar la deschiderea  
jocului, numele acestui caracter este folosit pentru a apela funcția care încarcă detaliile  
caracterului corespunzător în Game Manager.

```

        if (characterName == "Blue Man")
        {
            ChooseBlueMan();
            blueButtonText.text = "SELECTED";
        }
        else
            blueButtonText.text = "SELECT";
        if (characterName == "Brown Man")
        {
            ChooseBrownMan();
            brownButtonText.text = "SELECTED";
        }
        else
            brownButtonText.text = "SELECT";
    }
    1 reference
public void ChooseBlueMan()
{
    gameManager.name = "Blue Man";
    PlayerPrefs.SetString("characterName", gameManager.name);
    gameManager.health = 150f;
    gameManager.moveSpeed = 5f;
    gameManager.damageMultiplier = 1f;
    gameManager.pistolMagAmmo = 12;
    gameManager.pistolMaxAmmo = -1;
    gameManager.smgMagAmmo = 50;
    gameManager.smgMaxAmmo = 150;
    gameManager.shotgunMagAmmo = 7;
    gameManager.shotgunMaxAmmo = 42;
    gameManager.character = blueMan;
    bool[] weaponsEnabled = { true, true, true, true };
}

```

Figura 3.8.1.9: Codul pentru selectarea caracterelor

### Meniul "?"

La apăsarea butonului cu simbolul ?, un nou meniu va apărea, unde jucătorul poate afla câteva informații despre controale, modurile de joc și funcționalitatea de selectare a caracterului.

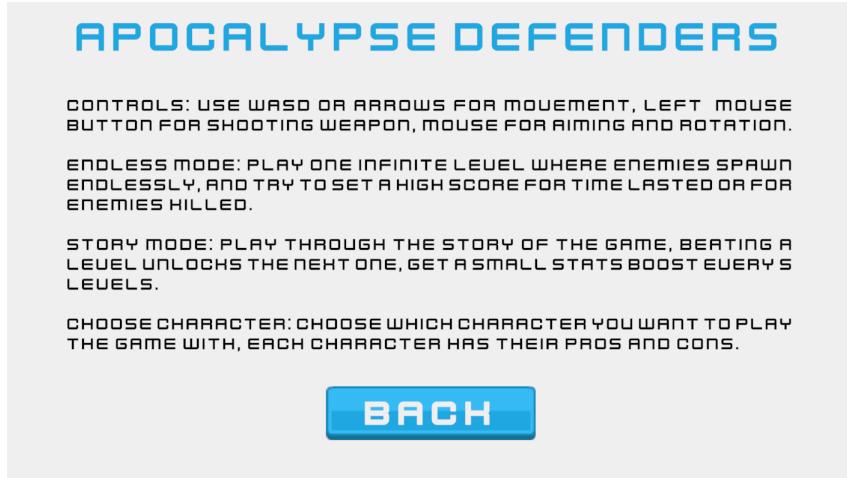


Figura 3.8.1.10: Meniul de tutorial

### Meniul Stats

La apăsarea butonului cu textul *Stats*, un panou va apărea deasupra meniului, în care se pot vedea mai multe informații:

- Dacă modul de poveste, pe niveluri, al jocului, a fost terminat, adică dacă jucătorul a câștigat toate nivelurile;

- Numărul de niveluri din modul de poveste câștigate de jucător;
- Scorul total acumulat în nivelurile câștigate de jucător în modul de poveste;
- Timpul total acumulat în nivelurile câștigate de jucător în modul de poveste;
- Scorul record în nivelul infinit;
- Timpul rezistat record în nivelul infinit;

Astfel, jucătorul poate afla informații despre propriile recorduri din joc și poate încerca să le doboare. Stabilirea unor recorduri și finalizarea unui joc în timp cât mai scurt, fenomen numit și *speedrunning*, devine din ce în ce mai popular în industrie. Acesta promovează competitivitatea, munca, gândirea rapidă și multe altele.



Figura 3.8.1.11: Meniul de opțiuni

### 3.8.2 Interfața grafică în nivel

Pentru o mai bună imersiune în universul jocului, interfața grafică din timpul jocului este foarte importantă. Jucătorul trebuie să aibă toate informațiile necesare la îndemână, să fie ușor de înțeles pentru a juca jocul în modul intenționat și pentru a se bucura de experiență. Astfel, informațiile trebuie să fie clare, însă fără să ocupe o suprafață mare din ecran, pentru a nu da senzația de îngheșuală. Utilizatorul poate vedea în colțul din stânga sus cele 4 arme, numărul de gloanțe din încărcător și de gloanțe totale, numărul de pe tastatură care trebuie apăsat pentru a selecta arma, pe care dintre acestea a selectat-o, prin schimbarea culorii din jurul armei selectate în portocaliu și scorul sub acestea. În partea de sus a ecranului se află cronometrul, care pornește atunci când începe nivelul și care este pe pauză odată cu jocul. Unele caractere nu au toate armele disponibile, astfel că acest lucru trebuie reflectat în interfața grafică. O armă indisponibilă este tăiată cu un X roșu, iar jucătorul nu o poate selecta sau folosi. Pentru fundalul pentru selectarea aramelor sunt folosite imagini ale butoanelor din pachetul pentru interfața grafică [8].



Figura 3.8.2.1: Interfața grafică din joc, cu pumnul indisponibil

### Codul pentru interfața grafică

Pentru a afișa arma activă, verificăm care dintre *GameObject*-ul celor 4 arme este activ și pornit, întrucât doar arma pe care jucătorul o folosește este activă și pornită. Atunci când o armă este activată, inclusiv la startul nivelului, când pistolul este activat, se activează poza cu arma selectată, care este suprapusă cu poza cu arma neselectată, care este dezactivată. Atunci când se schimbă arma, arma precedentă selectată este dezactivată, prin același procedeu. Apoi, pentru armele care au muniție, verificăm dacă variabila *infiniteAmmo* este adevărată sau falsă. Dacă este adevărată, afișăm simbolul infinit la muniția armei respective. Dacă este falsă, afișăm numărul de gloanțe din încărcător și numărul de gloanțe totale. De asemenea, se verifică dacăarma se reîncarcă, iar pe perioada reîncărcării, apare textul "Reloading..." în loc de muniția armei. Toate acestea se verifică în funcția *Update*, care se apelează în fiecare cadru.

```

if (Pistol.isActiveAndEnabled)
{
    pistolImages[0].gameObject.SetActive(false);
    pistolImages[1].gameObject.SetActive(true);
}
else
{
    pistolImages[0].gameObject.SetActive(true);
    pistolImages[1].gameObject.SetActive(false);
}

if (Pistol.infiniteAmmo)
{
    PistolAmmoText.text = "\u221E";
}
else
{
    if (Pistol.reloading)
    {
        PistolAmmoText.text = "Reloading...";
    }
    else
    {
        PistolAmmoText.text = Pistol.magAmmo.ToString() + "/"
                            + Pistol.totalAmmo.ToString();
    }
}

```

Figura 3.8.2.2: Codul pentru selectarea armelor

### 3.8.3 Meniul de pauză din nivel

Atunci când utilizatorul apasă tasta *Escape*, jocul este pus pe pauză, totul este oprit pe perioada pauzei, inclusiv cronometrarea timpului de joc. Pe lângă titlul "Game paused", jucătorul poate vedea mai multe informații despre caracterul său:

- viteza de mișcare;
- viața curentă și viața maximă;
- multiplicatorul de damage;
- timpul scurs până în momentul pauzei;
- scorul curent.

Utilizatorul mai are la dispoziție 3 butoane:

- *Restart*: restartează nivelul curent;
- *Options*: deschide meniul de opțiuni;
- *Main Menu*: returnează jucătorul la meniul principal.

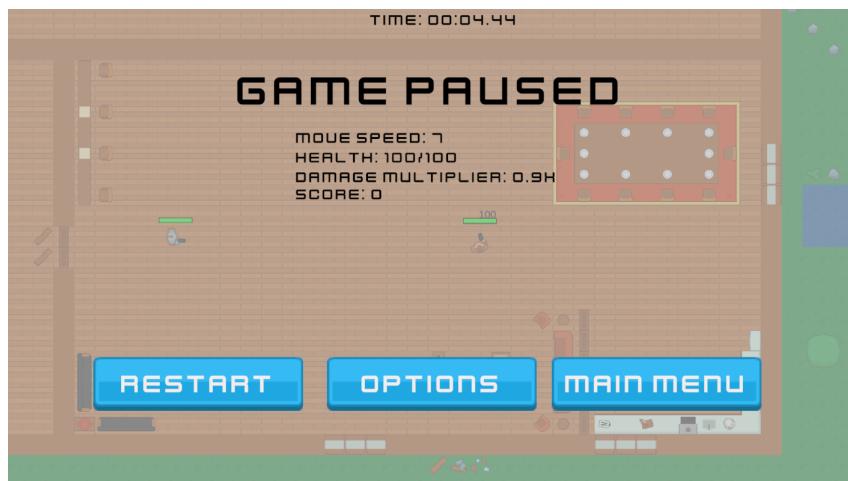


Figura 3.8.3.1: Meniul de pauză din nivel

Meniul de opțiuni deschis în meniul de pauză este același cu cel din meniul principal, descris la secțiunea 3.8.1, însă fundalul este la fel ca meniul de pauză, adică jocul în desfășurare.

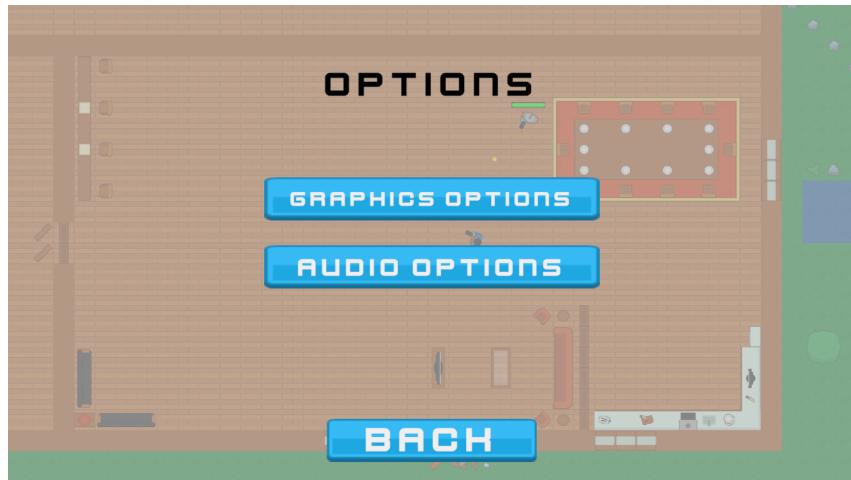


Figura 3.8.3.2: Meniul de opțiuni din nivel

#### Nivelul endless

În nivelul endless, meniul de pauză este puțin diferit față de cel din nivelurile din poveste: dacă jucătorul își bate recordul de timp sau de scor, un mesaj va apărea în meniul de pauză, iar butonul de reîntoarcere la meniul principal *Main Menu* este înlocuit de un buton cu textul *Give Up*, care returnează jucătorul la meniul principal și verifică salvarea scorului, dacă este cazul.

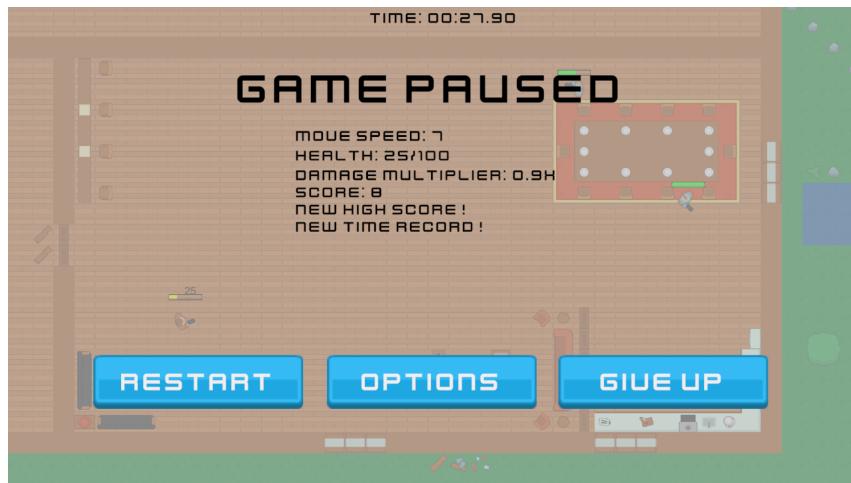


Figura 3.8.3.3: Meniul de pauză din nivelul endless

#### 3.8.4 Meniul de victorie și înfrângere din nivel

Atunci când jucătorul câștigă sau pierde nivelul, un panou este afișat, cu informațiile și butoanele corespunzătoare:

- Restart: restarează nivelul;
- Next Level: încarcă următorul nivel;

- Main Menu: returnează jucătorul la meniul principal.

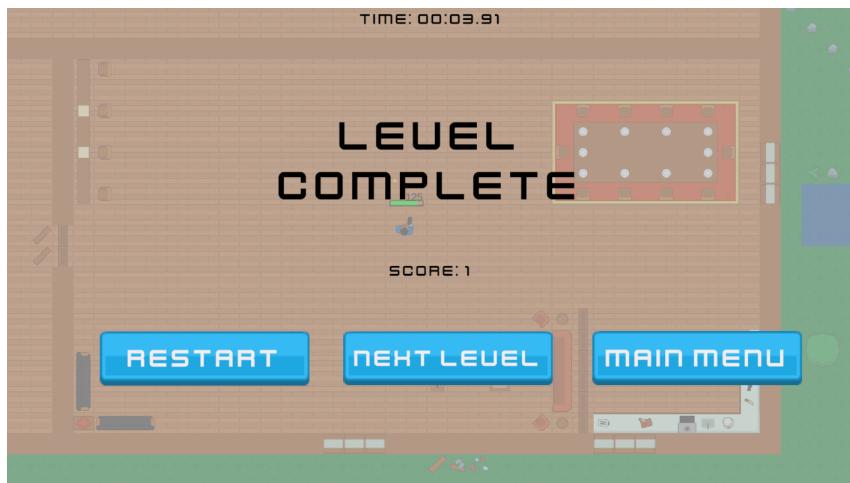


Figura 3.8.4.1: Panoul de victorie

Pentru a accesa următorul nivel, este apelată funcția NextLevel, care folosește numărul de nivel din Game Manager, adunat cu 1, pentru a accesa următorul nivel, al cărui nume corespunde cu numărul său în listă. Acest nivel este încărcat, numărul de nivel din Game Manager este incrementat cu 1, iar *timeScale* devine 1, pentru că începe un nivel, iar pauza nu mai este necesară. Acest meniu este necesar doar în modul de poveste.

```
public void NextLevel()
{
    int lv = gameManager.currentLevelIndex + 1;
    string levelName = "Level" + lv;
    SceneManager.LoadScene(levelName);
    Time.timeScale = 1f;
    gameManager.currentLevelIndex++;
}
```

Figura 3.8.4.2: Funcția NextLevel

Pentru panoul de înfrângere, sunt disponibile butoanele:

- Retry: restartează nivelul;
- Main Menu: returnează jucătorul la meniul principal.

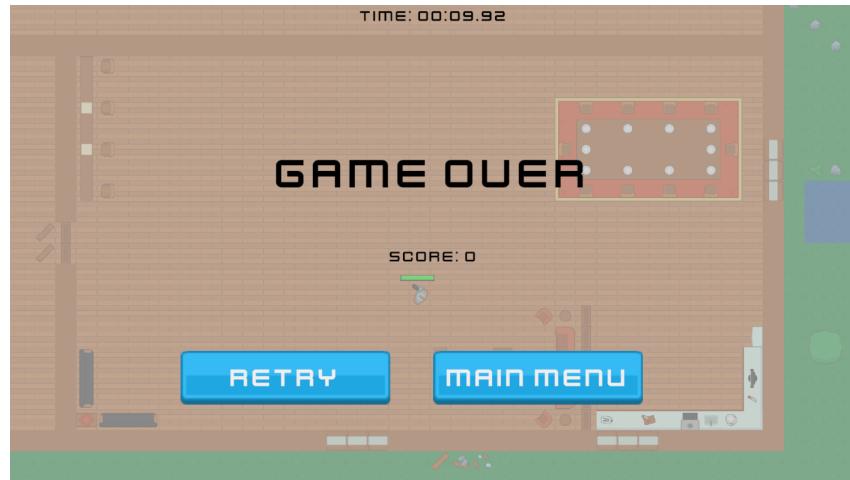


Figura 3.8.4.3: Panoul de înfrângere

### 3.8.5 Panourile cu text informativ și poveste

Înaintea începerii unui nivel din poveste, jucătorul este întâmpinat de un panou care conține butonul start, care face panoul să dispară și nivelul să înceapă și text care poate fi relevant pentru povestea jocului, explicarea unor mecanici de joc sau listarea obiectivului aceluiași nivel, precum eliminarea unui număr de inamici sau rezistarea pentru un anumit timp. De asemenea, sunt explicate și tipurile de power-up și cum funcționează acestea.



Figura 3.8.5.1: Panoul de start

## 3.9 Scalabilitatea jocului

Toate obiectele au fost gândite și realizate în aşa fel încât să poată fi personalizate în funcție de nevoie. De exemplu, la o armă poate fi modificată viteza de mișcare pe care jucătorul o are, numărul de gloanțe, damage-ul dat de gloanțele acesteia. Pentru a adăuga noi nivele, trebuie doar adăugate scenele necesare, butoanele în meniul de nivele, și numărul nivelelor trebuie adăugat în lista din Game Manager. Chiar și inamicii pot fi modificați cu ușurință, la fel ca jucătorul. Astfel, se pot adăuga cu ușurință niveluri noi, cu caracterul nostru, cu o hartă de alte dimensiuni, cu victorie atât prin timp cât și prin eliminarea inamicilor, cu un număr setat de inamici sau timp. Harta de joc poate fi de asemenea modificată cu ușurință, având la dispoziție multe imagini cu care pot fi create multe nivele. Acest lucru face ca jocul să poată fi extins cu ușurință, adăugând la conținutul și complexitatea acestuia.

## 3.10 Modificările făcute în editorul Unity

Ierarhia din cadrul scenei pentru meniu conține:

- Un obiect pentru cameră, cu componentele *Camera* și *Audio Listener*;
- Un obiect pentru lumini, cu componenta *Light* aplicată;
- Un obiect *Canvas*. Aceasta conține mai multe obiecte de același fel, reprezentând fiecare meniu și panou afișat în meniul principal, fiind descrise în capitolul 2.1;
- EventSystem: conține un *Standalone Input Module* și un *Event System*. Acestea se ocupă cu transmiterea evenimentelor: input de la tastatură, mouse, dar și cu alegerea cărui GameObject este considerat selectat sau cu alegerea cărui mod de input este folosit [12].
- AudioManager: un obiect la care este atașat scriptul *Audio Manager*, descris în secțiunea 3.6;
- GameManager: obiect la care este atașat scriptul *GameManager*, descris în secțiunea 3.6.

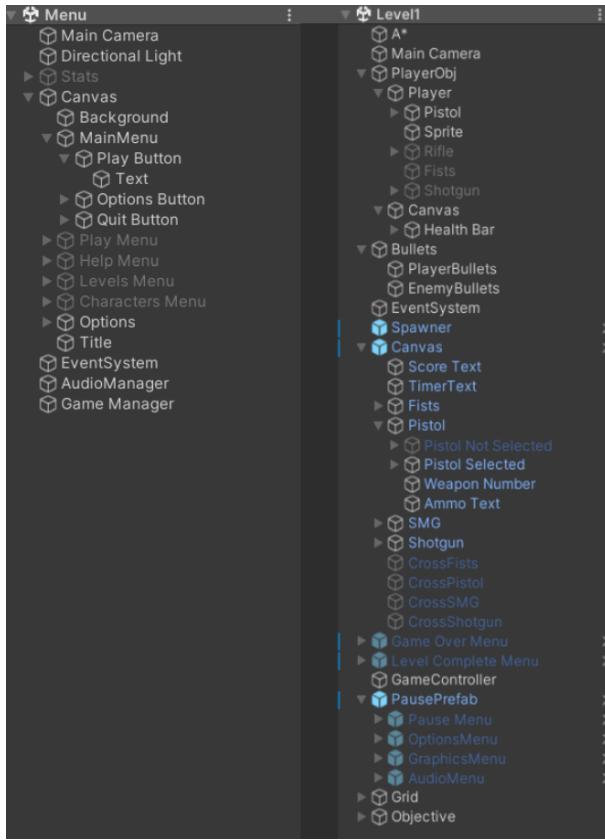


Figura 3.10.0.1: Ierarhia din meniul principal și dintr-un nivel

Ierarhia din cadrul scenei pentru un nivel conține:

- Un obiect pentru cameră, cu componentele *Camera* și *Audio Listener*;
- Un obiect *A\**, unde este atașat script-ul *A\**, descris în 2.4.1;
- Un obiect *PlayerObj* fără componente, care conține obiectul *Player*, cu armele și imaginea caracterului și obiectul *Canvas*, care conține un obiect *Health Bar*, ce reprezintă bara de viață a jucătorului;
- *EventSystem*: conține un *Standalone Input Module* și un *Event System*. Acestea se ocupă cu transmiterea evenimentelor: input de la tastatură, mouse, dar și cu alegerea cărui GameObject este considerat selectat sau cu alegerea cărui mod de input este folosit [12].
- *Bullets*: un obiect în care sunt instantiată drept copii gloanțele trase atât de jucător, cât și de inamici. Acest obiect a fost creat pentru a nu încărca ierarhia.
- *Spawner*: un obiect care conține doar script-urile pentru *spawnere*, cel de inamici și cel de powerup-uri.
- *GameController*: un obiect care conține doar script-urile *GameController* și *TimeController*, descrise în secțiunea 3.6;

- Grid: în acest obiect, care conține doar o componentă de tip *Grid*, sunt adăugate cele 3 *tilemap*-uri, care reprezintă "harta" de joc: două reprezintă fundalul și detaliile, precum podeaua și obiectele, iar unul reprezintă cele cu coliziuni, precum peretii, mesele, scaunele.
- Canvas: reprezintă interfața grafică din timpul nivelului, descrisă la secțiunea 3.8
- Game Over Menu, Level Complete Menu, Pause Menu, Options Menu, Graphics Menu, Audio Menu: meniurile din nivel, pentru pauza, setari sau finalizarea nivelului.
- AudioManager: un obiect la care este atașat scriptul *Audio Manager*, descris în secțiunea 3.6;
- GameManager: obiect la care este atașat scriptul *GameManager*, descris în secțiunea 3.6.

Obiectele de tip *Canvas*, adică meniurile și panourile, sunt formate din componente din partea stângă a pozei, enumerate în lista de mai sus. În partea dreaptă a pozei se află componente care formează un buton, toate butoanele din acest joc fiind la fel cu acesta. Acestea sunt folosite în toate obiectele *Canvas*. Butonul își schimbă imaginea în funcție de starea sa: în mod obișnuit, are imaginea din componenta *Image*; dacă utilizatorul ține mouse-ul deasupra butonului, își va schimba imaginea, iar dacă este apăsat, va avea altă imagine. Atunci când este apăsat, spre exemplu, butonul *Back*, acesta va activa un meniu nou și îl va dezactiva pe cel curent, folosind funcția *On Click*.

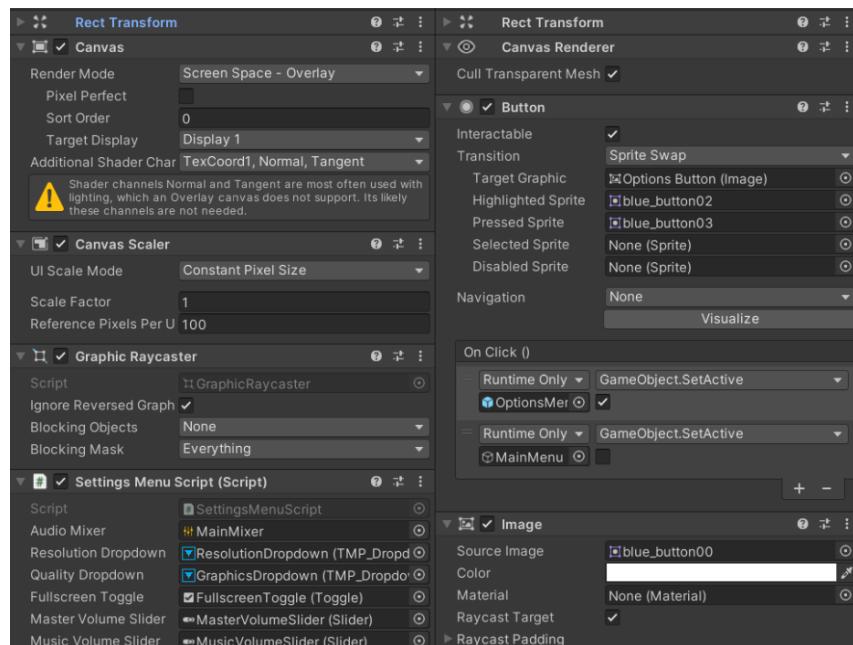


Figura 3.10.0.2: Configurația unui obiect de tip *Canvas* ca meniu și a unui buton în meniuri

În următoarea figură se pot observa obiectele *GameController*, *Spawner*, cu spawnerul pentru inamici și cel pentru powerup-uri și *Game Manager-ul*. Se pot observa atât variabile care conțin alte obiecte, folosite pentru referențierea acestor obiecte atribuite, dar și variabile folosite în execuția script-urilor, precum scor, coordonatele limită pentru spawnare sau detalii despre caracterul selectat de jucător sau lista de niveluri blocate și deblocate. În colțul din dreapta jos se poate observa și *Audio Managerul*. După cum se observă, aceste obiecte nu sunt vizibile în jocul propriu zis și nu au alte componente, în afară de scripturile atașate și *Audio Source*, utilizat pentru a transmite sunetele din *Audio Manager*.

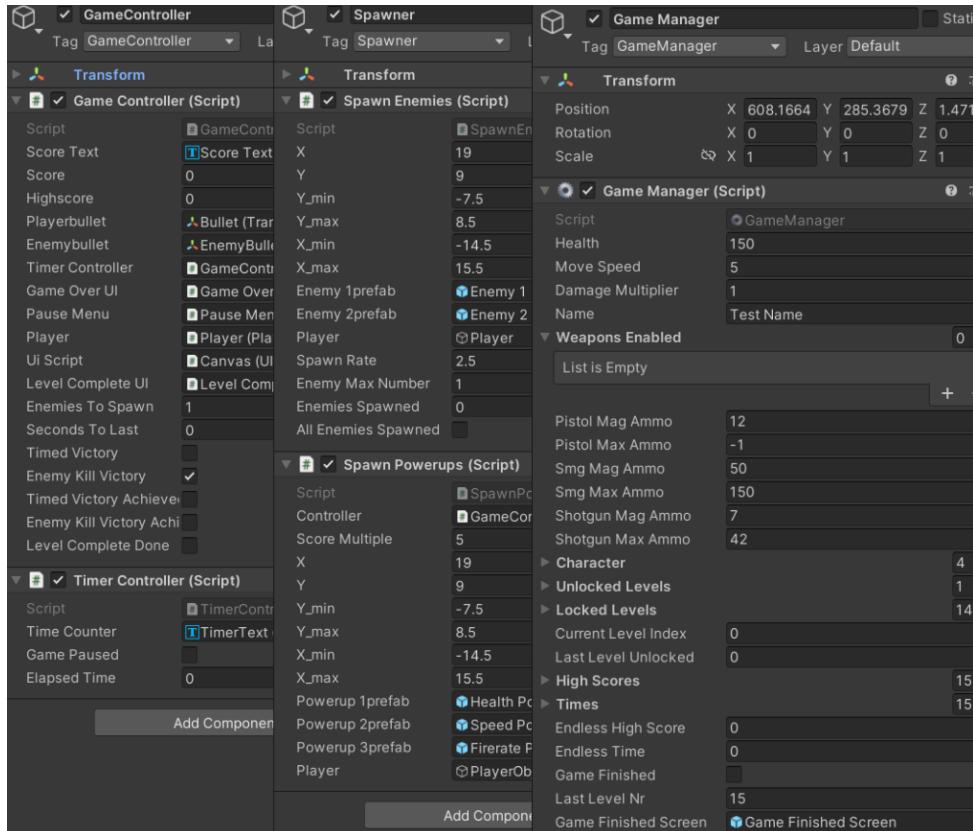


Figura 3.10.0.3: Configurația obiectelor Game Controller, Spawner, Game Manager și Audio Manager

Spre deosebire de obiectele din figura anterioară, următoarele obiecte, *Player* și două variante de inamic, vor fi vizibile în joc, vor avea corpuri și forțele li se vor aplica. Astfel, acestea conțin componente de tip *collider*, *rigidbody*, dar și *sprite renderer*, ultimul fiind atașat într-un obiect copil acestui obiect, pentru a evita anumite probleme întâmpinate cu orientarea caracterelor dacă toate componentele ar fi atașate pe același obiect. Inamicii mai au atașate și script-urile aferente *pathfinding-ului*. De asemenea, bara de viață a fiecărui caracter este un alt obiect, cu o componentă de tip *position constraint*, care previne rotația health bar-ului, iar împreună cu aceste obiecte din figură, sunt copiii unui obiect gol, pentru a se mișca împreună, însă fără a roti și bara de viață odată cu caracterul.

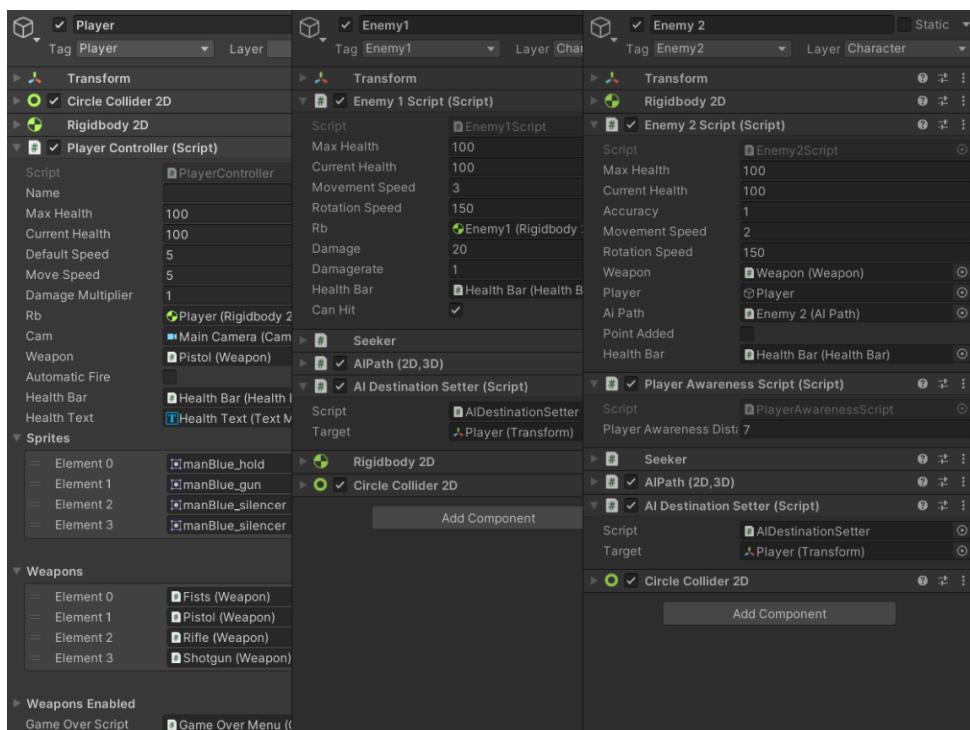


Figura 3.10.0.4: Configurație caractere pentru jucător și inamici

# Capitolul 4

## Concluzii

Pentru realizarea acestui proiect a fost necesară combinarea multor cunoștințe, dobândite pe parcursul academic. Cunoștințele și experiențele dobândite în cadrul studiilor mele în cadrul acestei facultăți mi-au permis: planificarea și cercetarea inițială, în care am gândit scopul și obiectivele jocului; am stabilit cerințele, platformele și modurile de dezvoltare; în partea de design și conceptualizare am creat conceptul inițial al jocului, precum povestea, mecanicile de bază și tematica, dar și structura jocului și a interfeței grafice; la implementarea tehnică, am folosit munca din pașii anteriori pentru a crea, programa și configura obiectele, nivelurile, interfața grafică, toate acestea integrând Unity, Microsoft Visual Studio și limbajul C#, și Adobe Photoshop, la finalizarea și documentarea jocului, pregătind jocul pentru utilizatori, testând toate funcționalitățile și reparând problemele și bug-urile detectate și documentând complet jocul, tehnic dar și conceptual, toate acestea au reprezentat o însumare a cunoștințelor și experiențelor dobândite în cadrul studiilor mele în cadrul acestei facultăți. Consider că jocul este într-o stare optimizată pentru lansare, însă există loc pentru și mai multe caracteristici și funcționalități, care ar face jocul și mai bun și ar onora și mai mult memoria și nostalgia jocurilor din copilărie, care m-au inspirat să creez acest joc.

### 4.1 Posibile dezvoltări ulterioare

Am creat jocul și obiectele care îl compun în aşa fel încât toate acestea să poată fi modificate extensiv, dar foarte simplu, pentru a putea adăuga sau modifica cu ușurință niveluri, caractere, arme, powerup-uri. Această scalabilitate a fost discutată mai în detaliu în secțiunea 3.9. Astfel, se pot adăuga mai multe niveluri, cu mai multe tipuri de inamici, de arme, sau de powerup-uri. De asemenea, alte dezvoltări ulterioare care s-ar potrivi și cu viziunea despre acest joc ar fi următoarele:

- Îmbunătățirea caracterului după câștigarea unui număr de niveluri: după ce jucătorul câștigă un anumit număr de niveluri, de exemplu 5, adică atunci când scapă

dintr-o cameră și harta se schimbă, acesta să poată alege un lucru la caracter pe care să îl poată îmbunătăți cu un anumit procent, precum viața, viteza, puterea armelor. Aceste îmbunătățiri ar fi valabile pentru toate caracterele, iar odată cu finalizarea jocului, utilizatorul ar putea să modifice alegerile sale pentru a încerca să își creeze caracterul perfect. Acest lucru ar și încuraja rejucarea jocului, întrucât acesta poate fi terminat mai rapid și eficient odată ce caracterul este mai puternic, ca urmare a acestor îmbunătățiri dobândite.

- Sistem de tipul *boss fight* la finalizarea unei hărți/camere, adică a 5 niveluri: după ce termină obiectivul ultimului nivel din acea etapa de 5 niveluri, pe hartă ar apărea un inamic mult mai puternic decât cei din timpul nivelurilor obișnuite, iar jucătorul trebuie să îl eliminate pentru a putea progesa și a câștiga ultimul nivel din acel set de 5 niveluri, care reprezintă o cameră.
- Mai multe arme care pot fi deblocate, din care jucătorul să își poată alege pentru a crea un set de 4: la câștigarea unor niveluri, jucătorul deblochează alte arme, cu diferite caracteristici. Dintr-un meniu de configurare, jucătorul ar putea alege dintre toate aceste arme câteva pe care să le folosească în joc, în funcție de strategie, de inamicii din nivel și de stilul de joc și preferințele utilizatorului.
- Abilitate specială pentru fiecare caracter: la atingerea unui anumit scor, jucătorul poate apăsa o tastă pentru a activa abilitatea specială a caracterului, care este foarte potentă, însă nu durează decât o perioadă scurtă de timp. Aceste abilități ar fi diferite pentru fiecare caracter.
- Mai multe caractere care pot fi deblocate pe parcursul poveștii: la câștigarea unor anumite niveluri, jucătorul deblochează alte caractere, pe care le poate alege din meniul special creat. Aceste caractere ar fi diferite ca aspect, dar și ca lucruri precum viața, viteza, armele disponibile, puterea armelor, abilitate specială.
- Clasament online al recordurilor pentru nivelul endless: pentru a fi mai competitiv, jucătorul poate compara recordul personal în nivelul infinit cu cele mai bune din lume sau cu prietenii, dacă jocul ar fi integrat pe o platformă care permite acest lucru, precum *Steam* sau *Epic Games*.

# Bibliografie

- [1] Brackeys, *Introduction to AUDIO in Unity*, URL: <https://www.youtube.com/watch?v=60T43pvUyfY>, (accessed: 15.04.2023).
- [2] Brackeys, *TOP DOWN SHOOTING in Unity*, URL: <https://www.youtube.com/watch?v=LNLV0jbrQj4>, (accessed: 10.04.2023).
- [3] Xiao Cui și Hao Shi, „A\*-based pathfinding in modern computer games”, în *International Journal of Computer Science and Network Security* 11.1 (2011), pp. 125–130.
- [4] Turbo Makes Games, *How to Make an In-Game Timer in Unity - Beginner Tutorial*, URL: <https://www.youtube.com/watch?v=qc7J0iei3BU>, (accessed: 02.06.2023).
- [5] Aron Granberg, *A\* Pathfinding Project*, URL: <https://arongranberg.com/astar/>, (accessed: 10.03.2023).
- [6] Kenney, *Space Shooter Redux*, URL: <https://www.kenney.nl/assets/space-shooter-redux>, (accessed: 30.05.2023).
- [7] Kenney, *Top-down Shooter Assets*, URL: <https://www.kenney.nl/assets/top-down-shooter>, (accessed: 01.04.2023).
- [8] Kenney, *UI Pack*, URL: <https://www.kenney.nl/assets/ui-pack>, (accessed: 01.04.2023).
- [9] OpenClipart-Vectors, *Fist, Power, Hand*, URL: <https://pixabay.com/vectors/fist-power-hand-anger-violence-149497/>, (accessed: 14.06.2023).
- [10] pixelmannen, *10 Pixel Weapons*, URL: <https://opengameart.org/content/10-pixel-weapons-0>, (accessed: 29.05.2023).
- [11] Unity, *Canvas*, URL: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html>, (accessed: 10.06.2023).
- [12] Unity, *Event System*, URL: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html>, (accessed: 02.06.2023).
- [13] Unity, *Important Classes - MonoBehaviour*, URL: <https://docs.unity3d.com/Manual/class-MonoBehaviour.html>, (accessed: 01.04.2023).

- [14] Unity, *Mathf.Atan2*, URL: [https://docs.unity3d.com/ScriptReference/Mathf.  
.Atan2.html](https://docs.unity3d.com/ScriptReference/Mathf.Atan2.html), (accessed: 01.04.2023).
- [15] Unity, *PlayerPrefs*, URL: [https://docs.unity3d.com/ScriptReference/Player  
Prefs.html](https://docs.unity3d.com/ScriptReference/Player<br/>Prefs.html), (accessed: 10.06.2023).
- [16] Unity, *Quaternion.Euler*, URL: [https://docs.unity3d.com/ScriptReference/Qu  
aternion.Euler.html](https://docs.unity3d.com/ScriptReference/Qu<br/>aternion.Euler.html), (accessed: 30.05.2023).
- [17] Unity, *Time.timeScale*, URL: [https://docs.unity3d.com/ScriptReference/Tim  
e-timeScale.html](https://docs.unity3d.com/ScriptReference/Tim<br/>e-timeScale.html), (accessed: 02.06.2023).