



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICA ŞI
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

OUTLAW OUTPOST

Absolvent
Stan Ana-Maria

Coordonator științific
Conf. dr. Mureșan Claudia

Bucureşti, iunie-iulie 2023

Rezumat

Industria jocurilor video este una dintre cele mai populare și mai profitabile industrie de divertisment din lume, aceasta continuând să crească și să se dezvolte de la an la an. Odată cu dezvoltarea tehnologiei din spatele jocurilor video s-au descoperit din ce în ce mai multe beneficii pe care acestea le oferă. De la reducerea stresului, facilitarea interacțiunilor sociale și ridicarea moralului până la dezvoltarea empatiei și a aptitudinilor cognitive, acestea devenind o formă de artă și creativitate creând un impact cultural semnificativ asupra vieții adolescentilor și a tehnologiei.

Outlaw Outpost este un joc top-down shooter 2D plasat în lumea captivantă a Vestului Sălbatic. Jucătorul calcă pe urmele unui cowboy dornic de aventură care este determinat să salveze orașul de Jack McCallister, un cinic șef de bandă de nelegiuți prezentat cu ajutorul unei grafici vizuale atrăgătoare și a mai multor mecanici de joc inovatoare. Lucrarea are ca scop prezentarea detaliată a mai multor tehnici și aplicații folosite pentru dezvoltarea unui joc video complet în Unity.

Procesele de dezvoltare ale acestuia au fost împărțite în mai multe etape: am stabilit obiectivele jocului și am dezvoltat concepțele specifice și harta conceptuală, am făcut designul jocului definind mecanicile de joc, povestea și nivelurile, am creat elemente vizuale folosind Photoshop, am trecut la etapa de programare și am implementat mecanicile de joc în motorul ales, am adăugat elemente vizuale și auditive, iar în final am supus jocul unui proces de testare pentru a elimina erorile.

Pentru a realiza acest proiect m-am folosit de mai multe informații dobândite pe parcursul celor trei ani de facultate: planificarea și dezvoltarea unei aplicații, folosirea unui motor de joc în limbaj C# și utilizarea programării orientate pe obiecte. În plus, am învățat să dezvolt un joc complet și complex folosindu-mă de informații noi dobândite, care reprezintă înțelegerea mai în amănunt a codului, obiectelor folosite în joc și multitudinea de funcționalități pe care Unity le oferă.

În final, am îmbinat pașii de dezvoltare ai unui joc cu personaje, povești, idei, imagini vizuale și mecanici originale alături de informațiile și aptitudinile dobândite pe parcursul anilor de studii și a propriilor experiențe pentru a dezvolta un proiect care are la bază crearea unei experiențe plăcute și relaxante pentru jucători care stârnește interes, empatie și dorință de aventură.

Abstract

The video game industry is one of the most popular and profitable entertainment industries in the world, and it continues to grow and develop year after year. With the development of the technology behind video games, more and more benefits have been discovered. From reducing stress, facilitating social interactions and boosting morale to developing empathy and cognitive skills, they are becoming a form of art and creativity creating a significant cultural impact on the lives of teenagers and technology itself.

Outlaw Outpost is a top-down 2D shooter set in the exciting world of the Wild West. Follow in the footsteps of an adventure-seeking cowboy who is determined to save his town from Jack McCallister, a cynical outlaw gang boss presented with eye-catching visual graphics and several innovative gameplay mechanics. The paper aims to detail several techniques and applications used to develop a complete video game in Unity.

The development process of the game was divided into several stages: we established the game's goals and developed the basic concepts and concept map, I designed the game defining the game mechanics, story and levels, I created visual elements using Photoshop, I moved to the programming stage and implemented the game mechanics in the chosen engine, I added visual and auditory elements, and finally I put the game through a testing process to eliminate errors.

To complete this project, I used several insights gained during my three years of university: planning and developing an application, using a C# language game engine, and using object-oriented programming. In addition, I learned how to develop a complete and complex game using newly acquired information, which is the deeper understanding of the code, the objects used in the game and the multitude of features that Unity provides.

In the end, I combined the steps of developing a game with original characters, stories, ideas, visuals and mechanics along with the information and skills I have acquired through years of study and my own experiences to develop a project that is based on creating an enjoyable and relaxing experience for players that sparks interest, empathy and a desire for adventure.

Cuprins

1 Introducere	6
1.1 Motivația și scopul alegerii temei	6
1.2 Repere istorice ale jocurilor	7
1.3 Repere istorice ale motoarelor de jocuri	8
1.4 Prezentarea generală a temei	10
1.5 Contribuția proprie în realizarea lucrării	11
2 Tehnologii folosite	14
2.1 C#	14
2.2 Unity	14
2.2.1 Despre Unity	14
2.2.2 Istoria Unity	15
2.3 A* Path-Finding	16
2.4 Photoshop	16
3 Fundamente teoretice și proiectarea aplicației	18
3.1 Explicarea termenilor	18
3.2 Tipuri de joc	19
3.2.1 Story mode	19
3.2.2 Endless mode	20
3.3 Personajul Principal	21
3.3.1 Mișcarea	22
3.3.2 Împușcarea	23
3.3.3 Diferite tipuri de gloanțe	25
3.3.4 Viață și arme	25
3.3.5 Moarte	26
3.3.6 Animații jucător principal	28
3.4 Inamici	29
3.4.1 Deplasarea folosind A* Pathfinding	30
3.4.2 Enemy Controller	30
3.4.3 Viață inamic	32

3.4.4	Spawner	32
3.4.5	Boss Fight	33
3.5	Puteri speciale	35
3.5.1	Viață	35
3.5.2	Gloanțe	36
3.5.3	Viteză	36
3.5.4	Cheie	37
3.6	Interfață	38
3.6.1	Meniu principal	38
3.6.2	Meniu setări	40
3.6.3	Meniu Pauză	42
3.6.4	Panouri cu povestea jocului	43
3.7	Camera	44
3.8	Magazin	45
4	Concluzii	48
	Bibliografie	50

Capitolul 1

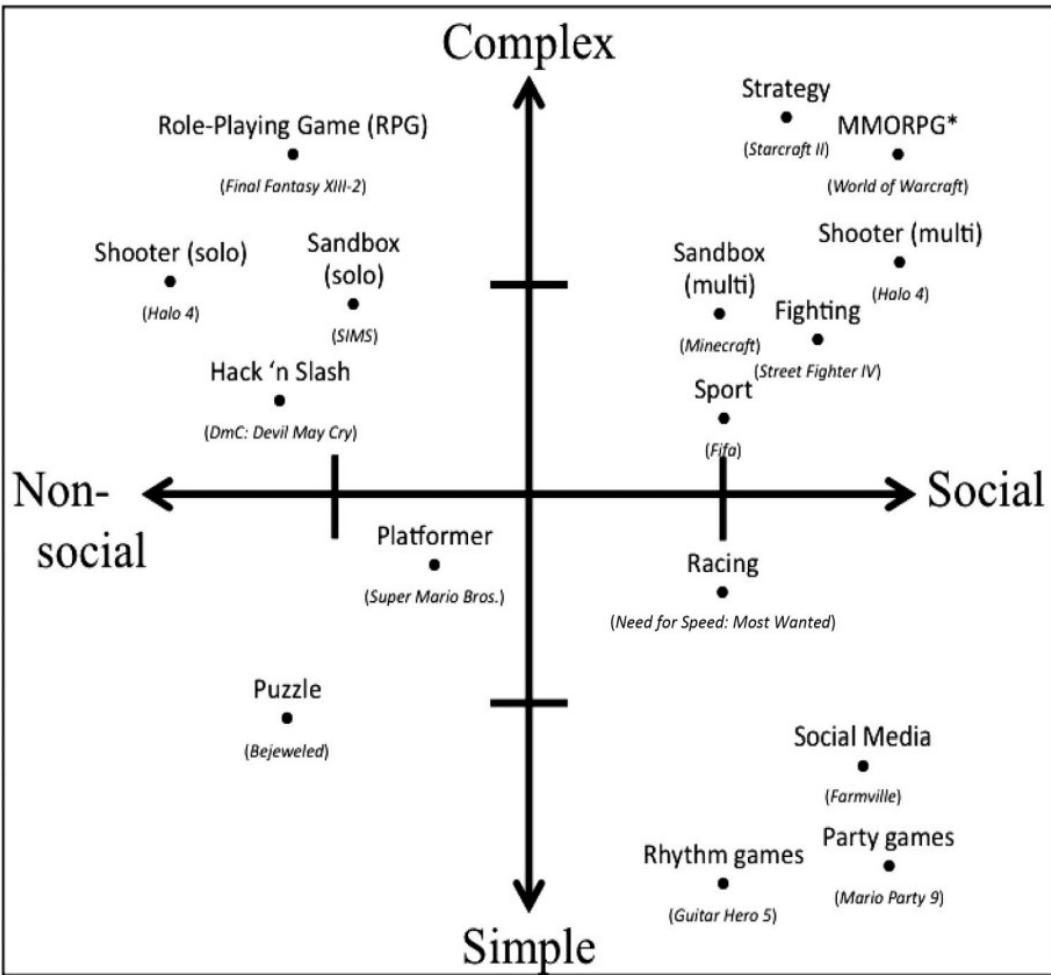
Introducere

1.1 Motivația și scopul alegerii temei

Întotdeauna am fost pasionată de jocuri video și tehnologie, având norocul să copilăresc în momente cheie ale dezvoltării acestora, iar la maturitate am observat că m-au ajutat și în dezvoltarea mea proprie. M-au ajutat să învăț lucruri noi, să clădesc relații mai puternice cu cei din jur, să-mi dezvolt creativitatea, empatia, gândirea logică și să-mi mențin atitudinea pozitivă asupra vieții.

Am ales acest tip de lucrare, deoarece până recent acestea au fost prezentate într-o lumină proastă când, de fapt jocurile video au la bază foarte multe beneficii pentru dezvoltarea aptitudinilor cognitive și este cunoscut ca unul dintre cele mai eficiente și eficace mijloace prin care copiii și tinerii generează sentimente pozitive [8]. Învățând să facă față eșecurilor continue în jocuri, copiii construiesc reziliență emoțională pe care se pot baza în viața de zi cu zi iar aceasta îi determină să fie mai calculați înainte de a întreprinde orice acțiune. Jocurile video au o influență pozitivă asupra proceselor mentale de bază cum ar fi luarea deciziilor, percepția, memoria și atenția [11]. Doresc să prezint mai multe metode tehnologice și artistice pentru crearea jocurilor video, promovând astfel importanța jocurilor video în viața și dezvoltarea armonioasă a tinerilor.

Prin această lucrare o să prezint tehnici de bază și complexe ale dezvoltării jocurilor video pentru a încuraja și ajuta pasionații să creeze de la interfață și hartă până la inteligența artificială a inamicilor.



Figură 1.1 - Harta conceptuală a principalelor genuri de jocuri video [8]

Astfel, putem să observăm harta conceptuală a principalelor genuri de jocuri video și diferențele beneficiei pe care ni le oferă. Jocurile sociale și simple oferă o atmosferă relaxantă, în care nu trebuie să ne concentrăm prea mult la joc ci să beneficiăm mai mult de socializarea cu cei cu care interacționăm. Cele sociale și complexe ne oferă posibilitatea de a lucra cu mai mulți jucători în echipă, încurajând dezvoltarea comunicării între aceștia, iar partea complexă împinge jucătorul în a fi foarte atent și concentrat, pentru a-și îndeplini misiunile împreună cu ceilalți jucători.

Cele non sociale au ca scop dezvoltarea aptitudinilor cognitive, încurajând atenția la detalii și concentrarea jucătorului, deoarece nu mai face parte din o echipă și fiecare mișcare pe care alege să o facă poate să influențeze jocul. Jocurile complexe non-sociale au potențialul de a dezvolta abilitățile de rezolvare a problemelor în timp ce, cele simple, oferă mai mult liniște și relaxare.

1.2 Repere istorice ale jocurilor

Istoria jocurilor video datează din 1948, când a apărut primul dispozitiv electronic de divertisment. În următorul deceniu, dezvoltarea jocurilor video însărcină dezvoltarea

calculatoarelor, iar primul succes comercial al jocurilor video datează din 1972, când jocul arcade Pong, o simulare de tenis de masă, a apărut și a fost urmat de dezvoltarea unor jocuri arcade precum pinball. Jocurile arcade se jucau în localuri și săli speciale publice până când Magnavox Odyssey, prima consolă de jocuri pentru acasă, a fost lansată în acel an, aceasta oferă până la 28 de jocuri video diferite. Aproape toate jocurile de pe acea consolă erau de tipul sport sau acțiune. În 1977, ca urmare a dezvoltării microprocesorului, a apărut consola de jocuri Atari VCS (Atari 2600) iar importanța unei astfel de console a fost neprețuită. Pentru prima dată, jocurile video, nu aveau ca scop doar de a adăuga bile sau de a distrage adversarii, ci necesitau deja o anumită gândire strategică. Numărul de utilizatori de calculatoare și console a crescut în fiecare an, iar între timp, numărul de jucători de jocuri video de acasă a depășit numărul de jocuri video de tip arcade din public. În ultimul deceniu al secolului al XX-lea, a fost realizat cel mai mare pas înainte în ceea ce privește calitatea graficii și complexitatea a jocurilor. Urmând dezvoltarea treptată a jocurilor din deceniul anterior, jocurile care introduc o a treia dimensiune în lumea jocurilor video, făcând o impresie mai puternică de realitate și permitând o conectivitate emoțională mai puternică cu personajele [11].

De la început jocurile video au fost făcute ca formă de divertisment dar o dată cu trecerea timpului acestea s-au dezvoltat nu doar ca tehnologie dar și ca povestea și mesajul pe care îl transmit, multe jocuri recente devenind populare datorită șirului narativ care prezintă povești foarte asemănătoare cu viața reală.

1.3 Repere istorice ale motoarelor de jocuri

Din punct de vedere istoric, motoarele de jocuri au fost uneori strâns legate de jocurile în sine. În 1987, Ron Gilbert, împreună cu ajutorul lui, Chip Morningstar, au creat SCUMM, sau Script Creation Utility for Maniac Mansion în timp ce lucrau la Lucasfilm Games. SCUMM este un mare exemplu de motor de joc care a fost creat special pentru un anumit tip de joc. Cuvântul "MM" din SCUMM vine de la Maniac Mansion, care a fost un joc de aventură aclamat de critică și primul care a folosit sistemul de tip "point-and-click" [7].



Figură 1.3.1 - Imagine din jocul Maniac Mansion [6]

Motorul de joc SCUMM a fost responsabil pentru convertirea scripturilor constând din cuvinte de tip token lizibile de către om, cum ar fi "walk character to door" în programe de mărimea unui octet care să fie citite de interpretul motorului de joc. Interpretul era responsabil pentru controlul actorilor din jocuri pe ecran și de a prezenta sunetul și grafica. Capacitatea de a scrie scripturi de joc în loc să îl codifice, a facilitat prototiparea rapidă și a permis echipei să înceapă să construiască și să se concentreze asupra gameplay-ului încă din prima etapă [7].

La mijlocul anului 1991, a avut loc o schimbare seismică majoră în industrie, la o companie numită id Software, atunci când John Carmack a construit un sistem 3D pentru un joc numit Wolfenstein 3D. Până atunci, grafica 3D erau în general limitate la jocurile de simulare a zborurilor care se mișcau lent sau la jocurile cu poligoane simple, deoarece hardware-ul de calculator disponibil era prea lent pentru a calcula și afișa numărul de suprafețe necesare pentru un joc rapid de joc de acțiune 3D rapid. Carmack a reușit să oculească hardware-ul curent cu limitările actuale prin utilizarea unei tehnici grafice numite raycasting. Aceasta a permis o afișare rapidă a mediilor 3D prin calcularea și afișarea doar a suprafeței vizibile pentru jucător, în loc de întreaga zonă din jurul jucătorului [7].

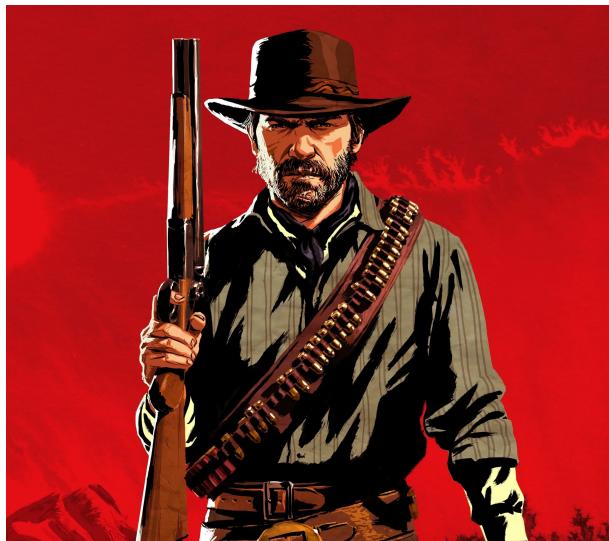


Figură 1.3.2 - Posterul jocului Wolfenstein 3D [14]

Studiourile moderne de dezvoltare de jocuri AAA, cum ar fi Bethesda Game Studios și Blizzard Entertainment au adesea propriile lor studiouri interne și motoare de joc proprii. Motorul de joc intern al Bethesda se numește: Creation Engine și a fost folosit pentru a crea The Elder Scrolls V: Skyrim, precum și Fallout 4. Blizzard are propriul motor de joc folosit pentru a crea jocuri precum World of Warcraft și Overwatch. După ce acel joc initial este lansat, motorul de joc își găsește adesea o nouă viață atunci când este reutilizat pentru următorul joc care ieșe din studioul de jocuri respectiv. Este posibil ca motorul să necesite actualizări pentru a rămâne la zi și pentru a profita de cea mai recentă tehnologie, dar nu trebuie reconstruit de la zero. În cazul în care o companie de dezvoltare de jocuri nu are un motor intern, utilizează de obicei un motor open-source sau licențiază un motor de la o terță parte cum ar fi Unity. Pentru a crea un joc 3D semnificativ în zilele noastre fără a utiliza un motor de joc ar fi o sarcină incredibil de solicitantă - atât din punct de vedere finanțier, precum și din punct de vedere tehnologic. De fapt, studiourile de jocuri cu motoare de joc proprii necesită echipe separate de programare dedicate în întregime construirii caracteristicilor motorului și optimizarea acestuia [7].

1.4 Prezentarea generală a temei

Un joc pe care o să îl prezint este Red Dead Redemption II produs de Rockstar în 2018 care are o tematică asemănătoare cu lucrarea mea, Vestul Sălbatic și responsabilitatea jucătorului în raport cu lumea care îl înconjoară. Când jucătorii preiau pentru prima dată rolul lui Arthur Morgan, un cowboy din vestul sălbatic al Americii, știu puține lucruri despre el și sunt încă în procesul de înțelegere și învățare a mecanicii jocului. Este nevoie de timp pentru a trece de această fază până când acțiunile lor devin instinctuale. Jucătorii încep să recunoască trăsături în Arthur și încep să se conecteze la el, poate recunoscând trăsături similare în ei însăși. Jucătorii abordează un joc video ca pe orice situație sau interacțiune, aducându-și experiențele din trecut, perspectiva proprie a lumii și vietii. Persoanele care joacă jocuri sunt adesea conduși în mod inerent să adere la propriul simț al moralității, mai ales atunci când trebuie să ia decizii rapide și intuitive în orice moment. Datorită legăturii pe care jucătorii o au cu Arthur, există acțiuni pe care jucătorii le pot comite și care i-ar putea face să se simtă vinovați pentru asta. Ca haiduc, Arthur tinde să fie o persoană destul de violentă, care face acte discutabile din punct de vedere moral și adesea violente de-a lungul jocului. Jucătorul descoperă în curând ce este corect și ce este gresit din punct de vedere moral în lumea jocului, care coincide aproximativ cu moralitatea lumii reale. Sentimentul de empatie față de victimă tinde să oblige jucătorul să se opreasca înainte de a comite acțiuni imorale care ar duce la o mai mare dezonoare pentru el însăși și pentru Arthur [11].



Figură 1.4.1 - Arthur Morgan [5]

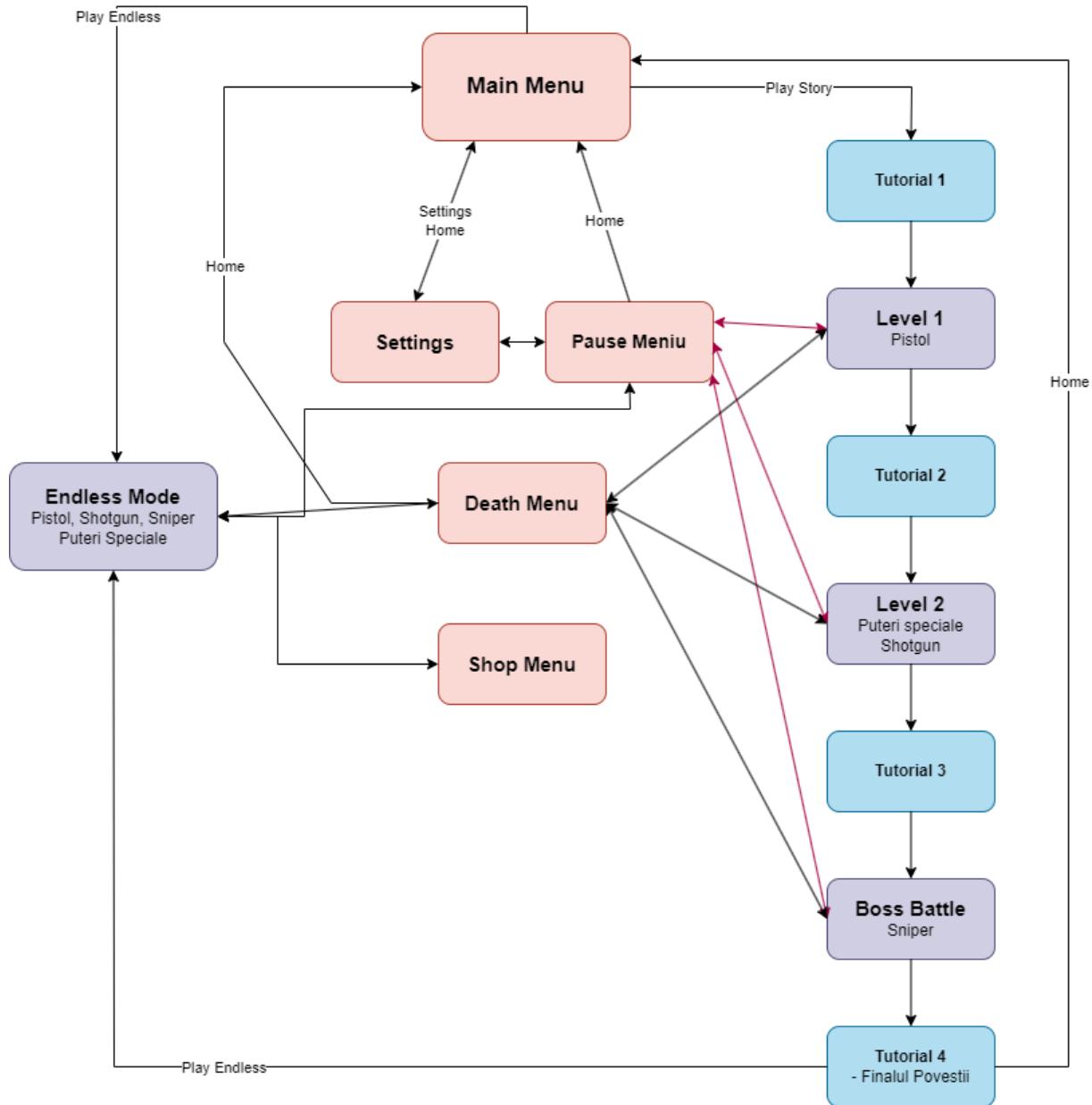
Personajul principal al lucrării mele este tot un cowboy care împinge jucătorul să facă alegerea morală corectă. Jocul începe prin întâlnirea cu un alt cowboy mai bătrân care îl înștiințează pe personajul principal că o bandă de bandiți condusă de Jack McCallister terorizează un oraș din apropiere pe nume Outlaw Outpost. Jucătorul abordează povestea jocului ca o situație reală în care trebuie să acționeze în mod onorabil datorită empatiei și setului de valori morale pe care le detine. Astfel, utilizatorul urmărește firul narativ al jocului și pornește în misiune spre Outlaw Outpost să-l salveze de banda de bandiți.

1.5 Contribuția proprie în realizarea lucrării

În lucrarea mea am implementat mai multe tehnici de bază ale programărilor jocurilor video dar am și creat multe funcționalități originale care îl diferențiază de restul jocurilor de acest tip de pe piață. În primul rând, am scris o poveste și dialoguri pentru joc specifice perioadei, pentru a îi da o tentă de realism și probleme specifice perioadei vestului sălbatic. Am implementat trei tipuri de arme diferite care se regăsesc la jucătorul principal, dar și la inamici, fiecare având alte caracteristici și un număr diferit de gloanțe disponibile. Jocul are două tipuri diferite de joc care pot fi alese:

Story Mode: poveste liniară cu trei nivele și un inamic final special care are gloanțe de tip HoomingMissles, care urmăresc jucătorul. Acest mod este mai mult folosit pentru a prezenta jucătorului mecanicile de bază ale jocului.

Endless Mode: un singur nivel care ține până când personajul principal este învins și scorul este reținut într-o tabelă de scor locală. În plus, jucătorul strângе bani în joc pe care îi poate cheltui într-un magazin special. Acesta oferă opțiunea de a cumpăra două tipuri de arme noi, pe lângă pistolul de start și mai multe puteri care au ca scop să ajute jucătorul să obțină un scor cât mai mare.



Figură 1.5.1 - Diagrama jocului [imagine autor]

Diagrama din figura 1.5.1 prezintă harta conceptuală a jocului planificată de mine. Când jocul este deschis utilizatorul este întâmpinat prima dată de meniul principal al jocului prezentat mai sus de fereastra **Main Menu**. Din acesta jucătorul poate să joace povestea jocului sau Endless mode, să schimbe setările jocului sau să iasă din joc.

Dacă acesta apasă butonul de **Story** atunci va urma diagramele din dreapta. Intră în scena de **Tutorial 1** care îți oferă primele informații despre poveste și te trimită în scena următoare care reprezintă nivelul 1 al jocului și acest proces se continuă până ajungem în **tutorialul 4**. La sfârșitul poveștii jucătorul poate alege două variante, fie apasă butonul de **Home** și revine la meniul principal ori apasă butonul **Next** și intră în **Endless Mode**. În fiecare nivel al poveștii personajul deblochează arme și puteri care sunt scrise sub titlul nivelului, iar din fiecare nivel se poate intra în meniul pauză apăsând tasta **Escape**. Din

acesta se poate ajunge înapoi la meniul principal sau să intre în meniul de setări.

Dacă este apăsat butonul de **Endless** atunci urmăram diagrama din stânga care are ca scop obținerea unui scor cât mai mare. Din acest mod jucătorul poate să intre în 3 meniuri diferite: **meniul de pauză** care este prezent și în Story, meniul de moarte care este la fel prezent în meniul de Story și meniul de magazin care îi oferă jucătorului posibilitatea să cumpere abilități și arme.

Capitolul 2

Tehnologii folosite

2.1 C#

C# este un limbaj de programare modern, orientat pe obiecte care permite dezvoltatorilor să creeze multe tipuri de aplicații sigure și robuste care rulează în .NET. C# își are rădăcinile în familia de limbaje C și va fi imediat familiar pentru programatorii C, C++, Java și JavaScript. C# oferă construcții de limbaj pentru a susține direct aceste concepte, făcând din C# un limbaj natural în care se creează și se utilizează componente software [9].

De la începuturile sale, C# a adăugat caracteristici pentru a sprijini noile sarcini de lucru și practicile emergente de proiectare software [9]. Limbajul de programare C# este recunoscut pe scară largă ca unul dintre alegerile de top pentru dezvoltarea jocurilor video. Una dintre principalele motive pentru popularitatea lui în acest domeniu este compatibilitatea sa strânsă cu motorul de joc Unity, fiind flexibil și ușor de înțeles.

2.2 Unity

2.2.1 Despre Unity

Unity este un motor de jocuri cross-platform dezvoltat de Unity Technologies, anunțat și lansat pentru prima dată în iunie 2005. De atunci, motorul a fost extins treptat pentru a suporta o varietate de platforme desktop, mobile, console și de realitate virtuală. Este deosebit de popular pentru dezvoltarea de jocuri pentru dispozitive mobile iOS și Android, este considerat ușor de utilizat pentru dezvoltatorii începători și este popular pentru dezvoltarea de jocuri indie [18].

Motorul poate fi folosit pentru a crea jocuri tridimensionale (3D) și bidimensionale (2D), precum și simulări interactive și alte experiențe. Motorul a fost adoptat de industrii din afara jocurilor video, cum ar fi cea cinematografică, auto, arhitectură, inginerie, construcții și forțele armate ale Statelor Unite ale Americii [18].

Începând cu 2018, Unity a devenit o alegere populară în crearea jocurilor mobile, fiind utilizat în aproximativ jumătate din jocurile mobile disponibile pe piață. De asemenea, aproximativ 60% din conținutul de realitate augmentată și realitate virtuală a fost creat utilizând platforma Unity. Tehnologia Unity servește ca bază pentru majoritatea experiențelor de realitate virtuală și augmentată, iar publicația Fortune a afirmat că Unity ”domină afacerea de realitate virtuală” [18].

2.2.2 Istoria Unity

De la debutul său în 2005, Unity a fost folosit pentru a dezvolta mii de aplicații de jocuri și aplicații pentru desktop, dispozitive mobile și console. O mică mostră a unor titluri bine cunoscute dezvoltate de-a lungul anilor cu Unity ar fi inclus: Thomas Was Alone (2010), Temple Run (2011), The Room (2012), RimWorld (2013), Hearthstone (2014), Kerbal Space Program (2015), Pokémon GO (2016) și Cuphead (2017) [7].



Figură 2.2.2.1 - Jocul Hearthstone [17]

Pentru dezvoltatorii de jocuri care doresc să își personalizeze fluxul de lucru, Unity oferă posibilitatea de a extinde editorul vizual implicit. Acest lucru este un extrem de puternic mecanism care permite crearea de instrumente, editori și instrumente personalizate. Imaginea-vă crearea unui instrument vizual pentru ca proiectanții dvs. de jocuri să poată cu ușurință modifica valorile obiectelor din joc, cum ar fi punctele de lovire pentru o clasă de personaje, abilitățile, distanța de atac sau obiectele care cad, fără a fi nevoie să intre în cod și să modifice valorile sau să utilizeze o bază de date externă. Toate acestea sunt posibile și simplu de utilizat folosind funcționalitatea Editor Extension pe care o oferă Unity. Un alt avantaj al Unity este Unity Asset Store. Asset Store este o vitrină online în care artiștii, dezvoltatorii și creatorii de conținut pot încărca conținut pentru a fi cumpărat și vândut. Asset Store conține mii de extensii de editor gratuite și contra cost, modele, scripturi, texturi, shaders și multe altele, pe care echipele le pot utiliza pentru a-și accelera termenele de dezvoltare și îmbunătăți un produs final [7].

2.3 A* Path-Finding

A* Path-finding este un algoritm care găsește drumul optim între două puncte. În această lucrare algoritmul găsește traseul optim între inamici și jucătorul principal folosind inteligența artificială. Acesta a fost dezvoltat și optimizat de multe ori de-a lungul timpului, devenind astfel cel mai popular algoritm de inteligență artificială folosit în crearea jocurilor video.

Pentru identificarea traseului, algoritmul A* examinează în mod repetat cel mai promițător loc neexplorat pe care l-a văzut. Atunci când o locație este explorată, algoritmul se termină dacă acea locație este obiectivul; în caz contrar, acesta notează toți vecinii acelei locații pentru explorarea ulterioară. A* este, probabil, cel mai popular algoritm de căutare a căilor de acces în jocurile AI [4].

```
1. Add the starting node to the open list.  
2. Repeat the following steps:  
    a. Look for the node which has the lowest  
        f on the open list. Refer to this node  
        as the current node.  
    b. Switch it to the closed list.  
    c. For each reachable node from the current  
        node  
        i. If it is on the closed list, ignore  
            it.  
        ii. If it isn't on the open list, add it  
            to the open list. Make the current  
            node the parent of this node. Record  
            the f, g, and h value of this node.  
        iii. If it is on the open list already,  
            check to see if this is a better  
            path. If so, change its parent to the  
            current node, and recalculate the f  
            and g value.  
    d. Stop when  
        i. Add the target node to the closed  
            list.  
        ii. Fail to find the target node, and the  
            open list is empty.  
3. Tracing backwards from the target node to the  
    starting node. That is your path.
```

Figură 2.3.1 - Pseudocodul A* [10]

Astfel, pentru proiectul dezvoltat o să folosesc un pachet care implementează automat acest algoritm în Unity. Aceasta se numește A* Pathfinding Projectcite [4] și prezintă algoritmi optimizați alături de un set mare de caracteristici diferite fiind ușor de folosit și implementat.

2.4 Photoshop

”Adobe Photoshop, aşa cum este cunoscut astăzi, este vârful de lance al gamei de produse software pentru editare de imagini digitale, fotografii, grafică pentru tipar, video

și Web de pe piață. Photoshop este un program cu o interfață intuitivă și care permite o multitudine extraordinară de modificări necesare în mod curent profesioniștilor și nu numai: editări de luminozitate și contrast, culoare, focalizare, aplicare de efecte pe imagine sau pe zone (selectii), retușare de imagini degradate, număr arbitrar de canale de culoare, suport de canale de culoare pe 8, 16 sau 32 biți, efecte third-party etc. Există situații specifice pentru un profesionist în domeniu când alte pachete duc la rezultate mai rapide, însă pentru prelucrări generale de imagine, întrucât furnizează instrumente solide, la standard industrial, Photoshop este efectiv indispensabil” [16].

În lucrarea mea am folosit Photoshop de mai multe ori pentru a crea mai multe elemente de joc. Acesta a fost extrem de important, deoarece într-un joc calitatea și imaginea este la fel de importantă ca mecanicile de joc. Am creat jucători și inamici în Photoshop, viață, meniuri și am schimbat culoarea mai multor elemente pentru a fi sigură că are caracteristici estetice plăcute ochiului.

Capitolul 3

Fundamente teoretice și proiectarea aplicației

3.1 Explicarea termenilor

În această secțiune o să definesc o serie termeni specifici Unity regăsiți de mai multe ori în proiect și cod.

Funcția Update() se auto-apelează la fiecare cadru [15].

Funcția FixedUpdate() este utilizată pentru actualizările variabilelor de tip RigidBody2D și a calculelor fizice [15].

Time.deltaTime Pentru a obține timpul scurs de la ultimul apel al funcției Update() [15].

Rigidbody 2D plasează un obiect sub controlul motorului de fizică. În 2D, obiectele se pot deplasa numai în planul XY și se pot roti numai pe o axă perpendiculară pe acest plan [15].

Box Collider 2D este un Collider care interacționează cu sistemul de fizică 2D și este folosit pentru a activa evenimente când se produce coliziunea cu alte obiecte [15].

PlayerPrefs este o clasă care stochează preferințele jucătorului între sesiunile de joc. Poate stoca valori de tip string, float și int în registrul platformei utilizatorului. Unity stochează PlayerPrefs într-un registru local, fără criptare. Se folosește la High Score [15].

GameObject reprezintă tipul de obiecte care pot exista într-o scenă [15].

Quaternion este o clasă care reprezintă o rotație absolută sau relativă și oferă metode de creare și manipulare a acestora [15].

Transform ne oferă o varietate de moduri de a lucra cu poziția, rotația și scala unui GameObject prin intermediul unui script, precum și cu relația sa ierarhică cu GameObjects părinte și copil [15].

OnCollisionEnter2D este trimisă atunci când un colizor de intrare intră în contact cu colizionarul acestui obiect (numai în fizica 2D) [15].

OnCollisionStay2D trimite fiecare cadru în care un colider de pe un alt obiect atinge coliderul acestui obiect (numai în fizica 2D) [15].

3.2 Tipuri de joc

În Outlaw Outpost jucătorul are de ales între două variante posibile. Acestea au la bază aceleasi idei și principii dar fiecare se diferențiază prin mai multe caracteristici. Pentru a dezvolta jocul am creat mai multe componente originale în Photoshop, iar pentru harta de bază am folosit mai multe seturi [12, 13, 1] care sunt gratis.

3.2.1 Story mode

Prima opțiune de joc se numește Story Mode și prezintă un tutorial sub formă de poveste jucătorului pentru a se obișnui cu mecanicile de joc. Astfel, firul narativ al poveștii este aceasta: personajul principal se întâlnește cu un cowboy bătrân care nu mai este destul de puternic. Acesta îi povestește că o bandă de bandiți condusă de Jack McCallister terorizează pe nedrept un oraș, Outlaw Outpost. Acesta acceptă cu onoare misiunea pentru a-i salva pe oamenii nevinovați și pornește în aventură. Înainte de fiecare nivel ne întâlnim din nou cu bătrânelul cowboy care ne oferă mai multe informații despre poveste și despre mecanicile jocului. Pe măsură ce înaintăm în nivele cu atât deblocam mai multe arme, puteri și rămășițe din povestea completă a jocului.

Povestea este alcătuită din mai multe nivele de forma omoară un număr dat de bandiți și avansează, iar finalul este un nivel special destinat unui Boss Battle cu șeful bandei de bandiți. După ce îl înfrângi pe acesta povestea este finalizată și ai două opțiuni: să te întorci la meniul principal sau să continui în modul Endless pentru a folosi ce ai învățat.



Figură 3.2.1.1 - Nivelele poveștii [imagine autor]

Figura 3.2.1.1 prezintă cele trei niveluri din modul de poveste din joc. Acestea sunt structurate și planificate de mine pentru a oferi jucătorului o experiență captivantă și plină de aventură, care au ca scop prezentarea mecanicilor de bază ale jocului și obișnuirea cu acestea pentru a pregăti jucătorul pentru nivelul final în care se întâlnește cu șeful de bandă și modul Endless.

3.2.2 Endless mode

Prima opțiune de joc se numește Endless mode și are ca scop acumularea cât mai mare de puncte până când personajul principal moare. Aici sunt prezentate mai multe tipuri de inamici, regăsiți și în Story Mode, care tot apar pe hartă până când jocul este încheiat. Aceștia apar cu ajutorul unor spawnere speciale care au mai multe coordonate în care pot să activeze inamici. În joc mai sunt prezente și mai multe puteri ajutătoare diferite pentru jucătorul principal, care vor fi descrise în subcapitolul lor separat și care pot apărea pe harta în locuri diferite când condiția de activare este respectată. Personajul principal are la dispoziție trei arme: un pistol, care este arma de bază a jucătorului, și două arme pe care jucătorul le poate debloca folosind bănuți pe care îi strâng când omoară inamici sau ridică un bănuț de pe hartă.

În acest tip de joc harta este limitată, dar mai mare decât cele prezente în Story Mode, și are aproape de centru un personaj cu care jucătorul principal trebuie să interacționeze

pentru a cumpăra și îmbunătății puterile. În magazin sunt două tipuri de obiecte: cele care îți oferă îmbunătățiri la putere, viață, gloanțe și cele care îți deblochează două arme noi, o pușcă și o pușcă cu lunetă.



Figură 3.2.2.1 - Nivelul Endless [imagine autor]

Figura 3.2.2.1 arată harta jocului Endless. Aceasta reprezintă un oraș din Vestul Sălbatic care este înconjurat de un gard pentru a limita locul în care se petrece acțiunea. În gardurile din dreapta sus și jos și stânga jos o să apară puterile speciale, iar în mijlocul hărții, în fața salonului este un NPC cu care trebuie să interacționeze pentru a accesa magazinul.

3.3 Personajul Principal

Personajul principal este un cowboy care pleacă pe urmele unei bande de răufăcători și a liderului acestieia. El este reprezentat în joc de un sprite 2D ca în Figura 3.3.1. Personajul este original fiind desenat și creat de mine.



Figură 3.3.1 - Personaj principal [imagine autor]

Acestui sprite 2D îi este atribuit un Rigidbody2D, pentru a avea caracteristicile unui corp fizic. Acesta mai are funcția Box Collider 2D, pentru a înregistra coliziunea cu puterile speciale și gloanțele inamicilor și un animator pentru a activa animațiile personajului. Cowboyul mai are atașate asupra sa 2 scripturi, cel de mișcare și cel de împușcare, iar în final este salvat ca un obiect de tip prefab pentru a fi mai ușor folosit când se adaugă scene noi.

3.3.1 Mișcarea

Mișcarea se face folosind tastele WASD la o viteză definită în cod. Viteza se poate schimba dacă jucătorul ridică o putere în timpul jocului. Figura 3.3.1.1 prezintă codul mișcării personajului principal 2D de tipul top-down (văzut de sus). Variabila movement este de tipul Vector2 care reține coordonatele curente ale personajului din spațiul 2D preluând poziția acestuia pe axa orizontală și verticală folosind Input.GetAxisRaw(), care este în continuu schimbare, deoarece este regăsită în funcția Update(). În variabila mPoz este salvată poziția cursorului în funcție de camera jocului.

```

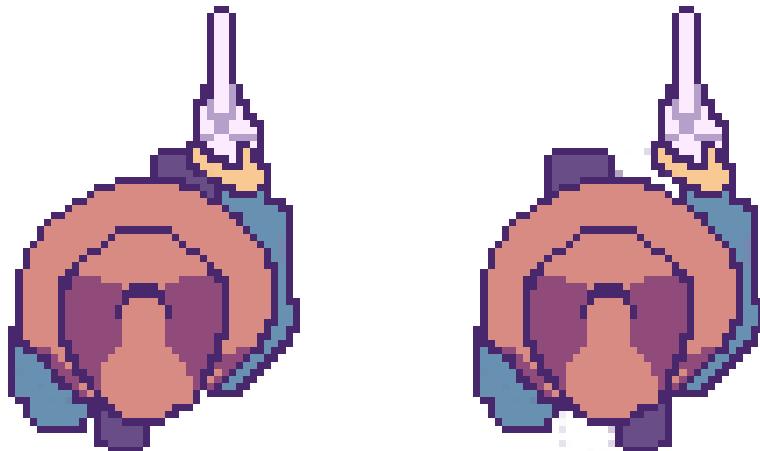
void Update()
{
    movement.x = Input.GetAxisRaw("Horizontal");
    movement.y = Input.GetAxisRaw("Vertical");
    mPoz = cam.ScreenToWorldPoint(Input.mousePosition);
}

Unity Message | 0 references
private void FixedUpdate()
{
    rb.MovePosition(rb.position + movement * moveSpeed * Time.fixedDeltaTime);
    Vector2 lookDir = mPoz - rb.position;
    float ang = Mathf.Atan2(lookDir.y, lookDir.x) * Mathf.Rad2Deg - 90f;
    rb.rotation = ang;
}

```

Figură 3.3.1.1 - Codul mișcării [imagine autor]

În Funcția FixedUpdate() regăsim mișcarea și rotirea personajului principal folosind o tehnică de bază [2]. Astfel, rb este o variabilă de tip Rigidbody2D care reprezintă corpul personajului și utilizând funcția MovePosition() se ocupă de mișcarea obiectului dat folosind poziția curentă a obiectului, vectorul moment, scalarul moveSpeed care reprezinta viteza cu care se mișcă corpul și scalarul Time.fixedDeltaTime care face ca mișcarea personajului să fie independentă față de cadrele jocului. Vectorul lookDir salvează direcția în care se uită obiectul iar variabila ang calculează un unghi în grade care este ajustat cu ajutorul -90f pentru a orienta obiectul corect în plan, acesta reprezentând rotația personajului principal.



Figură 3.3.1.2 - Animație mișcare [imagine autor]

3.3.2 Împușcarea

Împușcarea se face pe tasta click stânga iar personajul principal are 3 stări diferite care reprezintă trei arme distincte, fiecare având alte caracteristici. Jucătorul poate să urmărească în ce stare se află urmărind imaginea din dreapta barei de viață.

Pistol: o armă balansată și cea principală a personajului. Aceasta provoacă daune normale la o distanță medie. Tasta 1

Shotgun: această armă provoacă mai multe daune, trăgând trei gloanțe deodată dar care funcționează doar la o distanță mai apropiată. Tasta 2

Sniper: este arma cea mai puternică din arsenalul jucătorului. Daunele sunt majore și funcționează la distanțe mari. Dezavantajul este că aceasta este accesată mai târziu în joc și are un număr mai mic de gloanțe. Tasta 3

Pentru ca personajul principal să poată împușca o să am nevoie de două coduri, unul pentru fiecare glonț și unul pentru acțiunea de împușcare. Astfel, pentru fiecare stare a jucătorului în cod există o funcție diferită care instanțiază gloanțele. În prima parte a codului se verifică dacă a fost apăsată tasta click dreapta, se verifică dacă mai sunt gloanțe în armă și în ce stare este arma. Dacă condițiile pentru arma respectivă trec se cheamă funcția stării respective.

```
1 reference
void Shoot()
{
    GameObject bullet = Instantiate(bulletPr, fPoint.position, fPoint.rotation);
    Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
    rb.AddForce(fPoint.up * bulletForce, ForceMode2D.Impulse);
    noOfBulletsInRoundPistol -= 1;
}
1 reference
void Shoot1()
{
    GameObject bullet = Instantiate(bulletPr, fPoint.position, fPoint.rotation);
    Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
    rb.AddForce(fPoint.up * bulletForce, ForceMode2D.Impulse);
    GameObject bulletright = Instantiate(bulletRight, fPoint.position, fPoint.rotation);
    Rigidbody2D rbr = bulletright.GetComponent<Rigidbody2D>();
    rbr.AddForce(fPoint.up * bulletForce, ForceMode2D.Impulse);

    GameObject bulletleft = Instantiate(bulletLeft, fPoint.position, fPoint.rotation);
    Rigidbody2D rbl = bulletleft.GetComponent<Rigidbody2D>();
    rbl.AddForce(fPoint.up * bulletForce, ForceMode2D.Impulse);
}
```

Figură 3.3.2.1 - Codul împușcăturii [imagine autor]

În figura 3.3.2.1 sunt prezentate două funcții: funcția Shoot()[3] care este folosită pentru pistol și este asemănătoare cu cea de la pușca cu lunetă, singura diferență fiind obiectul cu care este instantiat glonțul și funcția Shoot1() care reprezintăarma pușcă.

În funcția Shoot() se instanțiază o variabilă de tip GameObject cu un prefab care reprezintă glonțul pistolului, în poziția și rotația respectivă pistolului. După instanțierea glonțului i se atribuie un Rigidbody2D și în final i se atribuie o forță care pornește din vârful pistolului cu caracteristicile setate din cod. Pentru pușca cu lunetă funcția este identică, schimbând doar prefabul cu care este inițializat glonțul. Funcția Shoot1() are

la bază aceeași idee ca funcția Shoot(), doar se mai adaugă trei instantieri de gloanțe diferite, fiecare prefab fiind reconfigurat puțin în stânga, respectiv dreapta.

Codul de împușcat mai are o funcție definită: cea de reîncărcare care o să fie de tipul async pentru a crea o amânare de câteva secunde când este apăsată tasta R. În funcție de câte gloanțe sunt în încărcător se vor calcula numărul de gloanțe rămase în încărcător și numărul de gloanțe rămase în total, în funcție de starea în care se află jucătorul.

3.3.3 Diferite tipuri de gloanțe

Pentru a crea arme diferite trebuie să creăm gloanțe cu proprietăți diferite. Toate acestea primesc un RigidBody2D și un collision2D pentru a înregistra ce obiecte au fost atinse.

Pentru gloanțele de pistol și pușcă folosim același prefab și tip de glonț, singura schimbare fiind că instantiem 3 gloanțe când tragem cu o pușcă și desigur câtă viață da glonțul inamicului când aceștia intră în coliziune. Acest lucru îl realizăm folosind funcția On-Collision2D, apoi verificăm eticheta cu care aceasta a interacționat. Dacă eticheta este "Enemy", "Enemy2", "Enemy3" atunci chemăm componenta specifică inamicului și apelăm funcția de TakeDamage().

Pentru glonțul special de pușcă cu lunetă schimbăm imaginea glonțului de pistol și atașăm un cod diferit. Îl salvăm ca prefab și atașăm la el caracteristicile care sunt și la glonțul de pistol și copiem codul de la pistol, singurul lucru diferit fiind numărul de gloanțe și momentul în care se produce coliziunea cu un alt inamic acestuia o să îi scadă mai mult din viață decât un glonț de pistol.

3.3.4 Viață și arme

Viața și arma actuală a personajului principal sunt afișate în partea stânga sus a ecranului ca în figura de mai jos cu numărul 3.3.4.1. Astfel, viața este salvată pe Canvas ca un element de tip slider care reprezintă bara de viață. În plus, personajul principal are și 3 arme disponibile pe care le poate debloca, fiecare având caracteristici diferite, care apar în dreapta barei de viață. În figura de mai jos este prezentată viața când personajul principal are activ pistolul, dacă acesta schimbă arma se va înlocui pistolul cu noua armă.



```
public void SetMaxH(int health)
{
    slider.MaxValue = health;
    slider.value = health;
}
3 references
public void setHealth(int healt)
{
    slider.value = healt;
}
```

Figură 3.3.4.1 - Codul și aspectul vizual al barei de viață [imagine autor]

Codul din figura 3.3.4.1 ajută la schimbarea vieții de-a lungul jocului. Prima dată salvăm o variabilă de tip Slider care reprezintă sliderul din Canvas al vieții. Pe lângă slider mai sunt două variabile de tip int care reprezintă viață maximă pe care o poate avea personajul principal (aceasta este la începutul jocului 100 dar poate crește în modul Endless) și variabila care reprezintă viață curentă a jucătorului. Variabila Slider.MaxValue reprezintă numărul maxim pe care îl poate avea sliderul iar variabila slider.value reprezintă valoarea curentă, aceasta se schimbă pe parcursul jocului pentru a prezenta viață curentă a jucătorului.

Pentru armele jucătorului în partea de sus a ecranului sunt afișatele gloanțele armei active curente sub forma: număr de gloanțe în încărcător / număr de gloanțe total al armei. În cod de fiecare dată când personajul trage acestea scad iar când numărul de gloanțe în încărcător ajunge la 0 jucătorul trebuie să reîncarce folosind tasta R. Când tasta este apăsată se încarcă încărcătorul și se scade din numărul de gloanțe total numărul de gloanțe încărcate. Jucătorul poate continua să tragă după ce este apăsată tasta R și este finalizată funcția await Task.Delay((int)(reloadTime / Time.timeScale)) care are ca scop să întârzie încărcarea gloanțelor simulând o armă reală.

3.3.5 Moarte

Încă o componentă care aparține jucătorului principal este opțiunea de a muri și de a-și salva scorul în lista de High Score dacă este mai mare decât cele trei scoruri maxime curente. Astfel, jucătorul pornește cu o viață dată care poate să fie mărită în modul Endless dacă acesta cumpăra puteri ajutătoare.

Figura 3.3.5.1 de mai jos reprezintă meniul de final al jocului care apare în modul Endless, când jucătorul are viață curentă mai mică decât 0. Acesta are un loc de input și două butoane, unul care te duce la meniul principal și unul care reîncarcă nivelul curent în locul de input unde jucătorul își poate trece numere pentru a putea fi salvat în tabelul de High Score. În modul de Story meniul o să arate la fel dar nu o să aibă locul de pus numele, deoarece nu există highscore în Story Mode.



Figură 3.3.5.1 - Meniul de moarte [imagine autor]

Figura 3.3.5.2 prezintă funcția de Update() aflată în codul de moarte al jucătorului. În aceasta verificam dacă viata curentă a jucătorului este mai mică sau egală cu 0. Dacă da atunci meniul de final este activat, activăm și componenta din meniu Input Field care reprezinta locul unde jucătorul își scrie numele. Salvam numele într-o variabilă, salvam și scorul curent al jucătorului când moare și setăm timpul jocului cu 0, adică scena este pusa pe pauza.

```
void Update()
{
    if (GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMovement>().currentHealth <= 0)
    {
        ui.SetActive(true);
        inputName.ActivateInputField();
        playerName = inputName.text;
        Score = GameObject.FindGameObjectWithTag("Canvas").GetComponent<Score>();
        Time.timeScale = 0f;
    }
}
```

Figură 3.3.5.2 - Funcția Update() a codului [imagine autor]

În figura 3.3.5.3 este prezentat codul care este activat de apăsarea butonului de Retry. În variabilele a, b și c sunt salvate primele trei cele mai mari scoruri, dacă unul din aceste

variabile nu există atunci se va salva valoarea 0, iar în variabilele de tip string sunt salvate numele respective scorurilor. Dacă unul sau mai multe din acestea nu există atunci se va înlocui cu o variabilă de tip string goală. În funcțiile de tip if se verifică unde se încadrează scorul obținut de jucător și îl salvăm. La final, reîncărcăm scena curentă de la 0 și setăm timpul la normal.

```

public void retry()
{
    int a = PlayerPrefs.GetInt("HighScore1", 0);
    int b = PlayerPrefs.GetInt("HighScore2", 0);
    int c = PlayerPrefs.GetInt("HighScore3", 0);
    string a1 = PlayerPrefs.GetString("HS1Name", "");
    string b2 = PlayerPrefs.GetString("HS2Name", "");
    string c3 = PlayerPrefs.GetString("HS3Name", "");
    if (a < Score.score)
    {
        PlayerPrefs.SetInt("HighScore1", Score.score);
        PlayerPrefs.SetString("HS1Name", playerName);
        PlayerPrefs.SetInt("HighScore2", a);
        PlayerPrefs.SetString("HS2Name", a1);
        PlayerPrefs.SetInt("HighScore3", b);
        PlayerPrefs.SetString("HS3Name", b2);
    }
    else if (b < Score.score && a != Score.score)
    {
        PlayerPrefs.SetInt("HighScore2", Score.score);
        PlayerPrefs.SetString("HS2Name", playerName);
        PlayerPrefs.SetInt("HighScore3", b);
        PlayerPrefs.SetString("HS3Name", b2);
    }
    else if (c < Score.score && a != Score.score && b != Score.score)
    {
        PlayerPrefs.SetInt("HighScore3", Score.score);
        PlayerPrefs.SetString("HS3Name", playerName);
    }
    SceneManager.LoadSceneAsync(SceneManager.GetActiveScene().buildIndex);
    Time.timeScale = 1f;
}

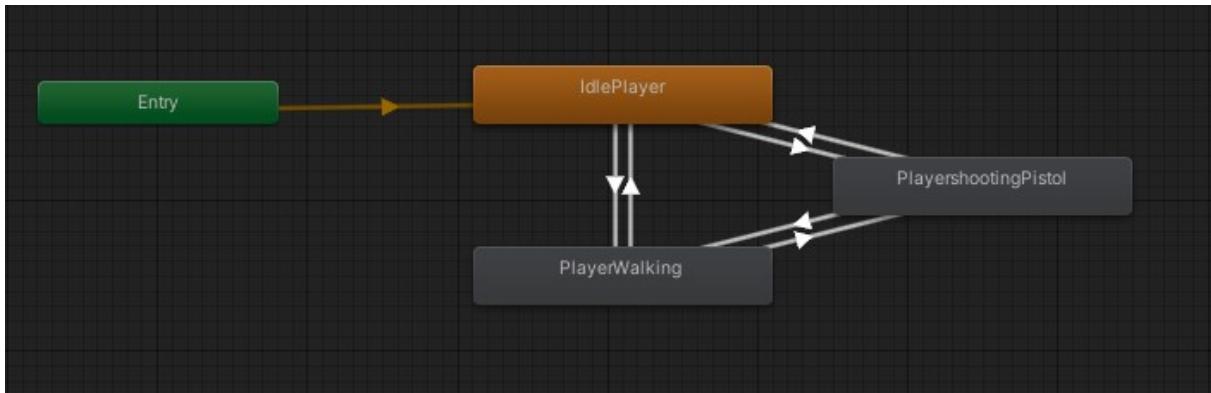
```

Figură 3.3.5.3 - Funcția retry() a codului [imagine autor]

În funcția care este reprezentată de butonul home se respectă codul exact ca la funcția retry(), singura diferență fiind că în loc să reîncărcăm scena curentă încărcăm scena de Home prin folosirea funcției SceneManager.LoadScene(0).

3.3.6 Animații jucător principal

Personajul principal are atașat asupra sa un animator care este activat pe tot parcursul jocului în toate scenele în care apare.



Figură 3.3.6.1 - Animatorul jucatorului [imagine autor]

Astfel, figura 3.3.6.1 prezintă schema animațiilor jucătorului. Când jucătorul nu își mișcă poziția acesta se află în starea de idlePlayer care reprezintă doar spriteul principal. Când acesta se mișcă atunci se trece în playerWalking care are mai multe imagini, iar când este apăsat click stânga atunci jucătorul intră în animația de împușcat care reprezintă spriteul de idle și unul specific împușcăturii. Acestea se schimbă în funcție de valorile de viteză și dacă tasta click stânga este apăsată, acestea fiind aflate în scriptul specific jucătorului.

3.4 Inamici

În acest proiect se regăsesc trei tipuri diferite de inamici care apar pe durata jocului și sunt denumiți după culoarea pălăriei. Aceștia au diferite tipuri de arme care îi scad din viață jucătorului diferite valori.

Inamicul roșu are ca armă principală un pistol.

Inamicul verde are ca armă principală o pușcă și apare în nivelul doi.

Inamicul mov are ca armă principală o pușcă cu lunetă și apare în nivelul trei.



Figură 3.4.1 - Inamici [imagine autor]

3.4.1 Deplasarea folosind A* Pathfinding

Inamicii prezenți în joc se mișcă cu ajutorul inteligenței artificiale folosind algoritmul A* path finding. La fel ca la personajul principal acestora le sunt atribuite funcțiile Rigidbody2D și Box Collider 2D pentru a le oferi caracteristici fizice 2D. Aceștia sunt salvați ca prefabs pentru a putea fi reutilizați în mai multe nivele și folosiți în codul spawner pentru modul Endless. Inamicilor le sunt atribuite trei coduri (seeker, AiPath, Ai destination setter) din librăria A* Pathfinding pentru a calcula drumul optim între fiecare inamic și personajul principal, iar pe lângă acestea am mai creat un cod care este folosit drept controllerul fiecărui inamic. În acest cod sunt reținute viața maximă a fiecărui inamic, viața curentă și distanța actuală între jucător și inamic. Când distanța actuală este mai mică decât o distanță stabilită de joc, care este diferită în funcție de ce tip de inamic este, atunci inamicul o să înceapă să tragă spre jucător.

3.4.2 Enemy Controller

Enemy controller este scriptul principal atașat inamicului care are ca scop împușcatul, să scadă din viață, să dispară de pe hartă și să fie pregătit de utilizarea sa în codul spawner.

```
private void Awake()
{
    player = GameObject.FindGameObjectWithTag("Player").transform;
    enHealth = GetComponentInChildren<EnemyHealth>();
    spawner = GameObject.FindGameObjectWithTag("Spawner1").GetComponent<Enemy1Spawner>();
    score = GameObject.FindGameObjectWithTag("Canvas").GetComponent<Score>();
    gameObject.GetComponent<AIDestinationSetter>().target = player.transform;
    _parent = gameObject.transform.parent.gameObject;
}
```

Figură 3.4.2.1 - Funcția Awake() din codul inamicului [imagine autor]

Fiecare inamic este salvat într-un prefab deja creat, unde este salvat spriteul 2D, codurile pentru A* PathFinding și funcțiile fizice. În figura 3.4.2.1 este prezentat codul de Awake al inamicului. Acesta este rulat când un inamic este activat de codul spawner. Pentru că aceștia să funcționeze corect când sunt activați trebuie să le atribu elemente din scenă. Mai jos explicăm ce înseamnă fiecare variabilă:

player caută în scenă obiectul cu tagul setat ”player” și îi salvează locația de pe hartă.

enHealth reține componența în care este salvat codul vieții ca un element de tip EnemyHealth.

spawner caută în scenă elementul spawner și codul acestuia ca un element de tip Enemy1Spawner.

score caută în scenă elementul cu tag canvas și salvează obiectul de tip Score ca un element Score.

target este setată poziția playerului curentă ca scopul destinației inamicului.

parent găsește părintele inamicului.

```
void Update()
{
    float pldistance = Vector2.Distance(player.position, transform.position);
    if(pldistance<=distrange && nfiretime<Time.time)
    {
        GameObject bullet = Instantiate(bulletPr, fPoint.position, fPoint.rotation);
        Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
        rb.AddForce(fPoint.up * bulletForce, ForceMode2D.Impulse);
        nfiretime = Time.time + firerate;
    }
    if(Health<=0)
    {
        score.addPoint(1);
        Enemy1Spawner.noOfEnemiesAlive--;
        Die();
    }
}
```

Figură 3.4.2.2 - Codul funcției Update() al inamicului [imagine autor]

În figura 3.4.2.2 se prezintă funcția Update() care are ca scop verificarea în fiecare frame a două condiții. În prima condiție se verifică dacă distanța între inamic și jucător este mai mică decât o distanță selectată și dacă timpul între ultimul glonț și următorul este mai mare decât timpul dat. Astfel, folosim codul de instanțiere a glonțului ca cel al jucătorului principal, singura schimbare fiind glonțul ales. A doua condiție este cea care verifică dacă inamicul mai are viață, dacă nu atunci scorul crește cu un punct, numărul de inamici în viață scade cu un punct și este chemată funcția Die();

```
public void TakeDamage(float dmg)
{
    Health -= dmg;
    enHealth.UpdateHealth(Health,maxHealth);
}

1 reference
public void Die()
{
    Destroy(gameObject);
    Destroy(_parent);
}
```

Figură 3.4.2.3 - Codul funcțiilor TakeDamage() și Die() [imagine autor]

Figura 3.4.2.3 prezintă funcția TakeDamage() și Die(). Prima funcție scade din nivelul de viață actual al inamicului numărul dmg, care se schimbă în funcție de arma folosită de jucător și viață care apare pe ecran prin chemarea funcției aflate în scriptul EnemyHealth, iar funcția Die() elimină din scenă obiectul de joc și părintele acestuia.

3.4.3 Viață inamic

Fiecare inamic are atașat un slider care reprezinta viața curentă a acestuia. În figurile de mai jos se poate observa cum acestea se văd în joc și codul specific. În funcția Update() codul are ca scop mutarea vieții inamicului, astfel încât aceasta să îl urmărească folosind transform.position care va deveni poziția actuală plus un vector3. În funcția UpdateHealth sliderul primește valoarea actuală împărțită la viața maximă.

```
public void UpdateHealth(float Health, float maxHealth)
{
    slider.value = Health / maxHealth;
}

@Unity Message | 0 references
void Update()
{
    transform.position = poz+tr.position ;
}
```



Figură 3.4.3.1 - Codul vieții inamicilor și reprezentarea acestuia în joc [imagine autor]

3.4.4 Spawner

Modul Endless al jocului are nevoie de un cod și un obiect în plus pentru fiecare tip de inamic prezent în poveste. Astfel, creăm în scenă trei obiecte empty care reprezintă Spawning pentru inamici, fiecare cu un cod atașat, care ajută la activarea mai multor inamici după o perioadă dată de timp.

```
void Awake()
{
    Instance = GetComponent<Enemy1Spawner>();
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
}
@Unity Message | 0 references
void Start()
{
    _parent = GameObject.Find("Enm");
    StartCoroutine(Spawn());
}
```

Figură 3.4.4.1 - Funcțiile Awake() și Start() ale Spawnerului [imagine autor]

Figura 3.4.4.1 prezintă prima parte a codului specific obiectelor. Funcția Awake() este chemată când scriptul este activat prima dată și o folosim pentru a instanția componenta specifică obiectului și pentru a salva componenta specifică jucătorului. În funcția Start() cautăm părintele obiectului și chemăm corutina specifică funcției Spawn().

```

        int ranX = Random.Range(0, 8);
        Vector3 spawnPosition = new Vector3(o[ranX].x, o[ranX].y, 0);
        Debug.Log(o[ranX].x);
        Debug.Log(o[ranX].y);
        GameObject enemy = Instantiate(Enemy1Prefab, spawnPosition, Quaternion.identity);
        enemy.tag = "Enemy";
        noOfEnemiesAlive++;
    }
    yield return new WaitForSeconds(7);
    if (play.isAlive())
    {
        StartCoroutine(Spawn());
    }
}

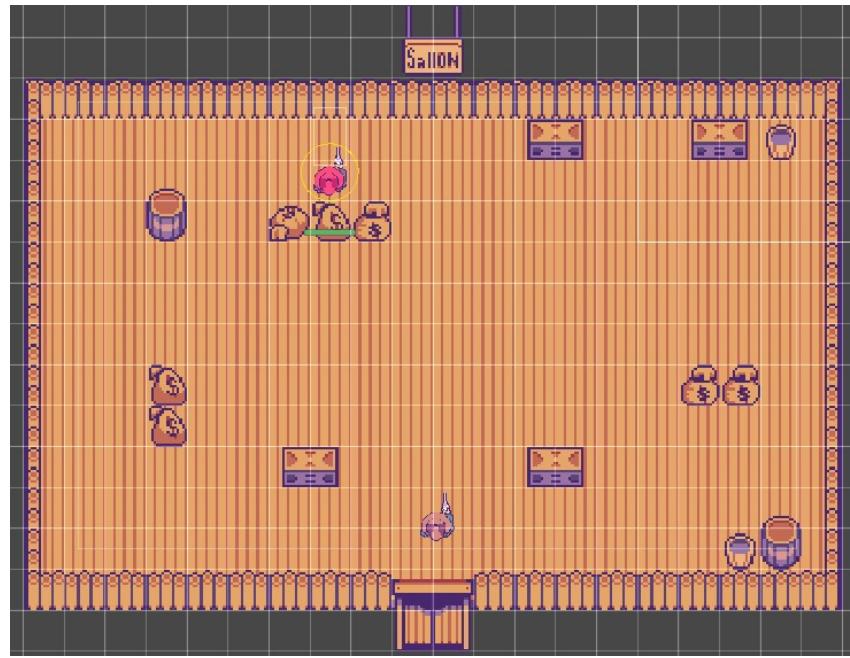
```

Figură 3.4.4.2 - Codul funcției Spawn()[imagine autor]

În figura 3.4.4.2 este prezentată ultima parte a codului din funcția Spawn(). În prima parte creăm o listă cu coordonatele a opt locații în care inamicii pot apărea fără să provoace coliziune cu alte elemente ale hărții. În variabila ranX salvăm un număr oarecare între 0 și 7 care să reprezinte una din locațiile salvate. În variabila de tip Vector3 salvăm coordonatele locației care coincide cu poziția selectată, iar coordonata de tip z o să fie întotdeauna 0, deoarece suntem într-un plan 2D. Instantiem în această locație un obiect prefabricat care reprezintă unul din cele trei tipuri de inamici și folosim Quaternion.identity care înseamnă fără rotație. Îi atașăm inamicul eticheta cu numele Enemy și creștem evidența de inamici aflați pe hartă. La începutul funcției este verificat să nu fie mai mult de un număr dat de inamici pe hartă pentru a nu aglomera harta jocului. În final, folosim yield return new WaitForSeconds(7) pentru a aștepta șapte secunde până a continua codul. După cele șapte secunde verificăm dacă jucătorul principal mai este în viață, dacă da, începem din nou corutina, dacă nu, nu mai activăm inamici noi.

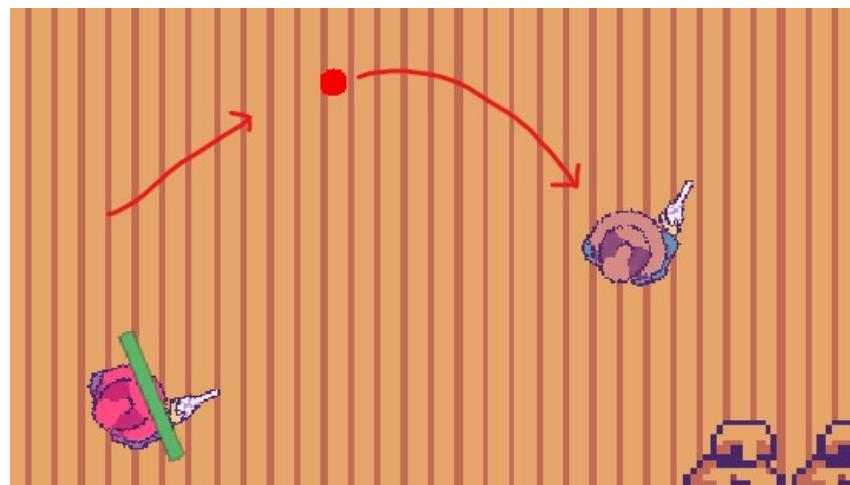
3.4.5 Boss Fight

La finalul poveștii personajul principal se va întâlni cu șeful bandei de neleguiți într-un salon, scena fiind cea prezentată în figura 3.4.5.1.



Figură 3.4.5.1 - Scena în care se petrece acțiunea [imagine autor]

Inamicul arată ca unul normal dar de fapt are mult mai multă viață decât un inamic normal și cel mai important lucru sunt gloanțele cu care acesta trage. Scena în care se petrece acțiunea este una mult mai mică și îngheșuită deci evitarea gloantelor devine mult mai grea, iar jucătorului nu îi sunt oferite puteri ajutătoare ca până acum. Astfel, jucătorul trebuie să se ferească și să fugă de gloanțele inamicului care nu dispar când fac coliziune cu altceva, ci dispar atunci când ating jucătorul sau trece un timp de 5 secunde.



Figură 3.4.5.1 - Cum urmărește glonțul jucătorul [imagine autor]

Gloanțele urmăresc jucătorul folosind `transform.position = Vector2.MoveTowards(transform.position, player.position, speed * Time.deltaTime)` care schimbă poziția glonțului folosind funcția `MoveTowards()` și coordonatele jucătorului în funcție de un timp și o viteză dată.

3.5 Puteri speciale

În timpul jocului, pentru a ajuta jucătorul, pe hartă apar mai multe tipuri de puteri care avantajează jucătorul dacă își îndeplinește misiunile.

3.5.1 Viață

Prima putere se numește HealthPowerUp și are ca scop restabilirea vietii inițiale. Aceasta apare când personajul principal are un scor divizibil cu un număr dat în modul Endless.

```
void Pickup(Collider2D collision)
{
    Instantiate(ef, transform.position, transform.rotation);
    PlayerMovement h = collision.GetComponent<PlayerMovement>();
    h.currentHealth = 100;
    healthb.setHealth(h.currentHealth);
    rnd.enabled = false;
    gameObject.GetComponent<Collider2D>().enabled = false;
    resp = false;
}
```

Figură 3.5.1.1 - Funcția Pickup() a codului puterii de viață [imagine autor]

Dacă condiția de scor este îndeplinită și personajul principal face coliziune cu puterea atunci se apelează funcția Pickup() din figura 3.5.1.1, iar coliziunea este trimisă ca parametru. În funcție, primul lucru pe care îl facem este să instanțiem un efect, numit în cod ef, în locul obiectului ridicat apoi salvăm un obiect de tipul PlayerMovement în h pentru a îi actualiza viața personajului principal în codul său.

Următorul pas actualizează viața curentă în bara de viață a UI-ului și dezactivează textura și colliderul obiectului pentru a dispărea de pe hartă până când este activat din nou. La final salvăm în variabila resp fals pentru a anunța codul că obiectul a dispărut de pe hartă și poate să fie activat din nou când condițiile de scor sunt îndeplinite.

```
List<Vector2> o = new List<Vector2>();
Vector2 z;
z.x = 16; z.y = 6; o.Add(z);
z.x = 15; z.y = -5; o.Add(z);
z.x = -9; z.y = -6; o.Add(z);
int ranX = Random.Range(0, 3);
Vector3 newPos = new Vector3(o[ranX].x, o[ranX].y, 0);
resp = true;
Transform playerPos = player.GetComponent<Transform>();
gameObject.GetComponent<Collider2D>().enabled = true;
transform.position = newPos;
rnd.enabled = true;
oldScore = score.score;
```

Figură 3.5.1.2 - Codul care instanțiază puterea pe hartă [imagine autor]

Astfel, în modul Endless, când condiția de scor este îndeplinită puterea o să apară pe hartă în una din cele trei locații selectate. Codul din figura 3.5.1.2 este prezent în funcția Update() și se activează după verificarea condiției. Dacă condiția de scor este îndeplinită atunci se creează o lista de obiecte de tip Vector2 care salvează trei locații de pe hartă în coordinate de tip x și y. Se alege aleatoriu una din cele trei locații selectate, salvam x și y în o variabilă de tip Vector3, ultima coordonată fiind 0 pentru că suntem în planul 2D. Salvăm în variabilă resp true pentru a anunța codul ca obiectul a apărut pe hartă și setăm Colliderul2D activ pentru a putea interacționa cu el, schimbăm poziția obiectului cu cea dată, activăm renderul pentru a apărea pe hartă și salvăm scorul curent în variabila oldScore pentru a fi sigur că,, nu se apelează din nou pentru același număr.

3.5.2 Gloanțe

Puterea de gloanțe parcurge același mod ca cea de viață, singura diferență fiind condiția pentru a apărea pe hartă, coordonatele unde este afișată și codul din funcția pickup(), unde se aplică puterea atributelor personajelor.

```
void Pickup(Collider2D collision)
{
    Instantiate(ef, transform.position, transform.rotation);
    Shooting h = collision.GetComponent<Shooting>();
    h.noOfBulletsPistol=h.noOfBulletsPistol+20;
    rnd.enabled = false;
    gameObject.GetComponent<Collider2D>().enabled = false;
    resp = false;
}
```

Figură 3.5.2.1 - Funcția de Pickup() al puterii gloanțelor [imagine autor]

Figura 3.5.2.1 reprezintă codul de pickup() al puterii. Când puterea este ridicată în locul acesteia se instanțiază un efect vizual reprezentat în cod de variabilă ef. În variabilă h salvăm componenta de tip Shooting pentru a avea acces la variabilă de gloanțe, adăugându-se pe linia următoare a codului 20 de gloanțe în plus jucătorului. Dezactivăm Colliderul de tip 2D, dezactivăm renderul imaginii pentru a nu mai apărea pe hartă și setăm variabilă resp fals pentru a anunța codul ca obiectul a dispărut de pe harta și poate să fie activat din nou când condițiile de scor sunt îndeplinite.

3.5.3 Viteză

Puterea de viteză disponibilă este puțin diferită de celelalte puteri prezentate mai sus. Această putere nu adaugă ceva jucătorului pe toată durata jocului, oferind doar pentru o perioadă de timp de un anumit număr de secunde o viteză sporită. Pentru a face acest

lucru ne vom folosi de o funcție specifică Unity numită StartCoroutine() care ne permite să repartizăm sarcinile pe mai multe cadre iar, în situațiile în care dorim, să utilizăm un apel de metodă pentru a conține o animație procedurală sau o secvență de evenimente într-un timp dat. Pentru a folosi StartCoroutine(Pickup(collision)) funcția Pickup() trebuie să fie de tip IEnumerator care trebuie să ofere un return de tip yield return.

```

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

IEnumerator Pickup(Collider2D collision)
{
    Instantiate(ef, transform.position, transform.rotation);
    PlayerMovement h = collision.GetComponent<PlayerMovement>();
    h.moveSpeed = 7;
    rnd.enabled = false;
    gameObject.GetComponent<Collider2D>().enabled = false;
    resp = false;
    yield return new WaitForSeconds(5);
    h.moveSpeed = 5;
}

```

Figură 3.5.3.1 - Funcțiile OnTriggerEnter2D() și Pickup() [imagine autor]

Astfel, în figura 3.5.3.1 de mai sus este prezentată funcția care se activează când obiectul intră în coliziune cu jucătorul, respectiv funcția care activează puterea pentru ca jucătorul să beneficieze de ea. În prima funcție când coliziunea are loc între jucător și putere este apelată corutina pentru funcția Pickup(). În funcție este instantiat un efect reprezentat de variabila ef care o să apară în locul obiectului ridicat și salvăm în variabila h componenta de tip PlayerMovement pentru a putea avea acces la viteza jucătorului. Schimbăm viteza curentă a jucătorului din cinci în şapte, , dezactivăm renderul și colliderul de tip 2D. Schimbăm variabilă numită resp în false, semnalând că a fost ridicată și apelăm yield return new WaitForSeconds(5), care are ca rol să așteptăm 5 secunde până se continuă codul, după 5 secunde schimbăm viteza jucătorului în cinci și revenim la normal.

3.5.4 Cheie

Prima parte a jocului reprezintă mai multe nivale și pentru a trece prin acestea personajul principal trebuie să găsească o cheie pe hartă.

```

private void Update()
{
    if (score.score >= x && score.score > 0)
    {
        txt.text = "Find the key to advance";
        rnd.enabled = true;
        gameObject.GetComponent<Collider2D>().enabled = true;
    }
    else if (score.score <= x-1)
    {
        rnd.enabled = false;
        gameObject.GetComponent<Collider2D>().enabled = false;
    }
}

```

Figură 3.5.4.1 - Funcția de Update() din codul obiectului cheie [imagine autor]

În figura 3.5.4.1 este prezentată funcția Update() care verifică dacă condiția ca scorul să fie mai mare sau egal cu un x dat este îndeplinită. Dacă da, variabila din Canvas de tip TextMeshPro își schimbă textul în "Find the key to advance" pentru a anunța jucătorul că și-a îndeplinit scopul, dă enable la renderul imaginii pentru a apărea pe hartă și la colliderul acesta pentru ca jucătorul să poată interacționa cu ea. Dacă condiția nu este îndeplinită, atunci renderul imaginii și colliderul sunt dezactivate pentru a nu apărea pe hartă.

În final, când jucătorul găsește cheia pe hartă și face coliziune cu aceasta se apelează funcția SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1) care are ca rol să încarce și schimbe scenă curentă cu scena care are indexul următor.

3.6 Interfață

3.6.1 Meniu principal

Jocul Outlaw Outpost are un meniu principal lucrat de mine care să se potrivească cu tematica și cromatica jocului. Acesta oferă jucătorului cinci tipuri de alegeri: primele două butoane reprezintă cele două moduri pe care jucătorul le poate alege (Endless și Story), opțiunea highscore care duce jucătorul la tabloul de scor al jocului de tip Endless, options pentru a configura jocul și quit pentru a ieși din joc.



Figură 3.6.1.1 - Meniul principal [imagine autor]

Butonul Play Story apelează SceneManager.LoadScene(SceneManager. GetActiveScene().buildIndex + 1) pentru a încărca următoarea scenă din joc îi timp ce Play Endless

apelează SceneManager.LoadScene(nrpozitieendless) care încarcă scena respectivă modului Endless.

Figura de mai jos notată cu 3.6.1.2 reprezintă meniul de HighScore care este încărcat prin dezactivarea meniului principal și activarea acestuia în aceeași scenă pe canvas. În meniu putem vedea scorurile curente aflate pe primele trei poziții, dacă acestea nu există sau apăsam butonul Reset se vor afișa trei lini cu 0 puncte.



Figură 3.6.1.2 - Meniul HighScore [imagine autor]

Funcțiile din figurile 3.6.1.3 prezintă codurile explicate la punctul anterior. În capitolul 3.3.5 explicăm cum vom salva aceste valori iar în acest capitol prezentăm doar cum le vom afișa. Astfel, în funcția de Start() a meniului schimbăm textul de tip Text Mesh Pro în lista de scoruri maxime folosind PlayerPrefs, iar când apăsăm butonul de reset înlocuim numele și scorurile maxime cu valorile nule.

```
public void ResetHighscore()
{
    PlayerPrefs.DeleteKey("HighScore1");
    PlayerPrefs.DeleteKey("HighScore2");
    PlayerPrefs.DeleteKey("HighScore3");
    PlayerPrefs.DeleteKey("HS1Name");
    PlayerPrefs.DeleteKey("HS2Name");
    PlayerPrefs.DeleteKey("HS3Name");

    text.text = PlayerPrefs.GetString("HS1Name", "") + ": " + PlayerPrefs.GetInt("HighScore1", 0) + " Points" +
        "\r\n" + PlayerPrefs.GetString("HS2Name", "") + ": " + PlayerPrefs.GetInt("HighScore2", 0) + " Points" +
        "\r\n" + PlayerPrefs.GetString("HS3Name", "") + ": " + PlayerPrefs.GetInt("HighScore3", 0) + " Points";
}

void Start()
{
    text.text = PlayerPrefs.GetString("HS1Name", "") + ": " + PlayerPrefs.GetInt("HighScore1", 0) + " Points" +
        "\r\n" + PlayerPrefs.GetString("HS2Name", "") + ": " + PlayerPrefs.GetInt("HighScore2", 0) + " Points" +
        "\r\n" + PlayerPrefs.GetString("HS3Name", "") + ": " + PlayerPrefs.GetInt("HighScore3", 0) + " Points";
}
```

Figură 3.6.1.3 - Funcțiile de Start() și ResetHighscore() din meniul principal [imagine autor]

3.6.2 Meniu setări

A doua scenă a jocului care este accesată apăsând butonul Settings din meniul principal este meniul de setări principale. Acesta este accesat prin ascunderea canvasului din meniul principal, dar cu salvarea backgroundului și afișarea canvasului secundar care reprezintă meniul de setări.



Figură 3.6.2.1 - Meniul de setări [imagine autor]

Figura 3.6.2.1 reprezinta meniul de setări, opțiunile acestuia și butonul back care întoarce jucătorul la meniul principal.

Fullscreen este un checkbox care oferă jucătorului opțiunea de a juca în fullscreen sau windowed.

Resolution este un slider și buton care oferă opțiunea jucătorului de a alege rezoluția jocului.

Graphics este un slider și buton care oferă jucătorului opțiunea de a alege graficile jocului de la very low la ultra, reprezentă scala de performanță și imagine.

Volume este un slider din care se poate seta volumul jocului.

```

public void SetVolume(float volume)
{
    audioMixer.SetFloat("volume",volume);
}

0 references
public void SetQuality(int i)
{
    QualitySettings.SetQualityLevel(i);
}

0 references
public void SetFullscreen(bool fl)
{
    Screen.fullScreen = fl;
}

0 references
public void SetRes(int i)
{
    Resolution r = resolutions[i];
    Screen.SetResolution(r.width,r.height,Screen.fullScreen);
}

```

Figură 3.6.2.2 - Codul meniului de setări [imagine autor]

Figura 3.6.2.2 prezintă funcțiile principale ale meniului de setări. Fiecare funcție reprezintă una din opțiunile prezentate în lista de mai sus: volume, Quality, Fullscreen și Rezoluție. Pentru a seta volumul creăm un audio mixer în care salvăm melodia de fundal și o legăm de sliderul dat, iar când acesta este schimbat se trimite ca parametru în funcție volumul pe care îl setăm în audio mixer. În Unity fiecare calitate a jocului este salvată începând cu 0 (very low) și tot crescând, iar eu mă folosesc de asta pentru a alege fiecare element din slider să fie un int care începe cu 0 și tot creste în funcție de calitate. Așadar, trimitem acest int ca parametru în funcția SetQuality și folosim funcția QualitySettings.setQualityLevel oferită de Unity pentru a trimite numărul calității selectate. Pentru a face jocul fullscreen sau windowed folosim în meniu un checkbox care salvează un bool de tip true dacă checkboxul este selectat sau false dacă este deselectat. Acest bool este trimis ca parametru funcției și folosit în funcția Screen.fullScreen = true/false care reprezintă dacă checkboxul este apăsat sau nu.

```

void Start()
{
    resolutions = Screen.resolutions;
    dropdown.ClearOptions();
    int curres = 0;
    List<string> o = new List<string>();
    for (int i = 0;i< resolutions.Length; i++)
    {
        string op = resolutions[i].width + " x " + resolutions[i].height;
        o.Add(op);
        if(resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.height)
        {
            curres = i;
        }
    }
    dropdown.AddOptions(o);
    dropdown.value = curres;
    dropdown.RefreshShownValue();
}

```

Figură 3.6.2.3 - Codul de obținere a rezoluțiilor [imagine autor]

Pentru a afla rezoluțiile disponibile pe care să le folosim în joc trebuie să aflăm ce rezoluții disponibile are calculatorul utilizând Screen.resolutions. Stergem din lista de rezoluții din meniu de setări rezoluțiile scrise de mână pentru a face loc rezoluțiilor corecte. Astfel, avem nevoie de un algoritm pentru a afișa toate rezoluțiile obținute. Începem prin a crea o listă nouă cu stringuri și parcurgem rezoluțiile obținute. Adaugăm la lista rezoluțiile transformate în string sub forma "lungime x înălțime" și cautăm care rezoluție este cea curentă calculatorului, când o găsim îi salvăm indicele. În final, salvăm în drop down opțiunile, setăm obținea curentă cu cea salvată și actualizăm imaginea pentru a o afișa.

3.6.3 Meniu Pauză

În timpul ambelor moduri jucătorul poate să apese tasta Escape pentru a activa meniu de pauză afișat în figura 3.6.3.1. În acesta se pot schimba setările jocului ca în meniu de setări, se poate reveni la meniul principal și să se revină la joc.



Figură 3.6.3.1 - Meniul de pauză [imagine autor]

Figura 3.6.3.2 reprezintă codul meniului de pauză, setările având deja setate codul prezent și la meniul de opțiuni. La începutul codului creăm două variabile una de tip gameObject care este meniul de setări și o variabilă de tip bool care reprezintă dacă jocul se află în meniu de pauză sau nu. Funcția update verifică dacă jucătorul apasă tasta escape, dacă da se verifică în ce stare se află meniul și se apelează funcția respectivă. Dacă acesta nu este deschis atunci va apărea, iar dacă este deschis, bool = true, acesta se va închide. Dacă Gameispaused este true atunci se cheamă funcția Resume() și obiectul de joc se dezactivează, timpul revine la normal și Gameispaused devine false. Dacă este false

atunci se apelează funcția Pause(), obiectul de joc devine activ și apare pe ecran, timpul jocului este setat la 0, adică se oprește, iar Gameispaused se schimbă în true. Dacă este apăsat butonul de Home atunci se apelează funcția Home() care schimbă scena curentă în cea a meniului principal.

```

public static bool Gameispaused = false;
@Unity Message | 0 references
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (Gameispaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}

```

```

void Pause()
{
    pauseMenu.SetActive(true);
    Time.timeScale = 0f;
    Gameispaused = true;
}
1 reference
public void Resume()
{
    pauseMenu.SetActive(false);
    Time.timeScale = 1f;
    Gameispaused = false;
}
0 references
public void Home()
{
    SceneManager.LoadScene(0);
}

```

Figură 3.6.3.2 - Funcțiile meniului de pauză [imagine autor]

3.6.4 Panouri cu povestea jocului

În modul de Story jucătorului îi sunt oferite părți ale poveștii și sugestii pentru a învăța să avanseze și să se pregătească pentru modul Endless al jocului.



Figură 3.6.4.1 - Primul panou din joc [imagine autor]

Figura 3.6.4.1 prezintă aceste panouri care o să apară între nivele, ajutând jucătorul să învețe mecanicile jocului. Acesta este făcut pe Canvas, păstrând titlul și imaginea de fundal al meniului principal și adăugând două containere și un buton. În containerul din stânga este imaginea cowboyului care ne dă informații. Aceasta este o imagine animată cu ajutorul animatorului oferit de Unity. În dreapta este un container cu text, de fiecare dată când butonul Next este apăsat textul se va schimba. Când butonul Next este apăsat de un număr dat de ori marcăm că au fost citite toate informațiile și încărcăm cu ajutorul SceneManagerului următoarea scenă din joc.

3.7 Camera

Pe parcursul jocului apar două tipuri de camere: camera principală și harta. Camera principală este ancorată de sus, fiind un joc 2D acesta trebuie să aibă o setare care urmărește personajul principal pe tot parcursul jocului, rămânând tot timpul în centrul ecranului.

```
void Update()
{
    transform.position = new Vector3(t.transform.position.x, t.transform.position.y, transform.position.z);
```

Figură 3.7.1 - Funcția Update() al camerei [imagine autor]

În figura 3.7.1 este prezentat codul pentru camera principală, aceasta are setările în Unity obișnuite unei camere 2D singura diferență fiind că în cod o programăm să urmărească jucătorul principal. În funcția Update() schimbăm la fiecare frame poziția camerei folosind o variabilă t care reprezintă playerul. Astfel, poziția nouă a camerei o să fie un Vector3 care reține coordonatele x, y și z ale jucătorului și setează poziția camerei cu acestea.

Pentru a crea o hartă de tip Mini Map duplicăm camera principală, îi modificăm mărimea și o salvăm ca textură. Astfel, în canvas creăm două RawImages una peste alta și pe cea din spate o mărim puțin pentru a crea o iluzie de border a hărții. În a două imagine înlocuim textura cu camera salvată și în locul imaginii va apărea camera.

```
void LateUpdate()
{
    Vector3 np = player.position;
    np.z = transform.position.z;
    transform.position = np;
```

Figură 3.7.2 - Funcția de LateUpdate() a Mini Mapului [imagine autor]

În figura 3.7.2 prezentăm codul hărții Mini Map folosind un LateUpdate(), care se apelează după ce toate funcțiile de tip Update sunt chemate. Avem nevoie de acest LateUpdate() pentru a fi siguri că harta arată toate acțiunile încheiate. Astfel, în funcție salvăm poziția jucătorului, păstrăm poziția z a camerei și transformăm poziția camerei doar în punctele de coordonată x și y.

3.8 Magazin

În modul Endless al jocului se regăsește opțiunea de magazin. Aceasta oferă jucătorului mai multe opțiuni de arme și puteri pentru a-l ajuta să câștige cat mai multe puncte.

```

bool x = false;
@Unity Message | 0 references
public void OnTriggerStay2D(Collider2D collider)
{
    if (collider.CompareTag("Player")){
        x = true;
    }
}
@Unity Message | 0 references
void Update()
{
    if (Input.GetKeyDown(KeyCode.Alpha9) && x==true)
    {
        shop();
    }
}

public void shop()
{
    shopMenu.SetActive(true);
    Time.timeScale = 0f;
    Gameinshop = true;
}

public void Resumeshop()
{
    shopMenu.SetActive(false);
    Time.timeScale = 1f;
    Gameinshop = false;
}

```

Figură 3.8.1 - Funcțiile pentru activarea magazinului [imagine autor]

Pe hartă în fața salonului se regăsește un personaj de tipul NPC care are atașat asupra sa un cod, un rigid body2D și un BoxCollision2D. Când personajul principal face coliziune cu acesta prima dată și apasă tasta 9 se va deschide un meniu de magazin ca în figura 3.8.2. Astfel, în codul 3.8.1 de mai sus este prezentat modul în care se deschide acest meniu de magazin. În primul rând, verificăm cu ajutorul funcției OnTriggerStay2D dacă jucătorul se află în raza de coliziune a NPC-ului folosind și condiția că personajul face coliziune cu un obiect care are tagul specific jucătorului. Dacă da, atunci salvăm într-o variabilă de tip Bool true, după aceea pe tot parcursul jocului personajul principal poate să interacționeze cu magazin oriunde apăsând tasta 9. Când tasta este apăsată se apelează funcția de tip void shop() în care activăm în scena gameObjectul care reprezintă meniul magazinului, oprim timpul setând timeScale cu 0 și salvăm într-o variabilă de tip bool faptul că suntem în meniu. Când butonul din fig 3.8.2 pe care scrie Resume este apăsat atunci este apelată funcția Resumeshop() care are ca scop revenirea în joc. Astfel, dezactivăm obiectul de joc care este meniul, setăm timpul înapoi la 1, variabila de tip bool devine false și se revine la joc.



Figură 3.8.2 - Meniul de magazin [imagine autor]

In figura 3.8.2 fiecare imagine este reprezentată de un buton, iar fiecare buton are atașat la el o funcție care schimbă aspecte ale jocului. Astfel în figura 3.8.3 se prezintă codurile care se activează când butoanele sunt apăsate, pe lângă butonul de Resume existând și cinci butoane care oferă diferite puteri jucătorului, două dispărând după ce sunt apăsate și trei care rămân până când devin maxime. Cele două butoane de pe rândul al doilea sunt conectate cu funcțiile din figura 3.8.3. Acestea au ca scop cumpărarea și deblocarea a două tipuri de arme noi: pușca cu lunetă (care oferă o daună mai mare dar cu gloanțe mai puține) sau pușcă (care trage cu trei gloanțe deodată și are mai multe gloanțe).

Astfel, când butonul cu pușca cu lunetă este apăsat se apelează funcția cu numele buySniper care verifică dacă jucătorul are destui bani să cumpere arma. Dacă da, pușca cu lunetă devine accesibilă în joc schimbând valoarea variabilei de tip bool pusca cu lunetă din Shooting, scădem din bani cât costăarma, afișăm banii curenți după ce am scăzut cât a costatarma și facem ca butonul de a cumpăraarma să nu mai fie activ. Când butonul de pușcă este apăsat se folosește aceeași metodă pentru a îl activa, diferența fiind că costăarma și faptul că schimbăm variabila sg din componenta shooting.

Pe lângă deblocarea armelor mai există cele trei avantaje pentru jucător care au ca

rol mărirea vieții, a daunelor făcute de arme și cumpărarea de gloanțe. Astfel, cele trei butoane de pe prima linie a meniului sunt reprezentate fiecare de o funcție proprie. Prima are ca scop mărirea vieții curente și face asta adăugând la viața maxima plus 10 viață, setând în player controller viața nouă și automat schimbând viața curentă la noua viață maximă. A două are ca scop să modifice câtă viață scade un glonț din viața inamicilor folosind din codul Shooting funcția updamage care adaugă la fiecare glonț al armelor încă cinci atac. Ultima putere îi oferă jucătorului mai multe gloanțe adăugând la fiecare arma un număr setat de gloanțe noi.

```

public void buySniper()
{
    if (money >= 100)
    {
        s.sniper = true;
        money -= money - 100;
        moneytxt.text = money.ToString() + " Coins";
        snip.SetActive(false);
    }
}
0 references
public void buySg()
{
    if (money >= 50)
    {
        s.sg = true;
        money -= money - 50;
        moneytxt.text = money.ToString() + " Coins";
        sg.SetActive(false);
    }
}

```

Figură 3.8.3 - Codul funcțiilor de cumpărat arme [imagine autor]

Capitolul 4

Concluzii

Industria de jocuri video este una dintre cele mai populare la momentul actual și, de la apariția ei, a reușit să se dezvolte odată cu modernizarea tehnologiei.

Obiectivul principal al acestei lucrări a fost să dezvolt un joc complet 2D top-down, plecând de la crearea și programarea personajelor, până la dezvoltarea unei povești și utilizarea mecanicilor complexe cu ajutorul informațiilor dobândite pe parcursul anilor de studiu dar și în plus.

În lucrare am prezentat mai multe procedee de bază pentru construcția unui joc.

Contribuția mea constă în crearea de obiecte originale, diferite de cele prezentate în coduri open source de pe internet.

Jocul Top-Down Shooter 2D creat de mine are ca personaj principal un cowboy care pleacă într-o poveste captivantă întreținută de atmosfera misterioasă a Vestului Sălbatic. Personajul trebuie să apere un oraș atacat de o bandă de tâlhari.

Jocul meu oferă o experiență relaxantă și distractivă cu ajutorul cromaticii și a stilului, evocând astfel atmosfera necunoscutului epocii de cucerire a vestului american. Aceasta este accentuată și de diferențele arme specifice perioadei (pistolul, pușca și pușca cu lunetă) precum și de puterile speciale care fac jocul mult mai interesant.

Cel mai important aspect al jocului este modul Endless care pune la încercare jucătorul prin obiectivul de a trai cât mai mult și de stabilirea unui scor cât mai mare. Jocul este interactiv, personajul putându-se folosi de mai multe puteri care apar pe hartă sau care pot fi și cumpărate de la un magazin.

În acest moment este un joc complet dar care poate să fie dezvoltat cu ușurință. În viitor, jocului i se pot adăuga o multitudine de dezvoltări:

- îmbunătățirea asseturilor de joc pentru a le spori calitatea și a le diferenția;
- crearea de personaje noi din care jucătorul poate să aleagă, fiecare având caracte-ristici proprii (viață mai multă, gloanțe mai multe, viteză mai mare etc), precum și inamici diferenți, pentru a crea o experiență cât mai amplă și original;

- adăugarea de niveluri Boss Battle noi utilizând un cod de inteligență artificială mult mai complexă și NPC-uri cu care personajul principal poate să interacționeze.
- crearea unui sistem de scor mai dezvoltat, în care poți să salvezi scoruri personale și ale prietenilor în funcție de nume și parolă.
- dezvoltarea firului narativ adăugând un sistem care influențează parcursul jocului în funcție de alegerile făcute de jucător.

Bibliografie

- [1] Pixel Archipel, *Western Fps 2D*, Accesat: 02-05-2023, Pixel Archipel, 2016, URL: <https://www.gamedevmarket.net/asset/western-fps-1-5288>.
- [2] Brackeys, *2D Movement in Unity (Tutorial)*, Accesat: 28.04.2023, Youtube, 2018, URL: <https://www.youtube.com/watch?v=dwcT-Dch0bA>.
- [3] Brackeys, *TOP DOWN SHOOTING in Unity*, Accesat: 29.04.2023, Youtube, 2019, URL: <https://www.youtube.com/watch?v=LNLV0jbrQj4>.
- [4] 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, URL: https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games.
- [5] Fandom.com, *Arthur Morgan*, Accesat: 01-06-2023, Villains Wiki, 2020, URL: https://villains.fandom.com/wiki/Arthur_Morgan.
- [6] Lucasfilm Games, *Maniac Mansion*, Accesat: 01-06-2023, Lucasfilm Games, 1987, URL: https://store.steampowered.com/app/529890/Maniac_Mansion/.
- [7] Jared Halpern și Halpern, „Developing 2D Games with Unity.”, în (2019), DOI: [10.1007/978-1-4842-3772-4](https://doi.org/10.1007/978-1-4842-3772-4).
- [8] Adam Lobel Isabela Granic și Rutger C. M. E. Engels, „The benefits of playing video games”, în *American psychologist* 69.1 (2014), pp. 66–75, DOI: [10.1037/a0034857](https://doi.org/10.1037/a0034857), URL: <https://doi.org/10.1037/a0034857>.
- [9] Microsoft, *A tour of the C# language*, Accesat: 29-05-2023, Microsoft, 2023, URL: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>.
- [10] N.Nilsson, „Artificial Intelligence: A New Synthesis”, în (1998).
- [11] Suleiman Mohamed Osman, „The Role of Video Games on Childhood Studying.”, în *Daha International University Academic Journal (DIUAJ)* 2.1 (2023), pp. 58–71.
- [12] Estúdio Vaca Roxa, *Generic Old West Pack*, Accesat: 23-04-2023, Estúdio Vaca Roxa, 2020, URL: <https://bakudas.itch.io/generic-oldwest-pack>.
- [13] SamuelLee, *Cowboy (Animated Pixel Art)*, Accesat: 23-04-2023, SamuelLee, 2020, URL: <https://samuellee.itch.io/cow-boy-animated-pixel-art>.

- [14] id Software, *Wolfenstein 3D*, Accesat: 01-06-2023, Bethesda Softworks, 1994, URL: https://store.steampowered.com/app/2270/Wolfenstein_3D/.
- [15] Unity Technologies, *Unity Documentation*, Accesat: 29-05-2023, Unity, 2023, URL: <https://docs.unity3d.com/ScriptReference>.
- [16] Wikipedia, *Adobe Photoshop*, Accesat: 29-05-2023, Wikipedia, 2023, URL: https://ro.wikipedia.org/wiki/Adobe_Photoshop.
- [17] Wikipedia, *Hearthstone*, Accesat: 01-06-2023, Wikipedia, 2023, URL: https://en.wikipedia.org/wiki/Gameplay_of_Hearthstone.
- [18] Wikipedia, *Unity (game engine)*, Accesat: 29-05-2023, Wikipedia, 2023, URL: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)).