UNIVERSITATEA DINBUCUREŞTI

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

The Knight

Absolvent

Mihai Lazăr

Coordonator ştiinţific

Lector Ştefan George Popescu

Bucureşti, iunie 2023

Rezumat

În această lucrarea, voi explora dezvoltarea unui joc video folosind motorul Unity, lansat în anul 2005 la Apple Worldwide Developers de către dezvoltatorul Unity Technologies, din punct de vedere al designului dar și din punct de vedere istoric. Totodată voi explora și relevanța jocurilor video în industrie precum și impactul lor asupra culturii și societății. Compnentele principale pe care le voi dezvolta vor fi: jocul propriu-zis cum ar fi caracterele și inaminicii, sisteme de meniuri, audio și de salvare.

Abstract

In this paper, I will explore the development of a video game using the Unity engine, released in 2005 at the Apple Worldwide Developers Conference by Unity Technologies, from both a design and historical perspective. I will also delve into the relevance of video games in the industry, as well as their impact on culture and society. The main components I will develop include the actual game, such as characters and enemies, menu systems, audio, and saving mechanisms.

Cuprins

Capitolul 1

Introducere

1.1 Descrierea lucrării

The Knight este un joc 2D, side view, action, adventure, platformer, level based, realizat în Unity, similar cu jocurile precum *Hollow Knight* și *Ori and the Will of the Wisps*. Subgenul principal în care se află este acela de metroidvania care are ca reper de dezvoltare principal neliniaritatea gameplay-ului.

Premisa este una simplă, jucătorul se trezește la marginea unui regat, și pornește intr-o aventură să descopere mai multe despre el si despre lumea în care se află. Pe parcursul jocului el descoperă că nu este bine venit în această lume, și va trebui să exploreze lumea și să răzbească provocăriile pe care le va întâmpina.

Unity este un motor de joc dezvoltat de Unity Technologies, anunțat și lansat pentru prima dată în iunie 2005 la Apple Worldwide Developers Conference ca motor de joc Mac OS X.

Am ales Unity pentru dezvoltarea aplicației, deoarece Unity oferă suport pentru platforme multiple, oferind capacitatea de a dezvolta jocul o singură dată și de al lansa pe mai multe platforme. De asemenea Unity oferă un set bogat de instrumente și funcționaliăți cum ar fi sistemele de animație, fizică, sunete, muzică etc. care ajută la o dezvoltare, testare și modificare a unei aplicați mult mai rapidă.

În realizarea lucrării am dezvoltat un sistem de meniuri, sistem de mișcare pentru caracterul jucătorului, o inteligență artificială simplă pentru inamici, un sistem de salvare, sistem de animații pentru inamici si caracterul jucătroului, designul nivelurilor.

De asemenea folosind Unity Store am achiziționat asset-uri gratis cum ar fi sprite-ul și

animațiile caracterului jucătorului și al inamicilor și sprite-uri pentru elemente din interfața utilizatorului și teren.

1.2 Motivația

În opinia mea, jocurile video sunt o formă de artă ce combină avansul tehnologic, cu formele mai vechi de artă. Pentru a dezvolta un joc ai nevoie de desen pentru sprite-uri, sculptură pentru modele 3D, compoziție muzicală, scris pentru poveste.

Pe lângă partea artistică dezvoltatori au nevoie de o viziune pentru a putea crea o experiență de neuitat. Realizarea acesteia se poate face prin mecanicile jocului, dificultatea jocului, flow-ul jocului etc. Flow-ul reprezintă capacitatea jocului de a captiva jucătorul, ce se poate realiza prin poveste, efecte vizuale, mecanicile jocului etc.

În cazul meu, jocurile dificile precum *Dark Souls, Elden Ring, Hollow Knight,* mi-au lăsat o impresie foarte bună, așa că am decis să încerc să dezvolt un joc similar cu Hollow Knight. Momentul de triumf pe care l-am simțit după ce am reușit să răzbesc o provocare pe care un joc o pune în fața mea au fost de neuitat, ceea ce m-a împins în direcția de a dori să ofer aceeași experiență mai departe.

De asemena, curiozitatea pentru a afla mai multe despre dezvoltarea jocurilor pe calculator, împreună cu pasiunea mea pentru ele, mi-au influențat alegerea de a dezvolta un joc pentru acestă lucrare.

1.3 Relevanța temei

Jocurile video sunt o formă de divertisment interactivă care reușesc să creeze experiențe virtuale prin intermediul tehnologiei informației. Jocurile video pot fi consumate prin diferite modalități, cum ar fi: calculatoarele, telefoanele mobile, consolelor, căștilor de realitate virtuală, etc.

Jocurile video au o varietate largă de genuri și stiluri în care pot apărea, cum ar fi genuri de acțiune, aventură, strategie, puzzle, platforming, se pot afla în formate singleplayer, online multiplayer, 2D, 3D, side view, top-down view, etc. Atât timp cât

există creativitate în ființa umană jocurile pot apărea în orice combinație de stiluri și genuri.

Veniturile industriei jocurilor video în 2022 au fost de 184.4 miliarde de dolari [3], comparativ cu industria muzicii cu 26.2 miliarde de dolari [2] si industria filmelor cu 77 miliarde de dolari [1] în același an, se observă că industria jocurilor video are venituri cu 103.2 miliarde de dolari mai mari decat industria jocurilor si filmelor combinată (Fig 1.1).

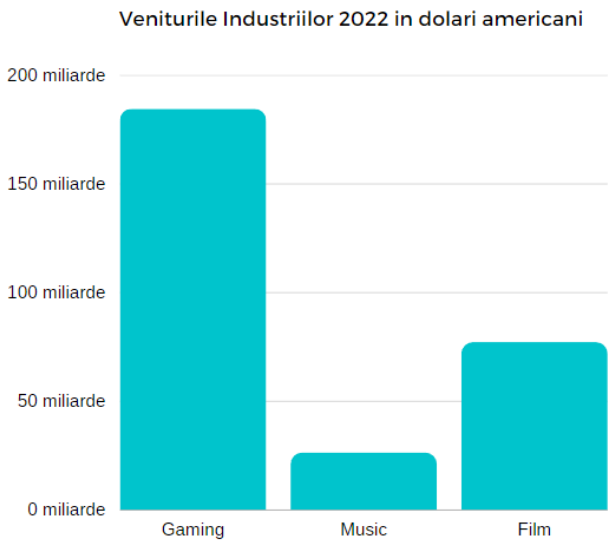

Fig 1.1

Jocurile video pot fi folosite și ca simulări ale lumi înconjurătoare, cum ar fi simularea interacțiunilor corpurilor cerești cum ar fi stelele, planetele, găurile negre, etc. ce se regăsește în jocul *Universe Sandbox*, simularea unui ecosistem uman cum ar fi jocurile open world ca și *Grand Theft Auto, Red Dead Redemption*, etc., simularea condusului de mașini, bărci, avioane, rachete ce se regăsesc în jocurile precum *Forza Horizon, Microsoft Flight Simulator, Spaceflight Simulator*, etc.

De asemenea jocurile influențează și dintr-un punct de vedere filosofic. De exemplu teoria simulării care spune că realitatea înconjurătoare este de fapt o simulare, a fost inspirată din faptul că specia umană a ajuns în punctul de a putea simula lumi întregi. Nu e greu să ne imaginăm că ar putea exista o specie suficient de avansată tehnologic încât să ne poată simula întreaga lume.

1.4 Repere istorice

"În octombrie 1958, fizicianul William Higinbotham a creat ceea ce se crede a fi primul joc video. Acesta era un joc de tenis foarte simplu, similar cu jocul clasic din anii 1970, Pong, și a avut un mare succes la o zi a porților deschise la Laboratorul Național Brookhaven." [5]

Donkey Kong, lansat în anul 1981 de către Nintendo, este primul joc platformer creat, scopul jucătorului este să urce scări și să evite butoaiele aruncate de către gorila Donkey Kong, în speranța de a salva femeia capturată de către gorilă.

Metroidvania este un subgen de jocuri de acțiune, aventură, ce se axează pe neliniaritate și de obicei prezintă o hartă mare interconectată. Numele metroidvania provine de la o combinație de nume ale jocurilor *Metroid* și *Castlevania.*

Metroid, lansat în anul 1986, creat de către Gunpei Yokoi, și deținut de către Nintendo, prezintă o femeie vânătoare de recompense pe nume Samus Aran, care explorează lumi extraterestre în scopul de a găsi ceea ce caută.

Castlevania, lansat în anul 1986, dezvoltat și publicat de către Konami, unde eroul are scopul de a explora castelul lui Dracula și de a răzbi legendarul vampir în scopul de a salva omenirea.

Grand Theft Auto 3, lansat în anul 2001, și publicat de către Rockstar Games, este primul joc open world de success, unde jucătorul poate să trăiască o viață periculoasă de mercenar. Pe parcursul jocului, personajul întâmpină diferite grupuri de oameni de la care primește misiuni, pe care acesta trebuie să le îndeplinească.

Dark Souls, lansat în 2011, dezvoltat de FromSoftware și publicat de către Namco Bandai Games, a fost primul joc de succes care a promovat dificultatea jocurilor în industrie. Acțiunea jocului se desfășoară întro lume fantastică plină de dragoni, spirite și alte besti fantastice, dar care este în același timp necruțătoare. Scopul personajului este să exploreze lumea să răzbească provocările și să obțină tronul regatului în care se află.

Hollow Knight a fost dezvoltat în mare parte de 3 persoane din Australia, Ari Gibson, William Pellen si David Kazi, ce au format echipa Team Cherry și cu ajutorul unui kickstarter de 50 de mii de dolari austrilieni, au reușit să lanseze jocul pe data de 24.02.2017. Până în anul 2019 au reușit să vândă peste 2.8 millioane de copi [4].

Hollow Knight este un joc dificil și nu oferă opțiunea pentru a alege dificultatea, ceea ce dintr-un punct de vedere al designul-ui îl consider benefic, dezvoltatorii având o mai mare putere de a-și exprima viziunea prin faptul că toți utilizatori vor avea experiențe asemănătoare.

Capitolul 2
Prezentarea Aplicației

2.1 Experiența utilizatorului

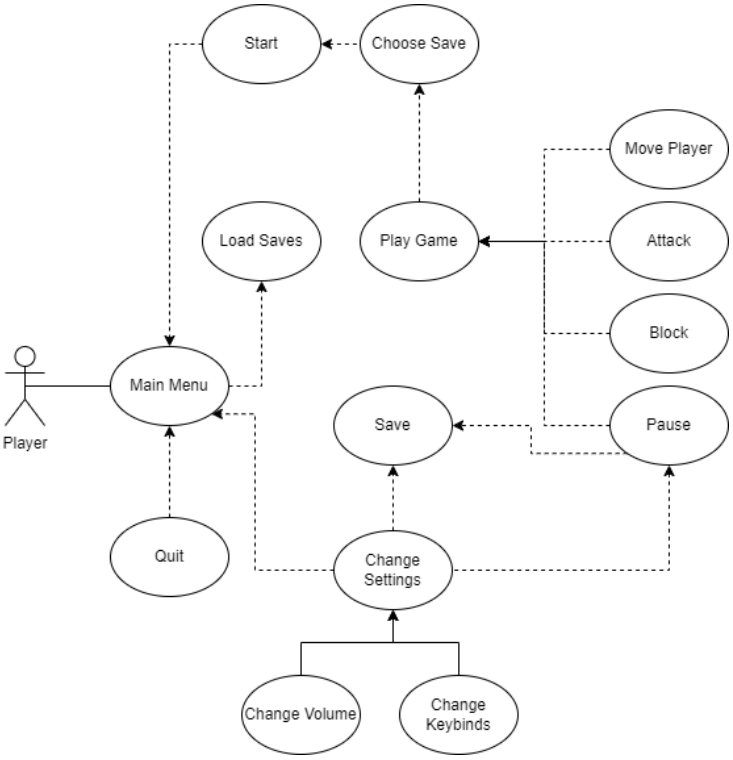

Fig 2.1

Utilizatorul când pornește aplicația este întâmpinat de un meniu princpal de unde poate să înceapă jocul, poate să-și schimbe setările sau să închidă complet aplicația.

În momentul când va apăsa butonul de start un meniu de unde își va putea alege salvarea va apărea. În acest meniu sunt prezente 3 sloturi dintre care poate alege. Pe fiecare slot vor apărea după logică următoarele opțiuni:

- Dacă există pe calculator salvarea respectivă slotului, atunci utilizatorul va avea opțiunea de a continua jocul de unde a rămas sau de a șterge salvare.
- Dacă nu există pe calculator salvarea respectivă slotului, atunci utilizatorul va avea doar opțiunea de a porni un joc nou.

În momentul când va apăsa butonul de opțiuni, utilizatorul va fi întâmpinat de un meniu unde își va putea alege dacă vrea să-și schimbe setările la sunete, ce reprezintă volumul muzicii sau volumul efectelor sonore, sau setările de configurație al butoanelor, adică își vor putea alege pe ce taste vor putea apăsa ca să execute in joc acțiuni de tipul: mișcare stânga, mișcare dreapta, sărit, atac, blocat sau interacțiune cu obiecte. În același timp va avea opțiunea de a reseta

configurația butoanelor la cea inițială care este: A – mișcare stânga, D – mișcare dreapta, SPACE – sărit, CLICK-STÂNGA – atac, CLICK-DREAPTA – block, E – interacțiune. Doresc să menționez că utilizatorul poate folosi tasta ESCAPE pentru a da toggle jocului pe pauză și a naviga înapoi în meniu, tastă pe care o consider că nu ar trebui modificată, așa că utilizatorul nu va primi opțiunea să o modifice.

2.2 Experiența utilizatorului în joc

În timpul jocului, utilizatorul se va întâlni cu 3 tipuri de inamici, inamici melee care se pot interacționa cu jucătorul doar din apropiere sau vor patrula dacă jucătorul nu a fost detectat, inamici zburători care vor sta inactivi cât timp nu vor detecta jucătorul sau vor spura spre jucătorul încercând să-l rănească la contact, și inamici ranged, care vor sta pe loc și la detectarea jucătorului vor trage cu un proiectil în direcția jucătorului.

În continuare, jucătorul se va putea mișca stânga, dreapta, va putea sări, va putea ataca și bloca atacurile inamicilor. În timp ce blochează jucătorul trebuie să înțeleagă că blocarea atacurilor se face doar în partea cu scutul, adică dacă inamicul este în dreapta jucătorului și jucătorul blochează spre dreapta atacul va fi blocat si jucătorul nu va pierde din viață, dar dacă blochează spre stânga și inamicul lovește din direcția dreaptă a jucătorului, jucătorul va fi lovit și va pierde viață. Menționez că jucătorul va putea sări doar dacă se află cu picioarele pe o platformă solidă de teren. Jucătorul are opțiunea să sară de pe pereți dar doar în cazul în care nu este în contact cu podeaua și este în contact cu peretele.

Fig 2.2

Pe parcursul jocului utilizatorul va putea interacționa cu puncte de salvare (Fig 2. 2) sub formă de steag roșu. Interacțiunea se întâmplă doar dacă jucătorul se află suficient de aproape de steag. Menționez că interacțiunea cu aceste puncte de salvare regenerează viața curentă a jucatorului la maximul posibil.

În deplasare prin lumea jocului, utilizatorul se va lovi de niște “uși”, care reprezintă o adâncitură în relieful terenului (Fig 2.3), în momentul în care jucătorul intră prin aceste “uși”, utilizatorul va fi transportat în altă cameră de pe hartă.

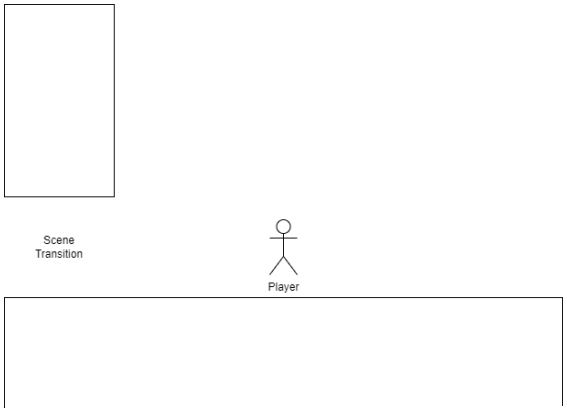

Fig 2.3

Utilizatorul poate pune oricând jocul pe pauză prin simpla apăsare a butonului ESCAPE, o dată apăsat acest buton un meniu de pauză va apărea în fața utilizatorului, ce va avea următoarele opțiuni: un buton de “Continue” pentru a putea ieși din meniul pauză și a reveni la joc, menționez că apăsarea butonului ESCAPE are același efect, un buton pentru setări care oferă aceleași interacțiuni ca și în meniul principal, și un buton de întoarcere la meniul principal, care va salva datele jucătorului și îl va duce la meniul principal.

Capitolul 3

Dezvoltarea aplicației

3.1 Dezvoltarea jocului

3.1.1 Tilemap

În dezvoltarea aplicație a trebuit sa i-au o decizie în legătură cu modalitatea de construire a hărții, pentru ușurarea procesului de construire a terenului am decis să folosesc un tilemap de tip dreptunghiular. Un tilemap dreptunghiular împarte întreaga scenă într-un grid de 1x1 unități unde se pot plasa tile-urile, ca niște piese de puzzle.

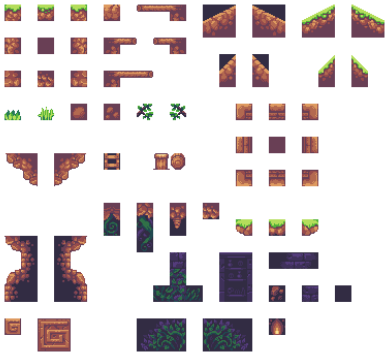

Fig 3.1

Pentru a obține tile-uri am importat un tileset de pe Unity Asset Store [6], acest asset are mai multe componente dar eu tot ce folosesc din el sunt tile-urile din figura 3.1.

Pentru a folosi tile-urile imagine, acestea trebuie pregătite, pentru a le pregăti trebuie urmați următorii pași: trebuie selectată imaginea în editorul unity, trebuie selectat sprite mode pe multiple, trebuie să adăugăm câți pixeli pe unitate ocupă un tile, în cazul meu 16 este numărul perfect, după ce am selectat aceste opțiuni apăsăm pe apply și urmăm să facem tăierea propriu zisă a imagini. Pentru a face această tăiere trebuie să deschidem imaginea în sprite editor selectăm tipul de slice pe grid by cell size, alegem pixel size-ul de (X = 16, Y = 16), apăsăm butonul de slice și după aplicăm modificările apăsând pe butonul apply.

Acum că avem un tilemap pregătit trebuie să creăm o paletă, pentru a crea o paletă de tile-uri, pentru a face asta ne ducem pe window -> 2D -> tile palette, după ce am deschis paleta creăm o paletă nouă, o salvăm în proiect și suntem gata de a construii hărțile perfect.

Pentru accest proiect am creat două tilemap-uri, unul pentru teren și unul pentru

background. Am creat 2 layere unul pentru background si unul pentru teren în așa fel încât layer-ul de teren, iar pentru fiecare tilemap am selectat în tilemap renderer layer ul respectiv fiecărui tilemap.

Pentru layer-ul de teren va trebui să-l configurez în așa fel încât player-ul să poată interacționa cu el. Pentru a realiza acest lucru la tilemap va trebui să adau două componente, un Tilemap Collider 2D și un Rigidbody 2D. Dacă pe configurația aceasta dăm start la scenă, observăm ca tot terenul cade, pentru a rezolva această problemă, în componenta Rigidbody 2D selectăm Body Type-ul pe static, dacă vom da play acum vom observa că terenul nu va mai cădea.

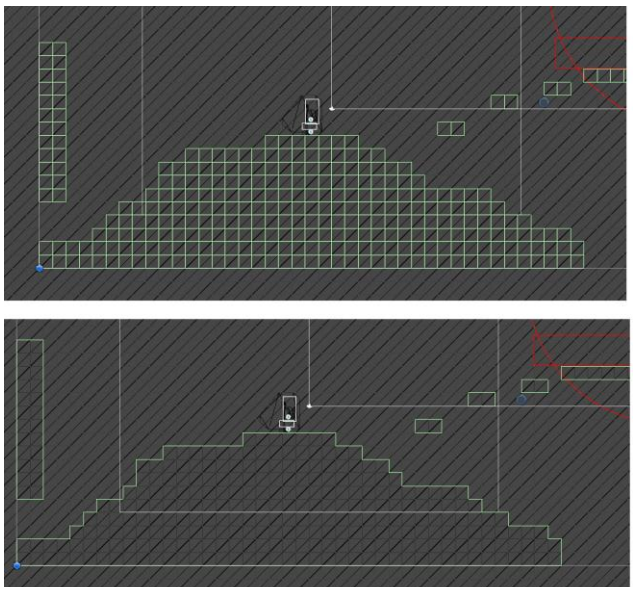

Fig. 3.2

Dacă schimbam shading mode-ul in wireframe observăm că fiecare tile de teren va avea colliderul ei, ceea ce este nenecesar, pentru a optimiza acest lucru vom adăuga componenta Compossite Collider 2D pe tilemap-ul terenului și în Tilemap Collider 2D vom bifa căsuța Used By Composite și vom obține efectul din figura 3.2.

3.1.2 Player

Pentru player am importat de pe Unity Asset Store [7] modelul de la figura 3.3. Pentru început am creat în scenă un nou obiect 2D de tip pătrat și am schimbat sprite-ul din componenta Sprite Renderer cu sprite-ul HeroKnight_0 și am ales sorting layer-ul egal cu Player, am mărit scale-ul obiectului pe axele x și y la 2.

Fig 3.3

Mai departe am adăugat componenta de rigidbody 2D unde am adăugat un material de fizici 2D care are proprietate de frecare și elasticitatea egale cu 0 pentru a ignora frecarea și reacțiunea la coliziunea cu terenul, și un box collider 2D pe care l-am configurat puțin mai mic decât sprite-ul lui, deoarece așa pot evita aparența de levitare a sprite-ului deasupra pământului.

Pentru orientarea corectă a obiectelor în scena am creat o clasă care va roti pe axa y cu 180 de grade pentru fiecare flip și pe care am adăugato ca și componentă la obiectul player.

Mai departe am adăugat componenta animator (fig 3.4) care are rolul de a anima mișcările caracterului jucătorului, caracterul se poate afla în următoarele stări: Idle, Running, Falling, Jumping, Attack, Block, Hurt, Wallslide, Death și parametrul de tip int pentru starea de mișcare, 3 de tip trigger pentru attack, hurt și block, și 2 de tip bool pentru isDead și walled. Logica este următoarea caracterul se află în una din cele 4 stări de mișcare: idle, moving, falling, jumping, la setarea unui trigger va executa animația trigger-ului respectiv și se va întoarce în starea de mișcare corectă atât timp cât nu a murit sau nu mai este agățat de un perete.

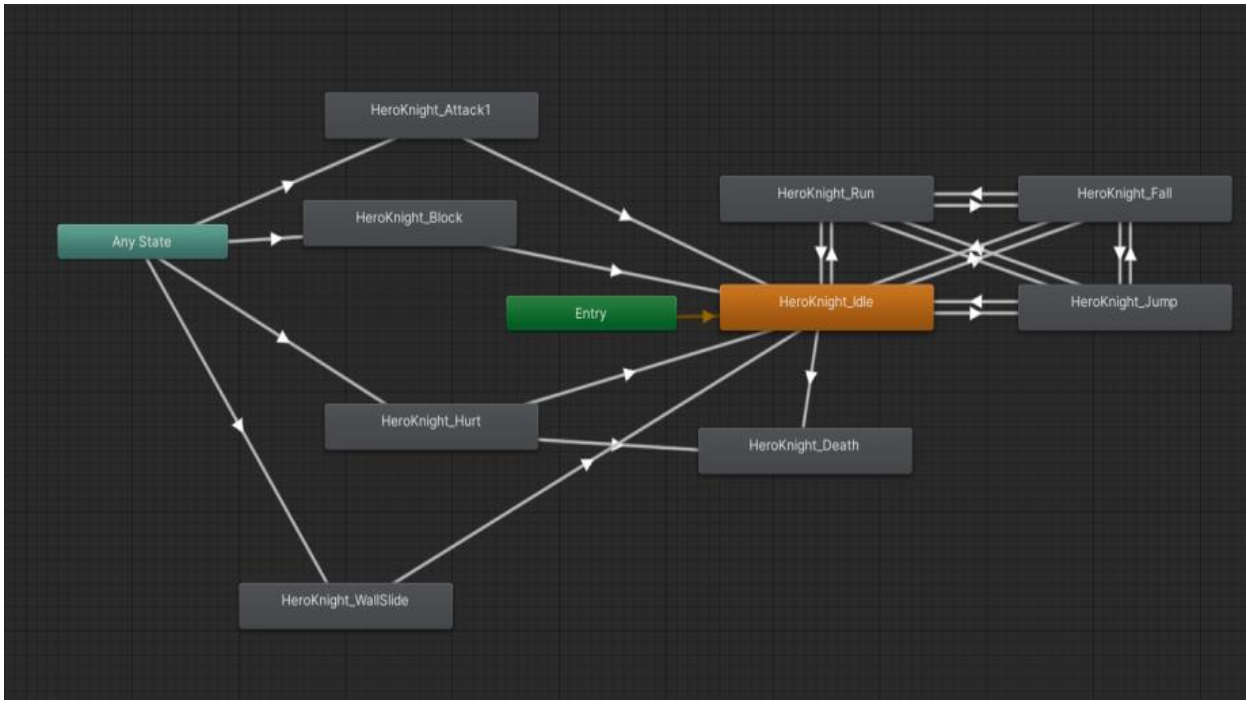

Fig 3.4

La pornirea scenei caracterul se află în starea Idle, la apăsarea butonelor de movement, player-ul se poate mișca fie în stânga fie în dreapta, în funcție de direcția în care merge player-ul va fi orientat în direcția corespunzătoare și starea i se va schimba în Moving. Dacă velocitatea pe axa y este pozitivă atunci indiferent de direcția de mișcare în animator player-ul se va afla în

starea Jumping, și odată ce va avea velocitatea negativă pe axa y, în animator se va afla în starea Falling, menționez că verificarea se face cu pragurile de -0.1 pentru verificare velocității negative si 0.1 pentru cea pozitivă.


Fig 3.5

În continuare voi discuta despre logica săritului, pentru a evita săritul nelimitat la fiecare apăsare de buton trebuie să facem o verificare dacă player-ul se afla în contact cu pământul, pentru a realiza acest lucru îmi trebuie un detector de pământ, pentru a realiza acest lucru voi adăuga un obiect Empty în obiectul player-ului care va avea rol de poziție pentru box cast-ul pe care îl voi face. „Un BoxCast este conceptual similar cu tragerea unei cutii prin scenă într-o anumită direcție. Orice obiect care intră în contact cu cutia poate fi detectat și raportat.” [8] Voi face acest cast la picioarele caracterului jucătorului și voi filtra după layerMask-ul terenului, adică voi lua în calcul doar collider-ele care fac parte din acest layer. Doar dacă detectez o coliziune cu terenul la picioarele jucătorului îi voi permite să sară.

Pentru detectarea coliziuni cu peretele voi face similar cu detectarea podelei, doar că de data aceasta o să verific dacă nu se află în coliziune cu pământul și se află în coliziune cu peretele, dacă acest lucru se întâmplă îi voi da permisiunea să sară de pe perete, acest lucru având denumirea în industria jocurilor de wall-jumping. Box cast-urile vizualizate le puteți vedea în figura 3.5.

Pentru partea de atac voi adăuga un detector similar ca la detectarea coliziuni cu peretele, dar de data aceasta voi folosi metoda OverlapCicleAll din Pysics2D, filtrată pe layer-ul inamicilor, pentru a selecta toți inamici care se află în acest cerc. De ce am ales un cerc și nu o cutie? Deoarece corespunde mai bine cu sprite-ul atacului (figura 3.6). De menționat că pe animația de atac am adăugat un eveniment care va face overlap-ul respectiv și le va lua din viața inamicilor găsiți in zona respectivă.

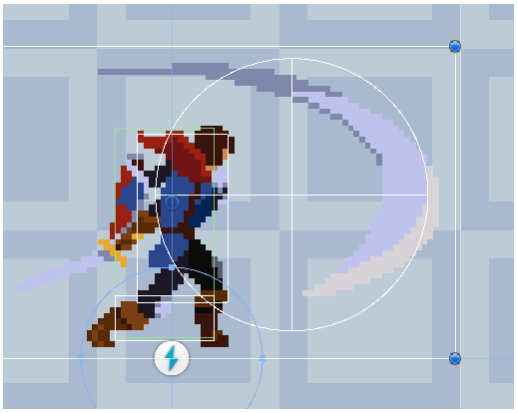
Fig 3.6

Pentru partea de blocare a atacurilor am adaugat în player un boxCollider care va apărea și va dispărea la apăsarea butonului mapat de utilizator. La

atacul unui inamic se va verifica dacă pe axa x acest collider se află între collider-ul inamicului și al player-ului, daca se află și este activ atunci player-ul nu va pierde din viață, iar la coliziunea cu proiectile acestea dispar la coliziunea cu collider-ul scutului. Collider-ul va fi activat si dezactivat de către comportamentul animației, adică la începutul animației acesta va fi activat, iar la sfârșit va fi dezactivat.

Pentru sistemul de viață am creat o componentă care va memora viața maximă și viața curentă, componentă ce poate fi adăugată la orice inamic din scenă, și va avea metodele takeDamage care va scădea din viața curentă a entități și va verifica dacă aceasta a ajuns 0, dacă da va activa animația de moarte și va dezactiva obiectul. În plus această componentă poate interacționa cu bara de viață care se află în scenă.

Bara de viață este un slider care va avea o componentă creată de mine, în ea vom putea seta valoarea maximă și valoarea curentă al slider-ului, dar voi adăuga și un gradient care își va schimba culoarea în funcție de valoarea pe care o va primi, deoarece gradientul are valori doar între 0 și 1 și slider-ul poate avea valori între 0 si orice număr decidem să punem ca si viață maximă, va trebui să extragem din slider valoarea normalizată, ceea ce este un lucru simplu de realizat deoarece unity ne oferă metoda de a extrage valoarea normalizată din slider.

Pentru sistemul de input voi importa noul sistem de input creat de Unity Technologies care este mult mai extensibil și customizabil decât cel clasic. Pentru a folosi acest sistem în primul rând trebuie importat, pentru a face acest lucru deschidem package manager-ul de la unity și vom căuta “Input System”, după ce l-am importat va trebui să creăm un input action asset, unde vom mapa butoanele de care avem nevoie, în cazul meu am nevoie de trei butoane pentru mișcare și patru butoane pentru restul acțiunilor, toate butoanele vor fi verificate doar dacă sunt apăsate așa că acțiunea lor va fi de tip buton, în cazul butoanelor de mișcare vreau să primesc o valoare negativă dacă player-ul se mișcă la stânga și pozitivă dacă se mișcă la dreapta, așa ca tipul acțiuni va fi de tip valoare, și tipul de control este axis, deoarece mă interesează o singură direcție.

După ce am creat input action asset-ul, vom adăuga o componenta Player Input, pe jucător, vom atribui în câmpul actions asset-ul nostru, vom schimba comportamentul să fie de tip Invoke Unity Events, și pentru fiecare acțiune pe care am creato în asset îi vom atribui o metodă creată de mine pe care să o execute.

În final, după finalizarea caracterului jucătorului, acesta a fost transformat într-un prefab.

Un prefab este un oniect prefabricat care poate fi reutilizat în alte scene.

3.1.3 Inamici Melee

Fig 3.7

Pentru inamicii de tip melee am importat de pe Unity Asset Store [9] modelul de la figura 3.7. Pentru început am creat în scenă un nou obiect 2D de tip pătrat și am schimbat sprite-ul din componenta Sprite Renderer cu sprite-ul LightBandit_0 și am ales sorting layer-ul egal cu Enemies, am mărit scale-ul obiectului pe axele x și y la 2.

Mai departe am adăugat componenta de rigidbody 2D unde am adăugat același material de fizici 2D ca și la player, și un box collider 2D pe care l-am configurat puțin mai mic decât sprite-ul lui, deoarece așa pot evita aparența de levitare a sprite-ului deasupra pământului. Pentru orientarea corectă a obiectului am adăugat aceeași componentă ca și la player care va roti obiectul pe axa Y cu 180 de grade.

Mai departe am adăugat componenta animator (fig 3.8) care are rolul de a anima mișcările inamicului, caracterul se poate afla în următoarele stări: Idle, Running, Attack, Hurt, Death și parametrul de tip int pentru starea de mișcare, 3 de tip trigger pentru attack, hurt și unul de tip bool pentru isDead. Logica este următoarea caracterul se află în una din cele 2 stări de mișcare: idle, moving, la setarea unui trigger va executa animația trigger-ului respectiv și se va întoarce în starea de mișcare corectă atât timp cât nu a murit.

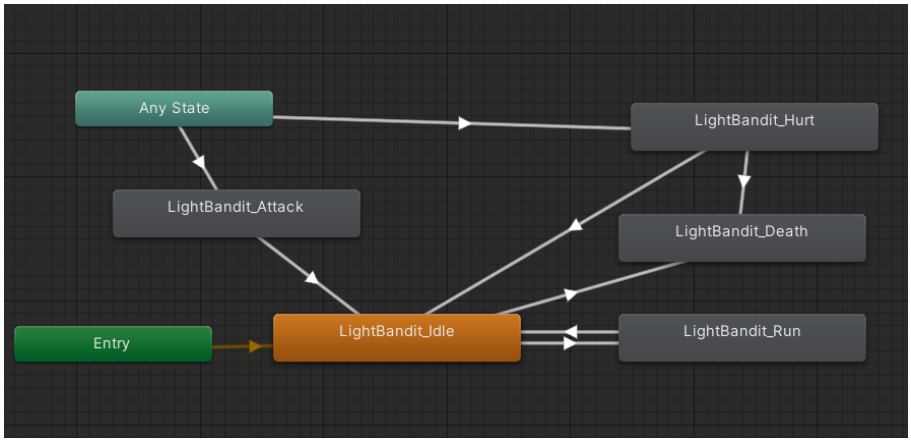

Fig 3.8

Inamicul va avea aceeași componentă de sistem de viață ca și player-ul, cu diferența că nu va avea atribuită o bară de viață.

La începutul scenei, inamicul se va afla în modul de patrulare. Modul de patrularea funcționează în felul următor, voi creea un obiect empty la care voi atașa o componentă ce se va ocupa de comportamentul de patrulă, în acest obiect gol voi adăuga inamicul ca și copil, și două obiecte empty ce vor avea rol de capetele zonei de patrol, un capăt stâng și unul drept. Inamicul va începe să meargă de la un capăt la altul, oprindu-se câteva secunde în starea de idle când ajunge intr-unul din capete (fig 3.9).

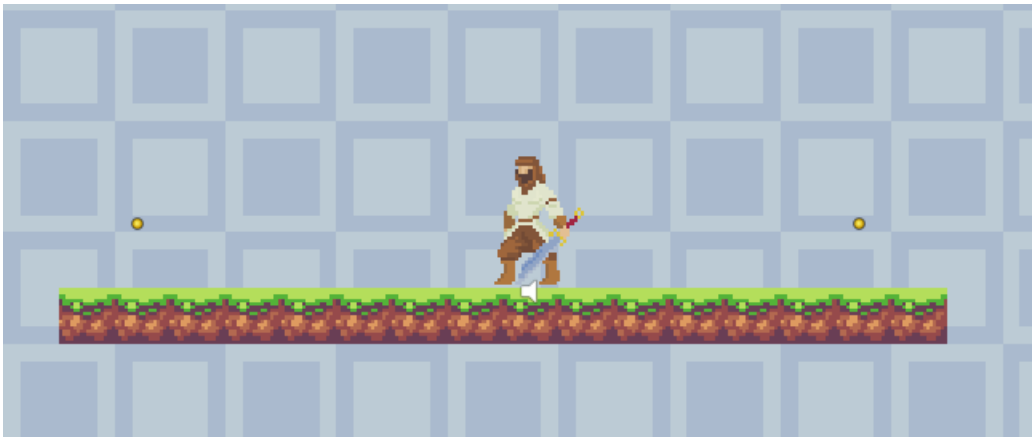

Fig 3.9

Pe lângă comportamentul de patrularea inamicul va verifica printr-un box cast (fig 3.10), filtrat pe layer-ul jucătorului, dacă se află jucătorul în aria lui vizuală, o dată detectat acesta părăsește comportamentul de patrulare și intră în comportamentul de urmărire, menționez că inamicul va urmări câteva secunde jucătorul după ce acesta iese din raza lui vizuală, după terminarea timpului acesta se va întoarce la comportamentul lui de patrulare.

Pentru a evita detectarea jucătorului prin pereți o dată detectat se va face un ray cast în direcția jucătorului pentru a verifica dacă se află pereți între acesta și jucător.

În timp ce inamicul urmărește jucătorul acesta va face un circle cast pentru a verifica dacă se află în raza lui de atac, odată detectat jucătorul în această rază, inamicul va iniția animația de atac, care va avea similar cu jucătorul un trigger la un anumit punct în care va încerca să rănească jucătorul.

Odată inițiat trigger-ul de rănire al jucătorului se va face o verificare dacă jucătorul se află în continuare în raza de atac, și se va verifica poziția scutului dacă acesta este activat, dacă

scutul este activ și se află între jucător și inamic, atunci jucătorul nu va pierde din viață, altfel acesta va pierde.

Similar cu jucătorul, inamicul are un detector sub formă de cerc în fața lui la picioare, cu scopul de a detecta dacă poate merge în față, dacă detectează lipsa pământului în fața lui acesta se va opri din mers.

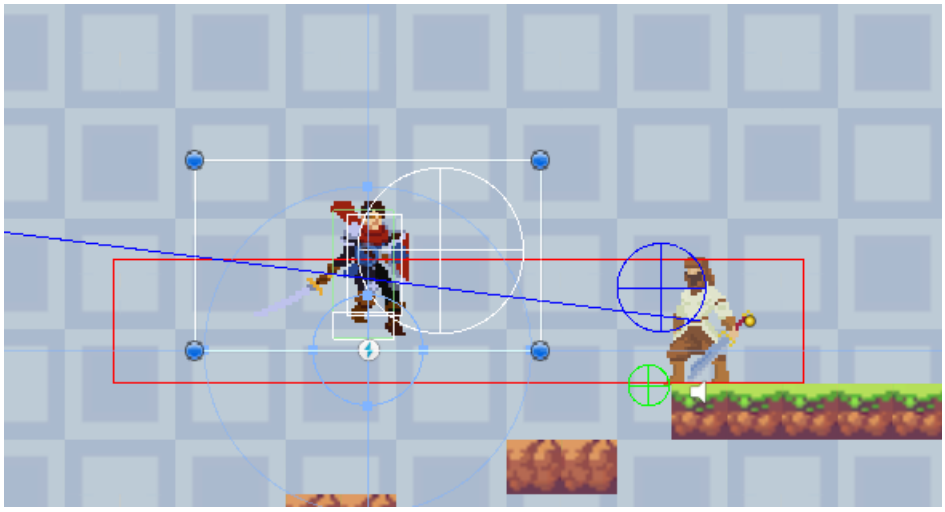

Fig 3.10

Într-un final, după ce am terminat de creat acest tip de inamic, îl voi transforma într-un prefab la fel cum am procedat și la caracterul jucătorului pentru al putea refolosi în scene viitoare.

3.1.4 Inamici zburători

Pentru inamicii de tip zburători am importat de pe Unity Asset Store [10] modelul de la figura 3.11. Pentru început am creat în scenă un nou obiect 2D de tip pătrat și am schimbat sprite-ul din componenta Sprite Renderer cu sprite-ul Idle (46x30)_0 și am ales sorting layer-ul egal cu ContactEnemies, am mărit scale-ul obiectului pe axele x și y la 3.

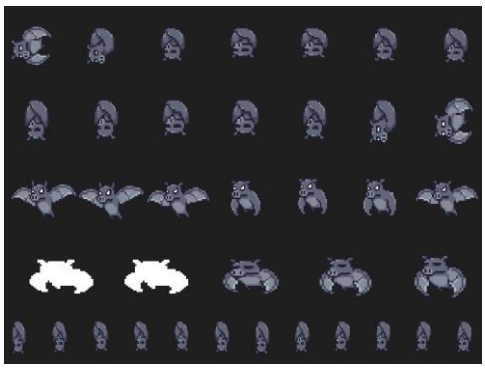

Fig 3.11

Mai departe am adăugat componenta de rigidbody 2D unde am adăugat același material de fizici 2D ca și la player, și un box collider 2D pe care l-am configurat puțin

mai mic decât sprite-ul lui. Pentru orientarea corectă a obiectului am adăugat aceeași componentă ca și la player care va roti obiectul pe axa Y cu 180 de grade și componenta de logică a vieții pe care o au și restul caracterelor..

Mai departe am adăugat componenta animator (fig 3.12) care are rolul de a anima mișcările inamicului, caracterul se poate afla în următoarele stări: Idle, Flying, Ceiling_out, Ceiling_in, Hurt, Death și parametrul de tip int pentru starea de mișcare, unul de tip trigger pentru hurt și doi de tip bool pentru isDead și IsMoving. Logica este următoarea caracterul se află în una din cele 2 stări de mișcare: idle, moving, la schimbarea stări inamicul va trece prin starea de Ceiling_out înainte de a intra în starea Flying și va trece prin starea Ceiling_in pentru a intra în starea de idle, la setarea unui trigger va executa animația trigger-ului respectiv și se va întoarce în starea de mișcare corectă atât timp cât nu a murit. Starea de Flying are un comportament în plus, la intrare în această stare parametrul de tip bool isMoving va fi setat la true, și la ieșirea din această stare este setat înapoi la false, acest comportament este adăugat pe acest inamic deoarece există stările de tranziție între Idle si Flying, și doresc să aplic forțe de mișcare pe inamic doar dacă acesta se află în starea de Flying.

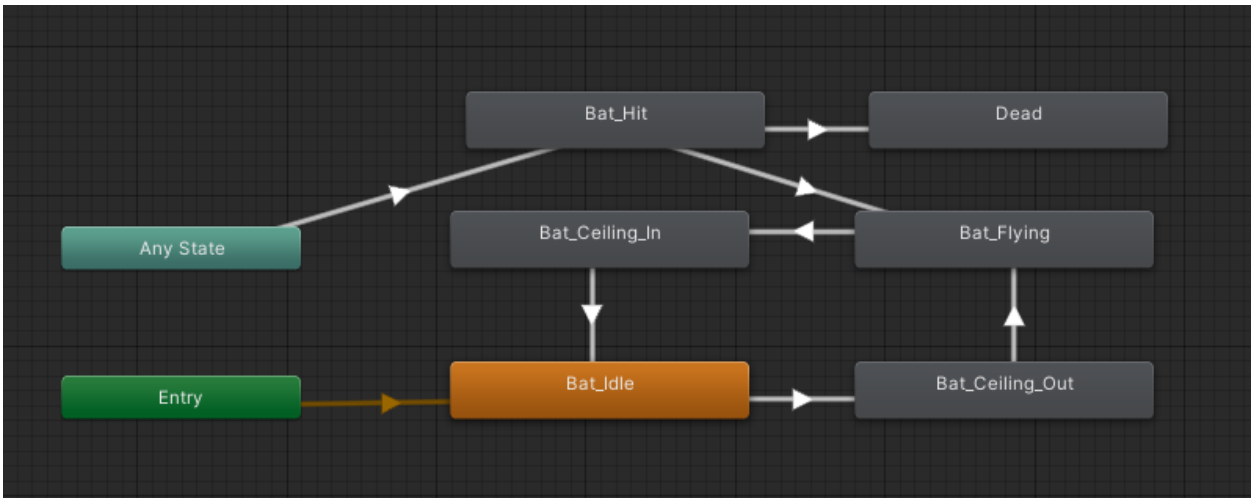

Fig 1.12

Inamicul va avea aceeași componentă de sistem de viață ca și player-ul, cu diferența că nu va avea atribuită o bară de viață. La începutul scenei, inamicul se va afla în modul idle.

Inamicul va verifica printr-un circle cast (fig 3.13), filtrat pe layer-ul jucătorului, dacă se află jucătorul în aria lui vizuală, iar pentru a evita detectarea jucătorului prin pereți o dată

detectat se va face un ray cast în direcția jucătorului pentru a verifica dacă se află pereți între acesta și jucător, o dată detectat acesta părăsește starea de idle și intră în starea de urmărire a jucătorului similară cu cea a inamiculuilor de tip melee, inițial acesta mergea într-o linie dreaptă spre poziția jucătorului și nu evita obstacolele care se aflau pe hartă, pentru a rezolva această problemă am decis să importez proiectul de A* pathfinding dezvoltat de catre Aron Granberg [11].

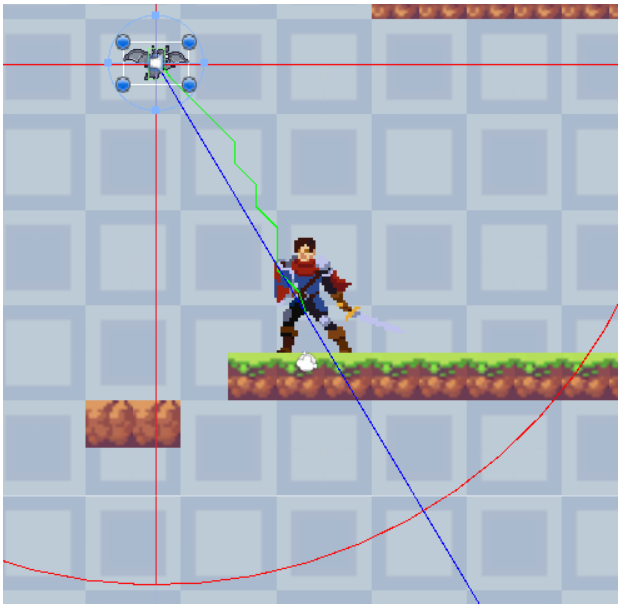

Fig 3.13

Pentru a folosi algoritmul de pathfinding, va trebui adăugat în scenă un obiect empty ce va avea componenta Pathfinder. În acesta am decis să folosesc un graf de tip grid unde voi bifa căsuța 2D în secțiunea grid graph, și voi bifa și căsuța “Use 2D Physics”, iar nodurile vor avea dimensiunea de 0.45, pentru o precizie mai mare la detectarea terenului.

Pentru generarea căii, va trebui să adăugăm componenta seeker pe inamicul nostru iar acum voi putea folosi algoritmul de A* pathfinding în scripturile mele.

Datorită faptului că inamicul nu are o animație de atacat, am decis sa creez componenta de rănire la contact, așa că inamicul va încerca să zboare în caracterul jucătorului. De asemenea inamicul va verifica dacă există un scut între el si hitboxul jucătorului, dacă acest scut este prezent, jucătorul nu va pierde viață. Pentru a îmbogății feedback-ul pe care îl primește jucătorul, la contact cu acest tip de inamic vor fi aplicate forțe de knockback, adică jucătorul va fi aruncat în direcția opusă și în sus față de inamic.

Într-un final, după ce am terminat de creat acest tip de inamic, îl voi transforma într-un prefab la fel cum am procedat și la caracterul jucătorului pentru al putea refolosi în scene viitoare.

3.1.5 Inamici Ranged

Pentru inamicii de tip ranged am importat de pe Unity Asset Store [10] modelul de la

figura 3.14. Pentru început am creat în scenă un nou obiect 2D de tip pătrat și am schimbat sprite-ul din componenta Sprite Renderer cu sprite-ul Idle (44x42)_0 și am ales sorting layer-ul egal cu Enemies, am mărit scale-ul obiectului pe axele x și y la 5.

Fig 3.14

Mai departe am adăugat componenta de box collider 2D pe care l-am configurat puțin mai mic decât sprite-ul lui. Pentru orientarea corectă a obiectului am adăugat aceeași componentă ca și la player care va roti obiectul pe axa Y cu 180 de grade și componenta de logică a vieții pe care o au și restul caracterelor.

Mai departe am adăugat componenta animator (fig 3.15) care are rolul de a anima mișcările inamicului, caracterul se poate afla în următoarele stări: Idle, Attack, Hurt și Dead, animatorul va avea doi parametrii de tip trigger unul pentru atac și unul pentru hurt, și un parametru de tip bool isDead, care va reprezenta dacă inamicul trebuie să se afle în stare de mort sau nu.

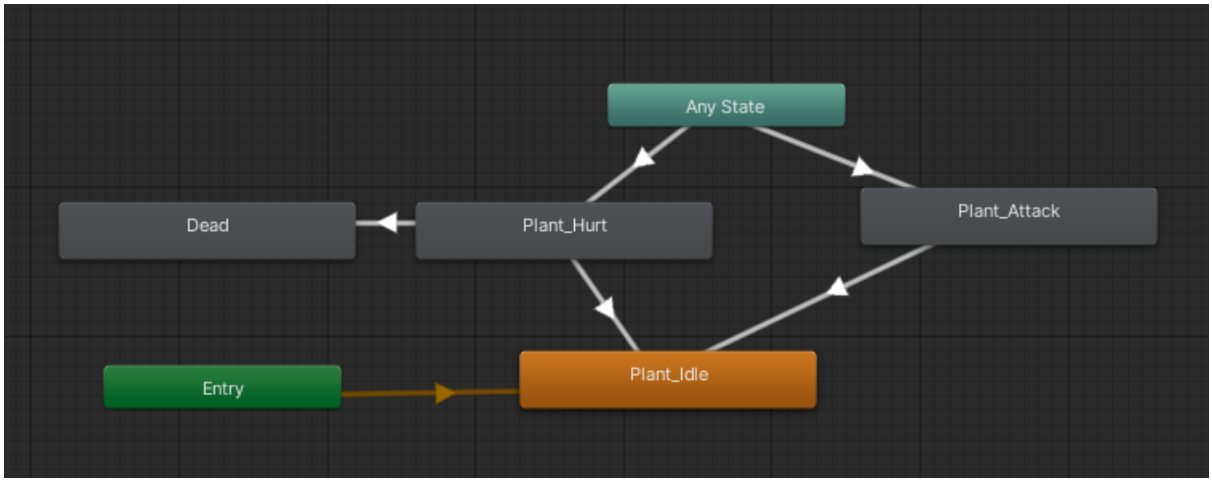

Fig 3.15

Detectarea jucătorului o fac cu un box cast (fig 3,16), destul de mare comparativ cu restul inamicilor, deoarece acesta este un inamic care atacă de la depărtare, consider că trebuie să aibă și zona de detecție mult mai mare. Bineînțeles că și la acesta voi face un raycast pentru detectarea pereților pentru a evita detectatul jucătorului când acesta se află în spatele unui zid.

Inamicului i-am adăugat un obiect empty firePoint, care va reprezenta punctul de unde inamicul va trage cu proiectile atunci când acesta va detecta jucătorul. În momentul când inamicul detectează jucătorul, un obiect de tip proiectil va fi instanțiat.

Pentru ca inamicul să poată trage cu proiectile, are nevoie de proiectile, așa că voi crea un prefab pentru proiectile care va fi instanțiat de fiecare dată când inamicul va trage. Acest proiectil va ignora coliziunile cu inamici, iar la coliziunea cu orice altceva va fi distrus, cu excepția coliziunii cu jucătorul când acesta va fi rănit și va pierde puncte din viață.

Ai-ul inamicului este unul simplu, dacă detectează jucătorul se va uita spre el.

Într-un final, după ce am terminat de creat acest tip de inamic, îl voi transforma într-un prefab la fel cum am procedat și la restul caracterelor pentru al putea refolosi în scene viitoare.

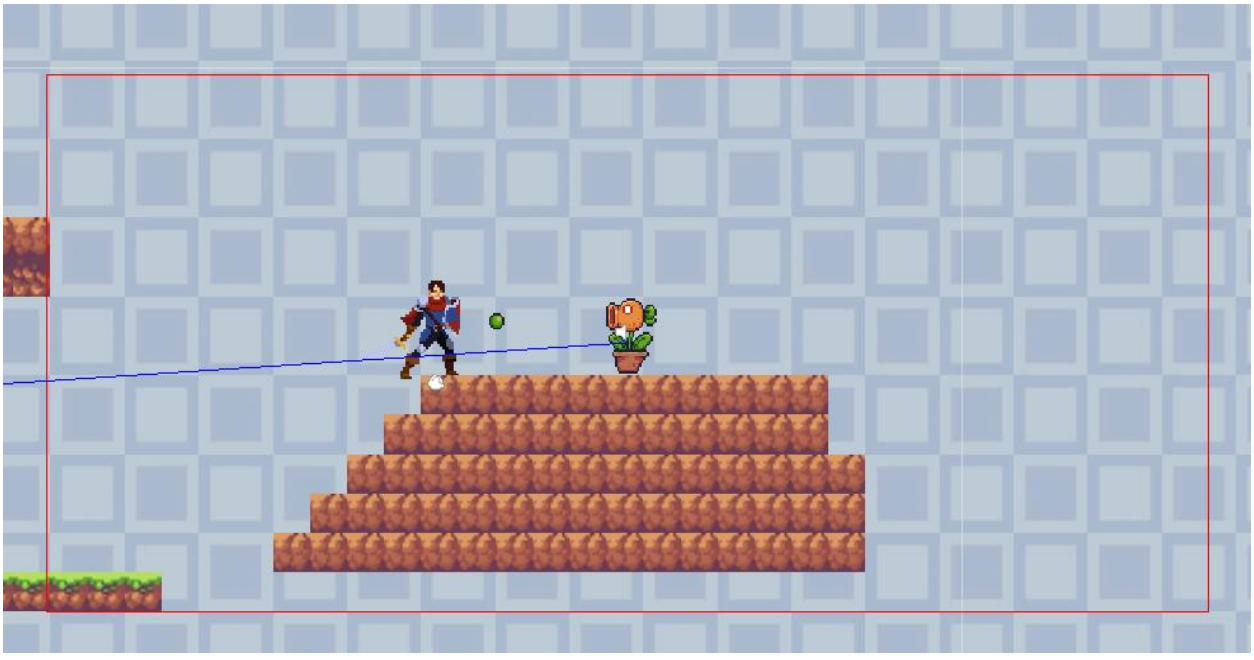

Fig 3.16

3.2 Sistemul de meniuri

Jocul este format din următoarele componente: meniul principal, meniul de selectare al salvărilor, meniul opțiunilor și meniul de pauză. Meniul principal este format dintr-un background, un text de titlu, și 3 butoane, butonul de start, butonul de opțiuni și butonul de ieșire din aplicație. Menționez că sprite-ul butoanelor a fost importat de pe unity asset store [12].

La apăsarea butonului de start va apărea un meniu cu slot-urile de salvare a datelor jocului, un slot poate apărea în una din 2 configurați, dacă slotul respectiv nu are o salvare regăsită pe calculator, atunci va apărea doar un buton de new game pentru a porni un joc nou, altfel dacă salvare este regăsită pe calculator utilizatorul va putea continua jocul de unde a rămas pe salvarea respectivă, sau poate șterge salvarea definitiv.

La apăsarea butonului de opțiuni, utilizatorul va fi întâmpinat cu meniul de opțiuni care este alcătuit din 2 butoane, un buton care va deschide setările sunetelor din aplicație, și un buton care va deschide meniul de remapare a butoanelor.

La apăsarea butonului quit, aplicația se va închide. Menționez că ambele meniuri au un buton back, care apăsat va trimite utilizatorul înapoi la meniul principal.

În timpul jocului, utilizatorul va putea deschide meniul de pauză, apăsând pe tasta ESCAPE, unde va primi optiunile de a continua, de a deschide meniul de opțiuni și de a se reîntoarce la meniul principal.

3.3 Sistemul de sunet

Sistemul de sunet este alcătuit din surse audio, și un output mixer (fig 3.17). Sunetele [13] și muzica [14] au fost importate pe gratis de pe Unity Asset Store.

Pentru început am creat mixer-ul audio și am creat un grup master, unde am adăugat 2 grupuri copii, unul pentru muzică și al doilea pentru efecte sonore.

Pentru muzică am creat un obiect empty la care am adăugat un script pentru a utiliza metoda DontDestroyOnLoad ce are ca efect păstrarea elementului la încarcarea diverselor scene. Dupa la acest obiect am adăugat sursa audio Ambient 2, cu outputul setat la grupul muzicii din

mixer.

Muzica este setată să înceapă la pornirea aplicației și de asemenea este pusă pe modul de loop. Spatial Blend-ul este setat pe 2D pentru a putea auzi muzica de oriunde.

În cazul efectelor de sunet, acestea sunt puse pe obiectele din scenă cum ar fi, avatarul jucătorului, inamicii melee, inamici zburători și inamici ranged.

Sunetele ce pot fi auzite în timpul jocului sunt: sunete de sărit, sunete de lovit cu sabia, sunete de mers, sunete de tras cu proiectile, sunete de lovituri pentru pierdutul punctelor de viață.

Pentru a putea modifica valoarea outputului dintr-un grup audio, în primul rând trebuie să expunem parametrul de volum, pentru realiza acest lucru trebuie să apăsăm pe grupul căruia dorim să-i expunem volumul, click dreapta pe volum în inspector, unde vom putea vedea opțiunea de expunere a parametrului. Voi realiza această expunere pentru ambele grupuri de muzică și efecte sonore.

După ce am expus variabilele, pentru a putea seta volumul, mă voi folosi de sliderele din canvas. În setarea volumului am observat că slider-ul își schimbă valoarea liniar, dar volumul mixerului se modifică pe o scară logaritmică, pentru a rezolva această problemă voi transforma valoarea slider-ului în modul următor: volum mixer = log10(valoare slider) * 20. Menționez că trebuie setat minimul la slidere din 0 în 0.0001, pentru a evita efectul de a ajunge la volumul maxim atunci când slider-ul se află pe valoarea 0.

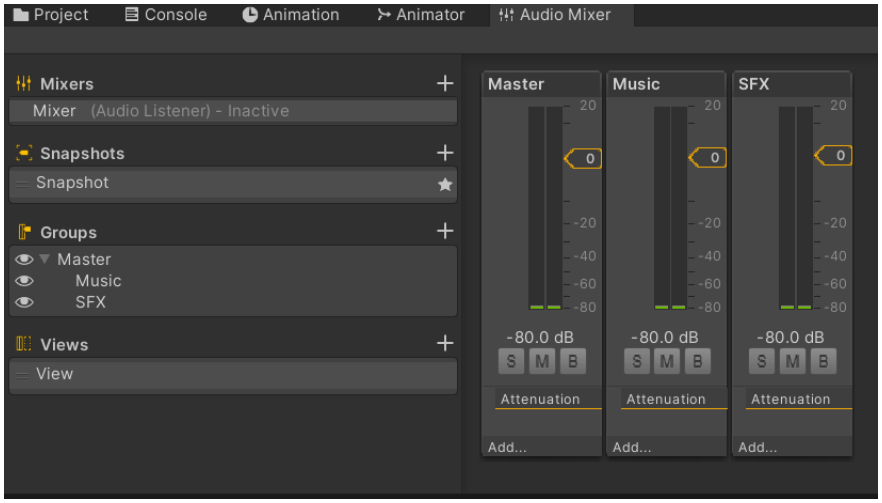

Fig 3.17

3.4 Sistemul de salvare

Este format din componente, folosirea obiectelor de tip scriptableObjects, și partea de formatare binara pentru salvarea locală a datelor pe care doresc să le salvez.

“Un ScriptableObject este un container de date pe care îl poți folosi pentru a salva cantități mari de date, independent de instanțele clasei. Unul dintre principalele cazuri de utilizare pentru ScriptableObjects este de a reduce consumul de memorie al Proiectului tău prin evitarea copiilor de valori.” [8]

“Clasa BinaryFormatter în C# realizează acțiunile de "serializare" și "deserializare" a datelor binare. Aceasta preia structuri de date simple precum numere întregi (int), numere zecimale (float) și colecții de litere și numere (string) și le poate converti într-un format binar. Acest lucru înseamnă că structuri mai complexe, precum o clasă, își pot codifica câmpurile într-un format binar pentru a fi salvate într-un fișier și citite ulterior de un program. În Unity, prin utilizarea acestei clase C#, este posibil să se salveze date, la fel cum s-ar putea face cu PlayerPrefs, dar la un nivel mai complex.” [15]

În aplicația mea, voi folosi formatarea binară pentru salvarea setărilor jucătorului și a salvărilor respective fiecărui slot, în cazul meu voi avea 3 slot-uri de salvări cum am menționat în subcapitolul 3.2. Calea la care vor fi salvate aceste date, este calculată folosind clasa Aplication și accesarea valori de la atributul persistentDataPath, această valoare fiind diferită în funcție de sistemul pe care se află aplicația.

În continuare voi folosi ScriptableObjects, pentru a stoca informați locale între scene în felul următor: pentru a tranziționa de la o scenă la alta am creat un trigger fig. 3.18 de tipul BoxCollider2D care va memora viața curentă și va schimba poziția pe care trebuie să o aibă în scena următoare, adică în momentul când jucătorul intră în contact cu accest trigger, o scenă nouă va fi încărcată, dar la încărcarea scenei se vor pierde datele curente ale jucătorului cum ar fi viața curentă, așa că acestea trebuie salvate undeva local. Mai departe trebuie să menționăm și poziția unde ar trebui să se afle jucătorul în scena următoare la încărcare.

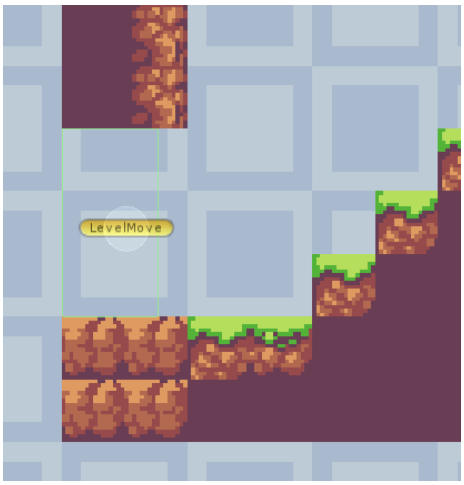

Fig 3.18

Menționez că aceste date nu sunt salvate în calculator.

În continuare voi discuta despre punctele unde jucătorul poate salva, acestea sunt simbolizate sub forma unui steag roșu (fig 2.2). Acest steag roșu este de asemenea un trigger, la coliziunea între jucător și acest trigger, jucătorul va avea opțiunea prin apăsarea tastei de INTERACT, care este mapată în mod inplicit pe tasta E, să salveze jocul salvarea presupunând viața, poziția și scena curentă în care se află, în același timp acest trigger va avea efectul de vindecare completă al jucătorului.

În momentul când jucătorul își va pierde toată viața acesta va reapărea la ultimul punct de salvare cu care a interacționat. Același efect îl va avea și atunci când utilizatorul va apăsa pe butonul de încarcare (“Load”), respectiv unuia dintre cele 3 slot-uri.

Capitolul 3

Concluzii

În concluzie, proiectul meu, The Knight, de dezvoltarea a unui joc în Unity a oferit o experiență valoroasă în ceea ce privește designul și implementarea unui joc captivant, obiectivul meu principal fiind crearea unui joc care să poată oferi o experiență distractivă și în același timp să ofere o provocare utilizatorului.

Consider că jocul și-a atins scopul de a ajunge într-o formă similară cu jocurile 2D, side view, action, adventure, platformer, level based, similar cu jocurile precum *Hollow Knight* și *Ori and the Will of the Wisp*. Apropiindu-se cât mai mult de subgenul metroidvenia, datorită faptului că jocul prezintă o hartă non-liniară, jucătorul având posibilitatea de explorare a lumi în modul lui specific, prin urmare oferind o experiență unică.

Prin utilizarea motorului Unity, am beneficiat de instrumente și funcționalități puternice, permițându-mi să dezvolt într-un mod eficient ajungând la un produs vizual atrăgător, cu mecanici de joc interesante și o interfață intuitivă pentru utilizator.

Proiectul a implicat dezvoltarea unui sistem de meniuri, un sistem de mișcare pentru personajul jucătorului, o inteligență artificială simplă pentru inamici, și un sistem de salvare al progresului jucătorului.

De-a lungul proiectului, am întâlnit provocări și am depășit obstacole care mi-au permis dezvoltarea abilităților în programare și design pentru un joc, de asemenea am lucrat într-un mod optim în așa fel încât să obțin o performanță cât mai bună pentru utilizator.

Relevanța temei este evidențiată de impactul industriei jocurilor în lumea divertismentului, având venituri semnificative și o varietate largă de genuri și stiluri de jocuri, jocurile video având capacitatea de a oferi experiențe virtuale variate.

În același timp jocurile pot ajuta și în știință, prin oferirea posibilități simulări diferitelor lucruri, cum ar fi comportamentele corpurilor cerești, dar și dintr-un punct de vedere filosofic așa cum a reușit să evidențieze prin influențarea teoriei simulării, care afirmă că lumea

înconjurătoare este defapt simulată.

Reperele istorice menționate, cum ar fi primul joc video creat de William Higinbotham în 1958, apariția subgenului metroidvania bazat pe jocurile *Metroid* si *Castlevania,* sau apariția jocurilor de succes precum *Grand Theft Auto, Dark Souls* și *Hollow Knight*, demonstrează evoluția și influența jocurilor video în industrie.

Într-un final, proiectul meu, The Knight, mi-a oferit oportunitatea de a explora tehnologiile oferite de unity în dezvoltarea jocurilor video, dar și oportunitatea de a păstra o gândire creativă pe parcursul dezvoltări proiectului. Sunt mândru de rezultatele obținute și sper ca jocul meu să ofere o experiență cât mai captivantă și memorabilă pentru jucători.

Bibliografie

[1] Bary Elad, *Film Industry Statistics – By The Distributor, Running Time, Demographic, Box Office Revenue*, https://www.enterpriseappstoday.com/stats/film-industry-statistics.html (ultima accesare: 07.06.2023)

[2] Ifpi, *IFPI Global Music Report: Global Recorded Music Revenues Grew 9% In 2022*, https://www.ifpi.org/ifpi-global-music-report-global-recorded-music-revenues-grew-9-in-2022/ (ultima accesare: 07.06.2023)

[3] Tom Wijman, *The Games Market in 2022: The Year in Numbers*, https://newzoo.com/resources/blog/the-games-market-in-2022-the-year-in-numbers (ultima accesare: 07.06.2023)

[4] Brianna Reeves, *Team Cherry Reveals Hollow Knight Sales Have Topped 2.8 Million*, https://www.playstationlifestyle.net/2019/02/14/hollow-knight-sales-top-two-million/ (ultima accesare: 07.06.2023)

[5] Ernie Tretkoff, *October 1958: Physicist Invents First Video Game*, https://www.aps.org/publications/apsnews/200810/physicshistory.cfm, (ultima accesare: 08.06.2023)

[6] Ansimuz, *Sunny Land*, https://assetstore.unity.com/packages/2d/characters/sunny-land-103349, (ultima accesare: 08.06.2023)

[7] Sven Thole, Hero Knight – Pixel Art, https://assetstore.unity.com/packages/2d/characters/hero-knight-pixel-art-165188, (ultima accesare: 08.06.2023)

[8] Unity Documentation, https://docs.unity3d.com/ScriptReference/, (ultima accesare: 08.06.2023)

[9] Sven Thole, *Bandit – Pixel Art*, https://assetstore.unity.com/packages/2d/characters/bandits-pixel-art-104130, (ultima accesare: 08.06.2023)

[10] Pixel Frog, *Pixel Adventure 2*, https://assetstore.unity.com/packages/2d/characters/pixel-adventure-2-155418, (ultima accesare: 09.06.2023)

[11] Aron Granberg, *A* Pathfinding Project*, https://arongranberg.com/astar/download, (ultima accesare: 09.06.2023)

[12] KartInnka, *Buttons Set*, https://assetstore.unity.com/packages/2d/gui/buttons-set-211824, (ultima accesare: 09.06.2023)

[13] leohpaz, *RPG Essentials Sound Effects - FREE!*, https://assetstore.unity.com/packages/audio/sound-fx/rpg-essentials-sound-effects-free-227708, (ultima accesare: 09.06.2023)

[14] alkakrab, *25 Fantasy RPG Game Tracks Music Pack*, https://assetstore.unity.com/packages/audio/music/25-fantasy-rpg-game-tracks-music-pack-240154, (ultima accesare: 09.06.2023)

[15] Dan Cox, *Using BinaryFormatter in Unity to Save and Load Game Data*, https://videlais.com/2021/02/24/using-binaryformatter-in-unity-to-save-and-load-game-data/, (ultima accesare: 09.06.2023)