**ANEXA NR.3**

UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREŞTI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

COORDONATOR ŞTIINŢIFIC:

Conferențiar Mironela Pîrnau

ABSOLVENT:

Popescu George Sorin

SESIUNEA IUNIE/IULIE

2023

UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREŞTI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Implementarea unui joc video 2D în Unity

COORDONATOR ŞTIINŢIFIC:

Conferențiar Mironela Pîrnau

ABSOLVENT:

Popescu George Sorin

SESIUNEA IUNIE/IULIE

2023

Cuprins

[Introducere 5](#_Toc132209984)

[Capitolul 1 – Tehnologia Unity pentru jocurile 2D 9](#_Toc132209985)

[1.1 Cele mai populare jocuri dezvoltate cu Unity 12](#_Toc132209986)

[1.2 Tehnologii alternative pentru dezvoltarea jocurilor 2D 14](#_Toc132209987)

[Capitolul 2 – Descrierea jocului dezvoltat 16](#_Toc132209988)

[2.1 Sursa de inspirație a jocului dezvoltat - The Binding of Isaac 16](#_Toc132209989)

[2.1.1 Un scurt istoric 16](#_Toc132209990)

[2.1.2 Descriere 17](#_Toc132209991)

[2.2 Descrierea jocului implementat 19](#_Toc132209992)

[Capitolul 3 – Descrierea funcționalităților jocului dezvoltat 20](#_Toc132209993)

[3.1 Meniul principal al jocului 20](#_Toc132209994)

[3.2 Meniul pentru setări 22](#_Toc132209995)

[3.3 Metode si funcționalități pentru caracterul jucătorului 26](#_Toc132209996)

[3.4 Metode si funcționalități pentru camerele labirintului 29](#_Toc132209997)

[3.4 Testarea aplicației și rezolvarea bug-urilor 39](#_Toc132209998)

[Bibliografie 41](#_Toc132209999)

Idee tabel comparatie tehnologii alternative

# Introducere

Odată cu invenția primului calculator in anul 1943, acestea au fost folosite în mare parte pentru rezolvarea calculelor care erau prea complicate sau prea costisitoare din punct de vedere al timpului. A fost nevoie de doar 4 ani ca cineva sa creeze un program de calculator pentru scopuri recreative, acesta fiind considerat primul joc interactiv pe calculator si a fost crea în anul 1947 de către Thomas T. Goldsmith Jr. și Estle Ray Mann și simula lansarea unei rachete. Acest program a fost considerat mult prea costisitor si doar o simplă curiozitate in cadrul lumii calculatoarelor pentru a fi dezvoltat mai departe.

În prezent jocurile pe calculator reprezintă o industrie masivă, valorând peste 300 de miliarde de dolari și sunt alese de milioane de persoane ca o modalitate de petrecere a timpului liber (Fig.1). Această industrie se află într-o continuă evoluție atât tehnologică cat si din punct de vedere al profiturilor anuale datorită faptului că este forțată sa inoveze din ce in ce mai rapid pentru a satisface cererile consumatorilor.

Fig. 1Numărul oamenilor care joacă jocuri video activ în întreaga lume

Industria jocurilor video este împărțită in funcție de platforma pentru care sunt produse jocurile, principalele platforme sunt calculatoarele, consolele si Telefoanele. Deși în ultimii ani industria jocurilor pe mobil a crescut foarte rapid datorita răspândirii telefoanelor mobile, jocurile pe calculator încă reprezintă un procent impresionant din totalul jocurilor de pe piață (Fig.2).

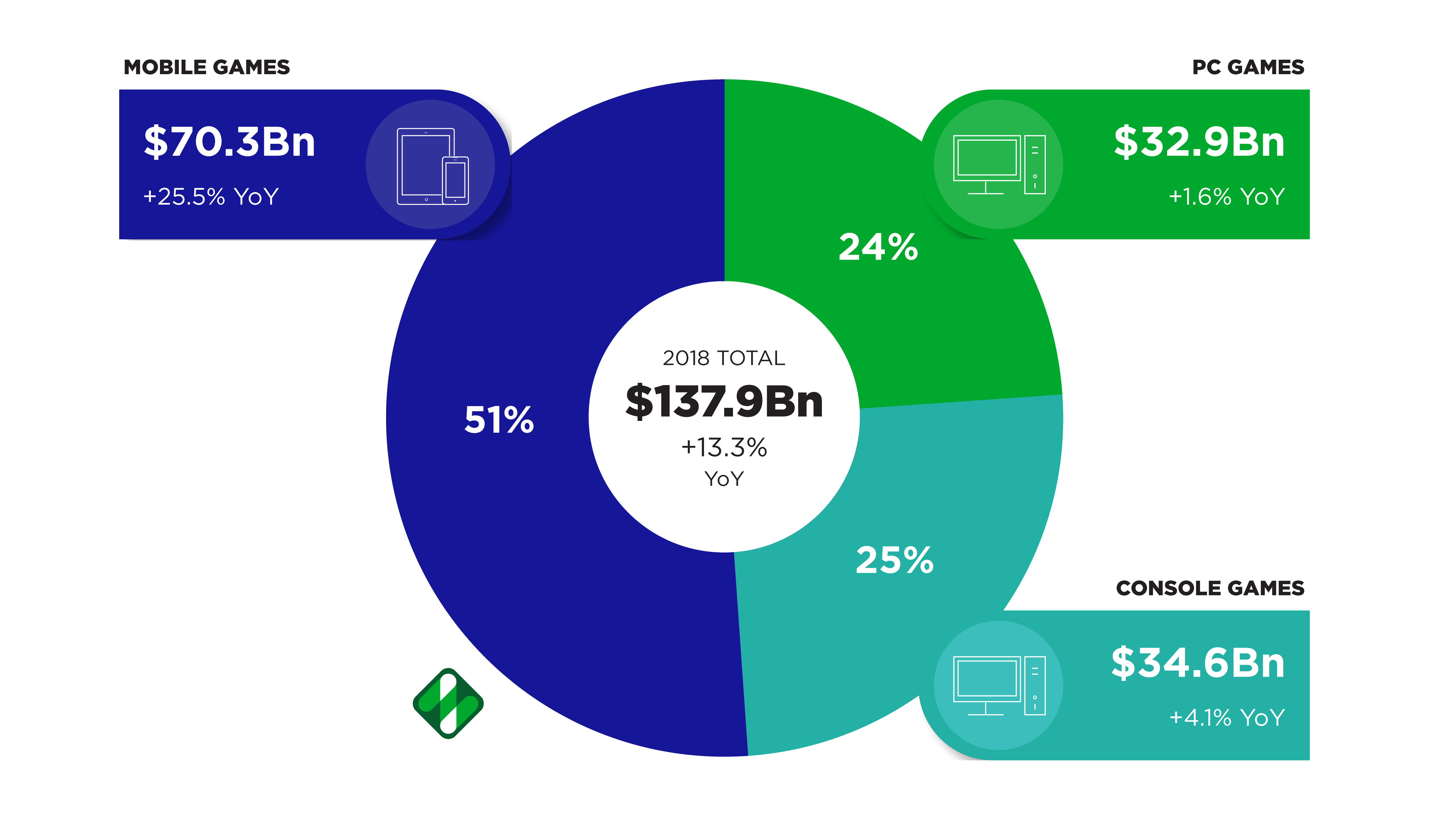


Fig. 2 Marketul jocurilor pe calculator [1]

Deși jocurile de pe mobil sunt clar mai răspândite si mai profitabile decât cele de pe computer acestea se confruntă cu o foarte mare problemă, limitările hardware ale platformei. Computerele personale nu se confruntă cu această problemă deoarece lasă utilizatorii să își modifice sistemele cu diferitele componente disponibile pe piață, dacă jocul pe care dorim sa îl jucăm are specificații prea mari pentru computerul nostru putem cumpăra o placa video sau un procesor care sa satisfacă cererile acelui joc.

Influențat de aspectele prezentate mai sus și de propria pasiune pentru acest subiect, îmi propun prin intermediul acestei lucrări implementarea unui joc video cu ajutorul motorului de jocUnity. Aplicația este inspirată dintr-o serie foarte populară de jocuri, ea se incadrează in categoria roguelike dungeon crawler. Această categorie de jocuri video este formată prin combinarea a două categorii, prima fiind cea de “roguelike” jocurile din această categorie sunt recunoscute dupa următoarele caracteristici [[2]](#_Bibliografie) :

* Progresul jocului nu este salvat de la o sesiune la alta, odată ce jucatorul a închis jocul sau a pierdut, va fi nevoit să o ia de la capat.
* Anumite aspecte, poate chiar intregul joc, va fi generat procedural, acest lucru oferă jocului o valoare de rejucabilitate foarte ridicata, experienta fiind vast diferita de la un jucator la altul.
* Toată acțiunea jocului se intamplă pe un singur ecran, nu exista extrane speciale pentru bătalii, scene de filmare sau orice altceva.

iar a doua fiind cea de “dungeon crawler” care se clasifică prin [3] :

* Explorarea unui labirint generat de către joc.
* Evitarea anumitor inamici și capcane cu scopul de a ajunge la finalul labirintului.
* Obținerea de obiecte care vor face explorarea mai ușoară

În cadrul acestui joc utilizatorul va explora diferite camere în cadrul unui labirint, acestea vor fi populate de inamici care trebuiesc învinși pentru a putea înainta, jocul se va încheia fie la pierderea tuturor punctelor de viața are jucătorului sau prin ajungerea într-o anumită cameră unde jucătorul poate încheia sesiunea de joc.

În Capitolul 1 este prezentat pe scurt motorului de joc Unity, popularitatea acestuia, cele mai populare jocuri realizate și câteva programe alternative pentru dezvoltarea jocurilor.

Capitolul 2 descrie pe scurt în ce va consta jocul dezvoltat în cadrul acestei lucrări, principala sursă de inspirație pentru acesta și mecanicele din cadrul jocului.

Capitolul 3 descrie în detaliu funcționalitățile jocului împreuna cu metodele și uneltele folosite pentru implementarea acestora.

Capitolul 4 descrie bug-urile descoperite în procesul de testare si cum acestea au fost rezolvate.

# Capitolul 1 – Tehnologia Unity pentru jocurile 2D

Unit[[1]](#footnote-1) este un motor de joc cross-platform creat și dezvoltat de Unity Technologies, a fost prima dată anunțat și lansat in Iunie 2005 în cadrul conferinței “Apple Worldwide Developer Conference” ca un motor de joc pentru sistemul de operare Mac OS X, dar la puțin timp a fost adaugat și suport pentru Microsoft Windows si platforme web. În de-a lungul aniilor motorului de joc i-au fost adaugate o varietate de unelte si modificări pentru a facilita dezvoltarea rapidă și ușoară a jocurilor video, fie ele pe calculator, mobil, VR sau chiar pe internet.

Popularitatea acestui motor de joc se datorează mai multor factori, cel mai important dintre aceștia fiind suportul oferit pentru dezvoltarea pe o multitudine de platforme, dezvoltatorii având o fexibilitate mai mare cand vine vorba de jocul făcut. Un alt factor foarte important este cât de ușor de învațat și folosit este de fapt. Unity ne oferă o interfață de utilizator(Fig.3) foarte ușor de înțeles și intuitivă, aceasta fiind de obicei compusă din 5 parți foarte importante:

1. Vederea scenei – Aici putem vedeam și modifica scena curentă a jocului.Putem adauga sau șterge diferite obiecte, putem face legaturi intre obiecte etc. În acest loc se vor face cel mai des modificari proiectului
2. Vederea jocului – Aici se vede o simulare a jocului în timp real odată cu rularea acestuia. În aceasta vedere nu putem modifica decat rezolutia ecranului simulat si scala imaginii.
3. Ierarhia obiectelor din scenă – Aici se vor vedea toate obiectele din scenă, sub forma ierarhică (parinți-copii)
4. Ierarhia proiectului – Aici se vede întreaga ierarhie a proiectului, de cele mai multe ori vor fi foldere speciale pentru fiecare parte a dezvoltarii jocului (pentru scripturi, asset-uri,scene etc)
5. Inspectorul de elemente – Aici se vor vedea proprietațiile obiectului selectat, cum ar fii poziția acestuia în scenă, scripturile și componentele atașate acestuia, texturile folosite etc. Graphical user interface

   Description automatically generated

Fig. 3 Interfața de utilizator Unity

Pe lângă aceste 5 componente principale Unity ne pune la dispoziție o întreagă gamă de unelte, setări și funcții, acestea fiind ascunse in meniurile din susul aplicației. Dintre acestea, unul dintre cele mai importante si utile este *Unity Asset Store [[2]](#footnote-2)*, acest meniu ne duce la site-ul oficial Unity. Din acest loc ne putem alege o multitudine de lucruri care ne vor ajuta în dezvoltarea jocului nostru, precum texturi 2D sau 3D, efecte audio, muzica etc. dezvoltate în întregime de către comunitate. Multe dintre acestea sunt desigur contra cost, dar exista și variante complet gratuite.

În cadrul magazinului online vom putea găsii și unelte specializate pentru diferite aspecte ale jocurilor video, cum ar fi ( unelte pentru facilitarea creări de animații, texturi și fizici pentru materialul textil, unelte special dedicate funcțiilor de securitate etc). Dacă aceste unelte sunt deosebit de populare sau îmbunătățesc funcții de baza acestea pot fi dobândite de Unity și integrate chiar in motorul de joc, acest lucru s-a întâmplat cu TextMesh Pro, o unealtă dedicata folosirii si modificării de text in cadrul jocurilor.

Fiecare resursă din cadrul Asset Store-ului este verificată si moderată de echipa Unity, acest lucru garantează un nivel de calitate dar și faptul ca tot ce vom găsii va fi complet compatibil cu motorul de joc.

A picture containing text, monitor, screenshot, black

Description automatically generated

Fig. 4 Pagina principala Unity Asset Store

## 1.1 Cele mai populare jocuri dezvoltate cu Unity

Cu titluri în număr de zeci de milioane jocurile sunt cele mai populare aplicații pe calulator. În continuare vom discuta si analiza pe scurt unele din cele mai populare jocuri dezvoltate cu Unity.

**Cuphead**

Este unul dintre cele mai populare jocuri video 2D realizare in Unity,el a fost lansat în anul 2017 cu o receptie extrem de pozitivă, vânzând peste un milion de copiii doar in prima zi. Jocul a cucerit prin farmecul său, acesta reușind sa recreeze stilul desenelor animate ale aniilor 30 și prin dificultatea nivelelor, oferind jucătoriilor o adevărată provocare care îl va face să stea pe marginea scaunului de fiecare dată cand se va juca.

Gameplay-ul jocului constă în numeroase boss fight-uri, acestea fiind cel mai impresionant aspect al jocului. Fiecare șef are o estetică complet aparte datorită modului în care acestia sunt desenați și reprezentați, ei lunând foarte multă inspiratie din desenele Disney. Între fiecare boss fight se afla unul sau mai multe nivele de tip run-and-gun (tragi și fugi) unde jucatorul trebuie să se ferească de numeroase proiectile inamice în timp ce incearca sa distrugă inamici sau obstacole din calea sa.[[4]](#_Bibliografie)

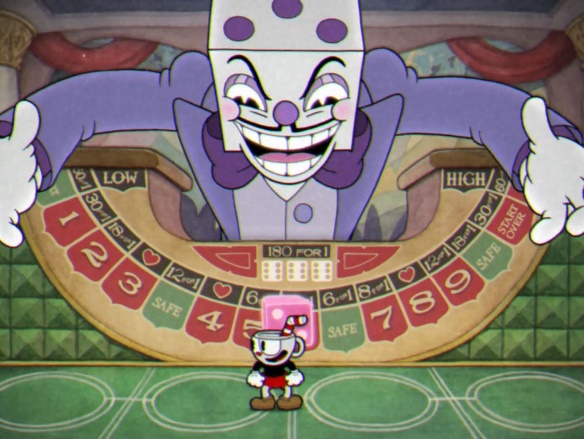


Fig. 4 Șeful unui nivel cazino[[3]](#footnote-3) Fig. 5 Nivel run-and-gun[[4]](#footnote-4)

**Among Us**

Este un joc video 2D dezvoltat și publicat de studioul independent Innersloth, jocul a devenit foarte rapid un clasic de cult, fiind iubit pentru aspectul său social si pentru simplitatea sa. Jocul este inspirat de către jocul de societate Mafia[5] si de filmul The Thing[6] aceste 2 inspiratii se reflectă desigur și in cadrul gameplay-ului unde jucatorii sunt blocați pe o naveta spatiala unde sunt nevoiți sa completeze diverse activitați in tip ce impostorii încearca sa îi saboteze și să le reducă numărul de jucători.

Principalul nivel de dificultate vine din aspectul social al jocului, unde jucatorii trebuie sa deducă care dintre aceștia este defapt impostorul, acest lucru se face cu ajutorul chat-ului din cadrul jocului sau prin intermediul diferitor modificari aduse jocului de către comunitatea sa, cum ar fi o funcție de chat vocal regional. Aici jucatorii trebuie sa joace rolul de detectiv în timp de impostorii îi vor minții, însela și acuza pentru ca ei sa scape de vina si suspiciune.

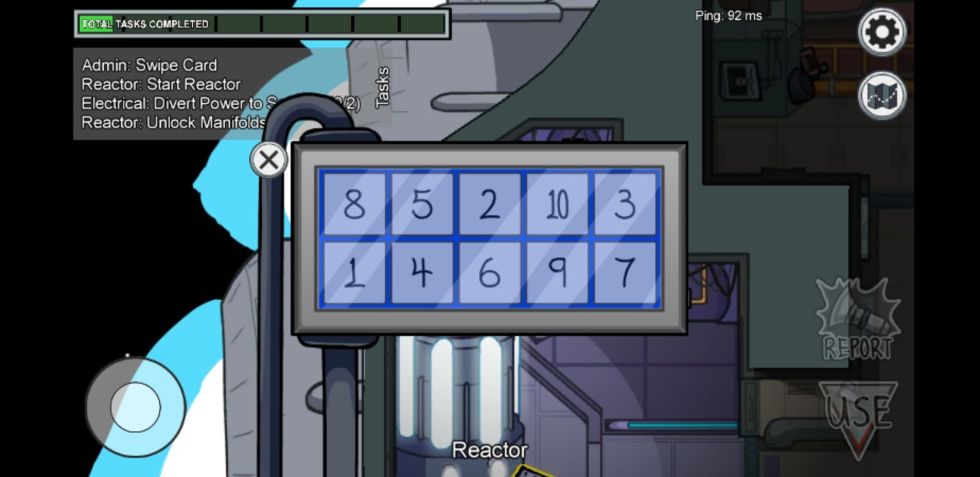


Fig. 6 Un jucător făcând una din diferitele activități din cadrul navei [[5]](#footnote-5)



Fig. 7 Toți jucătorii din cadrul mediului votând cine cred ca este impostorul [[6]](#footnote-6)

Din exemplele prezentate mai sus putem observa că industria jocurilor video este foarte variată. De cele mai multe ori jocurile țintesc un anumit public care are anumite așteptări, fie își dorește un joc care ii va provoca din punct de vedere al dificultății sau un joc care îi va face sa gândească ca un adevărat detectiv, principala componenta a acestor jocuri este distracția.

## 1.2 Tehnologii alternative pentru dezvoltarea jocurilor 2D

Deși Unity este cel mai popular motor de joc asta nu înseamnă ca este singurul sau cel mai bun. Fiecare alternativă, chiar si Unity, prezintă diferita dificultăți dar și oportunități pentru dezvoltarea jocurilor. În continuare vom discuta pe scurt avantajele si dezavantajele unor diferite tehnologii.

**Unreal Engine**

Unreal Engine[[7]](#footnote-7) considerat de dese ori cea mai buna alternativa pentru dezvoltarea jocurilor, împărtășește foarte multe din calitățile sale cu Unity, cum ar fii magazinul online pentru resurse, tutoriale realizate chiar de echipa Unreal Engine, abilitatea de a dezvolta atât jocuri 2D cât si 3D etc.

Spre deosebire de multe motoare de joc, Unreal Engine se axează foarte mult pe aspectele grafice ale unui joc, acesta putând sa producă efecte vizuale, texturi, animații și grafici mult mai detaliate. Bine înțeles că odată cu toate acestea ne este oferit si un repertoar mult mai mare de unelte care se ocupa cu îmbunătățirea si modificarea acestor aspecte împreuna cu unelte care ne ajuta sa optimizam jocurile pentru a folosii aceste texturi mai impresionante.

Cele mai mari dezavantaje ale acestui motor de joc constau în dificultatea sa de folosire, el ne fiind recomandat pentru proiecte mici, proiecte dezvoltate de către o singură persoana și este nevoie o selectare foarte atenta a unei echipe daca aceasta este necesară. [7]

**Buildbox**

Buildbox[[8]](#footnote-8) este un motor de joc cu totul special, acesta nu necesita deloc scrierea de cod pentru a realiza jocuri video, în schimb oferă dezvoltatorilor o mulțime de unelte prestabilite care îndeplinesc anumite cerințe (adăugarea de coliziuni automată anumitor obiecte, crearea și distrugerea obiectelor etc).

Dezvoltarea de jocuri video fără a fi nevoit sa înveți un nou limbaj de programare este cel mai mare avantaj deoarece dificultatea de implementare si dezvoltare scade considerabil, transformând tot procesul de dezvoltare într-o chestiune de câteva ore. Dar acest aspect este și cel mai mare dezavantaj al motorului de joc, deoarece nu avem control asupra multor aspecte. În momentul actual dezvoltarea jocurilor multiplayer este imposibila in cadrul acestui program, dar poate fi adăugată pe parcursul dezvoltării acestuia.

# Capitolul 2 – Descrierea jocului dezvoltat

În cadrul acestei lucrări am ca obiectiv dezvoltarea unui joc video 2D inspirat din seria jocurilor „The Binding of Isaac” unde jucătorul controlează un caracter cu scopul de a explora un labirint generat complet procedural și de a ajunge la capătul acestuia. Pe parcurs jucătorul poate obține diferite obiecte care îi vor îmbunătății abilitățile de baza cum ar fi daunele provocare, viteza de mișcare sau de atac, raza de atac etc. Toate aceste aspecte vor fi prezentate în detaliu în următorul capitol, unde voi prezenta cum au fost implementate în cadrul jocului.

## 2.1 Sursa de inspirație a jocului dezvoltat - The Binding of Isaac

Printre numeroasele jocuri pe calculator, regăsim și un joc numit „The Binding of Isaac” care nu numai a readus viață unui întreg stil de jocuri video dar este si considerat ca fiind reprezentantul acelui întreg stil, rogue-like.

### 2.1.1 Un scurt istoric

The Binding of Isaac este o serie de jocuri care își are începuturile in era jocurilor pe browser dezvoltate in Adobe Flash. El a fost creat în anul 2011 de către Edmund McMillen și Florian Himsl după o competiție între aceștia pentru a vedea cine poate sa dezvolte un joc video mai bun care se încadrează in tipul jocurilor rougue-like. Jocul abordează subiecte considerate foarte sensibile cum ar fi opiniile atât pozitive cât si negative ale lui Edmund referitoare la religie și efectele pe care aceasta le poate avea în viața unui copil ușor impresionabil.

Jocul nu a fost supus unor așteptări foarte mari, chiar creatorii acestuia se așteptau ca acesta să eșueze, dar au fost surprinși de succesul pe care acesta l-a avut odată ce a fost lansat pe platforma de jocuri Steam. Odată ce s-a observat ca jocul devine din ce în ce mai popular creatorii acestuia s-au decis sa lanseze o expansiune numită „Wrath of the Lamb” dar și-au dat seama ca nu mai pot dezvolta jocul după aceasta, datorita limitărilor platformei Flash. [8]

În ziua de azi jocul a fost dezvoltat de la 0 într-un motor de joc creat de către dezvoltatorii acestuia și se bucură de o popularitate deosebita in cadrul fanilor de jocuri rogue-like. Acesta este încă dezvoltat și îmbunătățit, având o expansiune nouă lansată în anul 2021 fiind primita cu brațele deschise de către fanii seriei.

### 2.1.2 Descriere

The Binding of Isaac este un joc top-down dungeon crawler, prezentat folosind modele bidimensionale, în care jucătorul îl controlează pe Isaac sau alte caractere care pot fi deblocate pe parcurs în timp ce explorează diferite nivele populate de monștri. Aceste nivele sunt generate procedural, făcând ca experiența de la o sesiune de joc la alta sa fie complet diferită(Fig.8).



Fig. 8 Harta labirintului generat aleatoriu

Principala buclă de joc constă in explorarea diferitelor camere din cadrul labirintului, daca o camera este însă populată de monștrii, jucătorul trebuie sa îi învingă pe aceștia pentru a putea continua(Fig. 8). Odată ce o camera este curățată de monștrii, aceasta rămâne în continuare goala pana la finalizarea jocului.



Fig. 8 O camera cu ușile închise deoarece este populată de monștri

Pe parcursul explorării jucătorul poate obține diferite obiecte care îl vor ajuta, acestea pot să îi crească daunele pe care atacurile sale le provoacă, sa îi crească viteza de mișcare, de atac și așa mai departe. Aceste obiecte sunt echipate automat de către caracter si ii vor schimba si aspectul vizual(Fig.9).

A screenshot of a video game

Description automatically generated A screenshot of a video game

Description automatically generated

Fig. 9 Un obiect care va crește punctele de viață, se poate observa și schimbarea vizuala a personajului

Principalul obiectiv al jocului este sa ajungem la finalul labirintului și să înfrângem monstrul șef. Odată înfrânt acesta va genera o cale de ieșire din labirint.



Fig. 10 Confruntarea cu un monstru șef

## 2.2 Descrierea jocului implementat

Obiectivul acestei lucrări este dezvoltarea unui joc video 2D cu mecanici inspirate de către jocul The Binding of Isaac. Jocul va consta în explorarea unui labirint generat procedural, complet unic de la o sesiune la alta. Camerele labirintului vor fi populate de inamici creați la întâmplare, jucătorul nu va putea părașii camera pana când toți inamicii vor fi învinși, aceștia au de asemenea o șansă să lase anumite obiecte după ce au fost învinși. Obiectivul jocului va fi ca jucătorul să ajungă în ultima camera a labirintului de unde va putea finaliza sesiunea de joc.

Toate acestea vor fi realizate cu ajutorul motorului de joc Unity și uneltele puse la dispoziție de către acesta. Toate texturile folosite au fost luate de pe Unity Asset Store sau făcute si/sau modificate de către mine.

# Capitolul 3 – Descrierea funcționalităților jocului dezvoltat

Jocul dezvoltat este format din 2 parți, prima parte este reprezentată de către meniul principal, meniul pentru setări și meniul pentru finalul jocului, a doua parte este reprezentată de jocul propriu zis. În cele ce urmează voi prezenta în detaliu ambele părți și funcționalitățile lor.

## 3.1 Meniul principal al jocului

Odată cu deschiderea jocului vom fi întâmpinați de prima scena a jocului, aceasta constă din două canvasuri. Dintre acestea, primul pe care îl vom vedea și cu care vom putea interacționa este acela pentru meniul principal care constă din 2 elemente de bază unul neinteractiv și unul interactiv:

1. Fundalul

Fundalul este un element neinteractiv care constă dintr-un element de tip imagine care reprezintă imaginea de fundal a meniului și un element de tip TextMeshPro care reprezintă textul pentru titlul jocului.

Map

Description automatically generated

Fig. 11 Meniul principal al jocului

1. Butoane

Cele 3 butoane reprezintă elementul interactiv din cadrul meniului principal, fiecare buton are o funcționalitate definită în scriptul „Buttons”

Butonul PLAY are funcționalitatea de a schimba scena curenta cu scena următoare, în acest caz scena următoare este setata ca si scena unde se va desfășura jocul, si de a reseta variabilele importante si care pot fi schimbate în cadrul jocului. Toate acestea sunt realizare în funcția StartGame din cadrul scriptului.

public int GameStartScene;  
  
 public void StartGame()  
 {  
 GameController.Health = 6;  
 GameController.MoveSpeed = 5f;  
 GameController.PlayerDamage = 1;  
 GameController.FireRate = 0.5f;  
 SceneManager.LoadScene(GameStartScene);  
 PauseMenu.isPaused = false;  
 Time.timeScale = 1f;

}

Butonul SETTINGS va dezactiva obiectul în care se afla meniul principal și îl va activa pe cel în care se afla meniul de setări. Acest lucru se poate realiza fără folosirea unui script dedicat deoarece Unity ne pune la dispoziție funcții predefinite pentru manipularea simpla a obiectelor cum ar fi setarea acestora drept active sau inactive.

Text

Description automatically generatedText

Description automatically generated

Fig. 12 Ierarhia scenei înainte și după ce a fost apăsat butonul SETTINGS

Butonul EXIT are rolul de a închide jocul acest lucru se realizează prin cadrul funcției ExitGame.

public void ExitGame()

{

Application.Quit();

}

## 3.2 Meniul pentru setări

Meniul de setări are rolul de a schimba diferite aspecte ale jocului pentru o experiență mai buna a utilizatorului. În cadrul acestui joc, aspectele care vor fi schimbate în acest meniu vor fi doar aspecte grafice precum rezoluția ferestrei jocului, daca jocul va fi in modul ecran complet si nivelul de detaliu al jocului.

A screenshot of a video game

Description automatically generated with medium confidence

Fig. 13 Meniul de setări

Acest meniu are 3 funcționalități definite in cadrul scriptului SettingsMenu :

* Setarea jocului în modul ecran complet, acest lucru se realizează foarte simplu în Unity datorita funcțiilor predefinite.

public void SetFullscreen(bool isFullscreen)

{

Screen.fullScreen = isFullscreen;

}

Această funcție are ca parametru valoarea butonului din cadrul meniului. Daca butonul este selectat valoarea parametrului va fi true si va seta variabila fullScreen predefinită de Unity ca true, ceea ce face ca jocul sa treacă în modul ecran complet și vice versa daca butonul nu este selectat.

* Alegerea rezoluției dorite este realizată cu ajutorul unui meniu de tip dropdown care este populat la rularea programului cu toate rezoluțiile pe care utilizatorul le poate folosii.

Resolution[] resolutions;

public TMPro.TMP\_Dropdown resolutionDropdown;

private void Start()

{

resolutions = Screen.resolutions;

int currResolutionIndex = 0;

resolutionDropdown.ClearOptions();

List<string> dropdownOptions = new List<string>();

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

{

string option = resolutions[i].width + " x " + resolutions[i].height;

dropdownOptions.Add(option);

if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.height)

{

currResolutionIndex = i;

}

}

resolutionDropdown.AddOptions(dropdownOptions);

resolutionDropdown.value = currResolutionIndex;

resolutionDropdown.RefreshShownValue();

}

public void SetResoultion(int resolutionIndex)

{

Resolution resolution = resolutions[resolutionIndex];

Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);

}

Această funcție salvează într-un vector de rezoluții toate rezoluțiile pe care ecranul utilizatorului le poate folosii, acestea fiind deja salvate de către Unity dar intra-un format care trebuie puțin modificat. Odată ce are vectorul, va trece prin el și va adăuga fiecare element in meniul de tip dropdown, în același timp verifica daca rezoluția adăugată este rezoluția ecranului folosit de către utilizator, daca da ii va salva indexul si o va seta ca rezoluția implicită. Funcția SetResolution are ca parametru indexul opțiunii selectate si schimba rezoluția curenta cu cea aleasă cu ajutorul funcției SetResolution predefinită de Unity.

* Alegerea calității imaginii pentru joc, este proiectată în mare parte precum alegerea rezoluției, dar în loc sa luam o lista cu rezoluții facem o listă de nivele de calitate scrisă de mana.

public void SetQuality(int qualityIndex)

{

QualitySettings.SetQualityLevel(qualityIndex);

}

QualityIndex este indexul opțiunii selectate, iar schimbarea calității se face prin funcția SetQualityLevel aceste nivele fiind definite în setările proiectului.

A picture containing text, calculator, electronics

Description automatically generated

Fig. 14 Lista de nivele de calitate definite de către mine

## 3.3 Metode si funcționalități pentru caracterul jucătorului

Odată ce jucătorul va apăsa butonul PLAY jocul propriu-zis va începe, acesta constă în controlarea unui caracter cu tastele W, A, S, D și aruncarea de proiectile cu ajutorul săgeților. Aceste 2 acțiuni sunt controlate de 3 scripturi de bază.

Primul script si posibil cel mai important este GameController, acesta se ocupă cu controlul variabilelor cele mai importante din cadrul jocului, cum ar fii punctele de viață ale jucătorului, viteza sa de mișcare, viteza cu care poate arunca proiectile, viața inamicilor și așa mai departe. Acest lucru este realizat prin folosirea de metode get si set pentru fiecare variabilă.

private static float health = 6;

public static float Health { get => health;set => health = value; }

private static float maxHealth = 6;

public static float MaxHealth { get => maxHealth;set => maxHealth = value; }

private static float moveSpeed = 5f;

public static float MoveSpeed { get => moveSpeed; set => moveSpeed = value; }

private static float playerDamage, playerDamageDefault = 1;

public static float PlayerDamage { get=> playerDamage; set => playerDamage = value; }

private static int enemyDamage = 1;

public static int EnemyDamage { get => enemyDamage; set => enemyDamage = value; }

private static int rangedEnemyDamage = 2;

public static int RangedEnemyDamage { get => rangedEnemyDamage; set => rangedEnemyDamage = value; }

private static float fireRate, fireRateDefault = 0.5f;

public static float FireRate { get => fireRate; set => fireRate = value; }

private static float bulletSpeed = 7f;

public static float BulletSpeed { get => bulletSpeed; set => bulletSpeed = value; }

private static float bulletSize = 0.5f;

public static float BulletSize { get => bulletSize; set => bulletSize = value; }

private static float bulletLifetime = 1f;

public static float BulletLifetime { get => bulletLifetime; set => bulletLifetime = value; }

Fiecare variabila este declarată și inițializată cu valoarea sa de bază, după care îi sunt definite metodele de get si set. Acest lucru este făcut pentru ca majoritatea acestor variabile vor scădea sau vor crește pe parcursul jocului, aceste modificări sunt controlate de diferite metode din cadrul acestui script.

public static void DamagePlayer (int damage)

{

health -= damage;

if(health <= 0)

{

KillPlayer();

}

}

public static void HealPlayer(float healAmount)

{

health = Mathf.Min(maxHealth, health + healAmount);

}

public static void DamageChange(float damage)

{

playerDamage += damage;

}

public static void MoveSpeedChange (float speed)

{

moveSpeed += speed;

}

public static void FireRateChange(float rate)

{

fireRate -= rate;

}

public static void BulletSizeChange(float size)

{

bulletSize += size;

}

internal static void MaxHealthChange(float health)

{

maxHealth += health;

}

public static void KillPlayer()

{

SceneManager.LoadScene(2);

}

Aceste metode controlează schimbarea variabilelor de joc precum punctele de viață ale jucătorului prin metodele HealPlayer si DamagePlayer sau viteza cu care jucătorul poate arunca cu proiectile prin metoda FireRateChange. Singura metoda din cadrul acestui script care nu se ocupa cu schimbarea variabilelor de bază ale jocului este KillPlayer, aceasta este chemată în cadrul metodei DamagePlayer daca jucatorul nu mai are destule puncte de viață pentru a continua jocul și va schimba scena curenta cu scena fiala a jocului.

Al doilea script care se ocupă cu controlul jucătorului se numește PlayerController și acesta se ocupa cu mișcarea obiectului în cadrul jocului și cu aruncarea proiectilelor de către jucător. Mișcarea caracterului este realizată în 3 pași.

Primul pas constă in obținerea componentei RigidBody a obiectului, aceasta se ocupa cu detecția coliziunilor in cadrul jocului și o putem folosii pentru a mișca obiectul. Obținerea acestei componente este realizata la începerea scriptului, în cadrul metodei Start care este executată odată ce scena în care este inclus obiectul atașat acestui script pornește.

void Start()

{

rigidbody = GetComponent<Rigidbody2D>();

}

Al doilea și al treilea pas se produc aproape simultan în cadrul metodei Update, care este chemată o dată în fiecare frame. În primul rând, vor fi luate imputule de pe tastele pentru mișcarea caracterului (W,A,S,D) și vor fi păstrate în cadrul a două variabile „horizontal” și „vertical”, acestea vor avea valori intre -1 și 1. Odată ce avem valorile în aceste variabile, vom folosii funcția „velocity” din cadrul componentei RigidBody.

float horizontal = Input.GetAxis("Horizontal");  
 float vertical = Input.GetAxis("Vertical");  
  
 rigidbody.velocity = new Vector3(horizontal \* speed, vertical \* speed, 0);

O altă funcționalitate foarte importantă pe care o are caracterul jucătorului este aruncarea de proiectile. Acest lucru se realizează foarte similar cu funcționalitatea de mișcare a jucătorului dar în loc sa mișcam un RigidBody vom instanția proiectilele.

void Shoot(float x,float y)

{

GameObject bullet = Instantiate (bulletPrefab,transform.position,transform.rotation) as GameObject;

bullet.AddComponent<Rigidbody2D>().gravityScale = 0;

bullet.GetComponent<Rigidbody2D>().velocity = new Vector3(

(x < 0) ? Mathf.Floor(x) \* bulletSpeed : Mathf.Ceil(x) \* bulletSpeed,

(y < 0) ? Mathf.Floor(y) \* bulletSpeed : Mathf.Ceil(y) \* bulletSpeed,

);

}

În cadrul acestei metode vom inițializa un obiect la coordonatele jucătorului si îi vom adăuga o componenta de tipul proiectil la coordonatele jucătorului. După ce am adăugat această componentă, o vom face și să se miște, acest lucru se face în funcție de valorile primite ca parametrii. Pentru ca viteza proiectilelor sa fie mereu constantă vom rotunjii x sau y la cel mai apropiat număr întreg diferit de zero (dacă este <0 va fi transformat in -1 și dacă este >0 în 1).

## 3.4 Metode si funcționalități pentru camerele labirintului

Un aspect important în cadrul jocului sunt camerele pe care jucătorul le va explora in timpul jocului ele având în cadrul lor inamici care trebuisec învinși pentru a putea explora altă camera. Acest lucru se realizează folosind scene predefinite de mine în cadrul proiectului, fiecare scenă reprezintă un tip de cameră care poate apărea în timpul jocului. Camerele pot fi cea inițială în care va începe jocul, cea în care se va sfârșii sesiunea de joc, o cameră goală și o cameră in care pot apărea inamici sau obiecte utile.

Cea mai importantă și interesantă cameră din cadrul jocului este aceea în care pot apărea inamici sau obiecte utile jucătorului. Acest lucru se realizează cu ajutorul unor scripturi dedicate acestei funcționalități, cele mai importante sunt acela pentru generarea matricei din cadrul camerei, în care se pot genera obiectele și cel care se ocupă cu generarea de obiecte in cadrul matricei generate.

Începând cu primul script din aceste două, cel pentru generarea matricei in cadrul camerei , numit GridController. În cadrul acestui script sunt folosite câteva variabile pentru definirea comportamentului său, acestea sunt:

public Room room;

[System.Serializable]

public struct Grid

{

public int columns, rows;

public float verticalOffset, horizontalOffset;

}

public Grid grid;

public GameObject gridTile;

public List<Vector2> availablePoints = new List<Vector2>();

* room, care reprezintă camera în care matricea va fi generată.
* Structura Grid care conține informații despre câte linii și coloane va avea matricea cât și offsetul orizontal si vertical folosit pentru a alinia matricea in dreptul camerei.
* gridTile care conține o imagine cu care vom putea vedea si modifica structura matricei pentru debugging, ea va fi ascunsă în timpul jocului.
* availablePoints care conține toate punctele in care vor putea fi generate obiecte.

Generarea matricei se face prin cadrul metodei GenerateGrid unde în în cadrul camerei se verifica pozițiile valide în care se poate pune un element de tip gridTile care va fi numit „X:coordonate Y:coordonate” pentru a face debugging-ul mai usor. Unele poziții vor fi trecute în lista pozitiiInvalide, acestea sunt pozițiile în care am considerat ca inamicii nu ar trebui sa fie, acestea sunt în fata ușilor pentru ca jucătorul sa nu fie atacat imediat cum intră în cameră.

public void GenerateGrid()

{

grid.verticalOffset += room.transform.localPosition.y;

grid.horizontalOffset += room.transform.localPosition.x;

var pozitiiInvalide = new List <(int, int)> // acestea sunt toate pozitiile in care nu se pot spawna monstrii sau iteme

{ //usa jos

(6,0),(7,0),(8,0),(9,0),(10,0),(6,1),(7,1),(8,1),(9,1),(10,1),

//usa stanga

(0,3),(1,3),(2,3),(0,4),(1,4),(2,4),(0,5),(1,5),(2,5),

//usa sus

(6,7),(7,7),(8,7),(9,7),(10,7),(6,8),(7,8),(8,8),(9,8),(10,8),

//usa dreapta

(13,3),(14,3),(15,3),(13,4),(14,4),(15,4),(13,5),(14,5),(15,5)

};

for (int y = 0; y < grid.rows; y++)

{

for(int x = 0; x < grid.columns; x++)

{

if (pozitiiInvalide.Any(t => t.Item1 == x && t.Item2 == y))

{

//ignoram cazul

}

else

{

GameObject go = Instantiate(gridTile, transform);

go.GetComponent<Transform>().position = new Vector2(x - (grid.columns - grid.horizontalOffset), y - (grid.rows - grid.verticalOffset));

go.name = "X: " + x + "Y: " + y;

availablePoints.Add(go.transform.position);

go.SetActive(false);

}

}

}

GetComponentInParent<ObjectRoomSpawner>().InitializeObjectSpawning();

}

Chart, treemap chart

Description automatically generated

Fig. 15 Matricea generată după rularea scriptului GridController

În matricea generată mai sus pătratele sunt pozițiile valide în care poate fi generat un inamic sau un obiect. Acest lucru este făcut de către scriptul ObjectRoomSpawner care se folosește de câteva variabile importante in cadrul său:

public struct RandomSpawner

{

public string name;

public SpawnerData spawnerData;

}

public GridController grid;

public RandomSpawner[] spawnerData;

* Structura RandomSpawner care conține numele obiectului care va fi creat si datele sale referitoare la conditiile în care va fi creat si cum va fi creat (numarul maxim,minim etc.).
* grid care este o referință la matricea generată in cadrul camerei.

Crearea de obiecte în cadrul camerei se face în cadrul metodei SpawnObject, care alege un număr la întâmplare care reprezintă numărul de iterații care le va face această funcție și un alt număr la întâmplare care reprezintă poziția în cadrul matricei unde va fi creat obiectul. După ce obiectul este creat, poziția din matrice va fi ștearsă din lista de poziții valide.

void SpawnObject(RandomSpawner data)

{

int randomIteration = Random.Range(data.spawnerData.minSpawn,data.spawnerData.maxSpawn+1);

for(int i = 0; i < randomIteration; i++)

{

int randomPos = Random.Range(0, grid.availablePoints.Count - 1);

GameObject go = Instantiate(data.spawnerData.itemToSpawn, grid.availablePoints[randomPos],Quaternion.identity,transform) as GameObject;

grid.availablePoints.RemoveAt(randomPos);

//Debug.Log("Spawned Object");

}

  }

**3.4 Generarea labirintului**

Cel mai important aspect din cadrul jocului este labirintul pe care jucătorul îl va explora. Acesta este la bază o colecție de scene (camere) amplasate una lângă alta. Această funcționalitate este cea mai amplă din cadrul proiectului, întinzându-se pe o multitudine de scripturi care se ocupă de mai multe aspecte

În prima faza vom avea nevoie de crawleri care se vor ocupa cu punerea bazelor labirintului. Aceștia sunt definiți în cadrul scriptului DungeonCrawler împreună cu funcționalitatea acestora de a-și schimba poziția.

public class DungeonCrawler : MonoBehaviour

{

public Vector2Int Position { get;set; }

public DungeonCrawler(Vector2Int startPos)

{

Position = startPos;

}

public Vector2Int Move(Dictionary<Direction, Vector2Int> directionMovementMap)

{

Direction toMove =(Direction) Random.Range(0, directionMovementMap.Count);

Position += directionMovementMap[toMove];

return Position;

}

}

Totodată avem nevoie și de niște informații suplimentare cum ar fi numărul de crawleri care doresc să fie generați și numărul minim, maxim de camere pe care aceștia le vor genera. Toate aceste informații sunt definite în scriptul DungeonGenerationData, care va crea un obiect de joc cu toate aceste informații pentru o modificare mai ușoară a acestora în cadrul jocului.

[CreateAssetMenu (fileName ="DungeonGenerationData.asset",menuName = "DungeonGenerationData/Dungeon Data")]

public class DungeonGenerationData : ScriptableObject

{

public int numberOfCrawlers;

public int intereationMin;

public int intereationMax;

}

Acum că avem toate ce avem nevoie, putem începe generarea labirintului. Acest lucru este realizat de scriptul DungeonCrawlerController. În prima parte a scriptului va fi creat un dicționar în care vom definii direcțiile pe care crawlerul le poate alege, acestea fiind sus, jos, stânga și dreapta deoarece dacă s-ar mișca pe diagonală camerele nu ar putea fi accesibile de către jucător.

public enum Direction

{

up = 0,

left = 1,

down = 2,

right = 3

};

public class DungeonCrawlerController : MonoBehaviour

{

public static List<Vector2Int> positionsVisited = new List<Vector2Int>();

private static readonly Dictionary<Direction, Vector2Int> directionMovementMap = new Dictionary<Direction, Vector2Int>

{

{ Direction.up, Vector2Int.up },

{ Direction.left, Vector2Int.left },

{ Direction.down, Vector2Int.down },

{ Direction.right, Vector2Int.right }

};

În cea de a doua parte a scriptului este definită metoda GenerateDungeon care va folosii datele din DungeonGenerationData pentru a crea numărul specificat de crawleri și dicționarul definit mai sus pentru a face fiecare crawler să aleagă cate o direcție la întâmplare, acest lucru este realizat de N ori, N fiind un număr la întâmplare intre iterationMin și iterationMax. Fiecare mișcare făcută de către un crawler va fi introdusă într-o listă cu toate pozițiile vizitate de către aceștia.

public static List<Vector2Int> GenerateDungeon(DungeonGenerationData dungeonData)

{

List<DungeonCrawler> dungeonCrawlers = new List<DungeonCrawler>();

for (int i = 0; i < dungeonData.numberOfCrawlers; i++)

{

dungeonCrawlers.Add(new DungeonCrawler(Vector2Int.zero));

}

int iterations = Random.Range(dungeonData.intereationMin, dungeonData.intereationMax);

for (int i = 0; i < iterations; i++)

{

foreach(DungeonCrawler dungeonCrawler in dungeonCrawlers)

{

Vector2Int newPos = dungeonCrawler.Move(directionMovementMap);

positionsVisited.Add(newPos);

}

}

return positionsVisited;

}

Acum că avem toate pozițiile vizitate, vom trece prin fiecare și vom crea camerele propriu zise. Acest lucru este realizat de către două scripturi, primul dintre acestea și cel mai simplu este DungeonGenerator care este alcatuit din 2 metode, Start și SpawnRooms.

În cadrul metodei Start se va genera lista de poziții valide pentru camere și se va chema metoda SpawnRooms definită mai jos.

public DungeonGenerationData dungeonGenerationData;

private List<Vector2Int> dungeonRooms;

private void Start()

{

dungeonRooms = DungeonCrawlerController.GenerateDungeon(dungeonGenerationData);

SpawnRooms(dungeonRooms);

}

Metoda SpawnRooms va trece prin lista de poziții vizitate și va genera cameră în fiecare poziție.

private void SpawnRooms (IEnumerable<Vector2Int> rooms)

{

RoomController.instance.LoadRoom("Start", 0, 0);

foreach(Vector2Int roomLocation in rooms)

{

RoomController.instance.LoadRoom(RoomController.instance.GetRandomRegularRoomName(), roomLocation.x, roomLocation.y);

}

}

Acest lucru este realizat cu ajutorul celui de al doilea script numit RoomController. El se ocupă cu majoritatea lucrurilor când vine vorba despre generarea de camere și este mult prea amplu pentru a putea analiza fiecare funcționalitate a acestuia, dar vom analiza cele mai importante metode din cadrul acestuia.

Principala metodă din cadrul acestei funcții este UpdateRoomQueue care se ocupa cu coada de camere care urmează sa fie adăugate în cadrul jocului sau care sunt in procesul de a fi adăugate.

void UpdateRoomQueue()

{

if(isLoadingRoom)

{

return;

}

if (loadRoomQueue.Count == 0)

{

if (!spawnedBossRoom)

{

StartCoroutine(SpawnBossRoom());

}

else if (spawnedBossRoom && !updatedRooms)

{

foreach (Room room in loadedRooms)

{

room.RemoveUnconnectedDoors();

}

UpdateRooms();

updatedRooms = true;

}

return;

}

currentLoadRoomData = loadRoomQueue.Dequeue();

isLoadingRoom = true;

StartCoroutine(LoadRoomRoutine(currentLoadRoomData));

}

Aceasta începe cu verificarea unor condiții:

1. Prima instrucțiune if verifică dacă jocul este in prezent în procesul de încărcare a unei camere. Daca este, metoda se întoarce rapid si nu va face nimic altceva.
2. A doua instrucțiune if verifică dacă coada de camere de încărcat este goală. Dacă este, metoda verifică dacă camera șefului a fost deja generată. Dacă nu a fost, se pornește o co-rutină pentru a genera camera șefului. Dacă camera șefului a fost deja generată, metoda actualizează camerele deja încărcate prin eliminarea ușilor neconectate și apoi actualizează camerele. După aceea, setează un indicator pentru a indica faptul că camerele au fost actualizate și apoi se întoarce rapid.

Dacă după verificarea acestor condiții încă mai există camere în coada de încarcat , metoda preia următoarea cameră de încărcat și setează un indicator pentru a indica faptul că jocul este acum în procesul de încărcare a unei camere. În cele din urmă, metoda pornește o co-rutină numită LoadRoomRoutine, trecând datele camerei care trebuie încărcate.

O altă metodă foarte importantă din cadrul acestui script este cea numită LoadRoomRoutine care se ocupă cu generarea asincronă a scenelor de joc, ceea ce poate îmbunătății performanța si reactivitatea jocului.

IEnumerator LoadRoomRoutine(RoomInfo info)

{

string roomName = currentWorldName + info.name;

AsyncOperation loadRoom = SceneManager.LoadSceneAsync(roomName,LoadSceneMode.Additive);

while(loadRoom.isDone ==false)

{

yield return null;

}

}

În primul rând se declară variabila roomName care conține numele scenei ce urmează sa fie încarcată, se declară de asemenea și o operațiune asincronă loadRoom, care încarcă scena cu numele definit mai sus folosind modul de încărcare aditiv (scena va fi încărcata in scena părinte). În timp ce camera se încarcă se va verifica constant daca procesul s-a terminat sau nu.

După ce toate camerele au fost încărcate în cadrul jocului se va încărca camera finală. Acest lucru se face prin metoda SpawnBossRoom.

IEnumerator SpawnBossRoom()

{

spawnedBossRoom = true;

int loadedRoomsCount = loadedRooms.Count;

yield return new WaitForSeconds(0.5f);// astept jumatate de secunda sa fiu sigur ca toate camerele au fost create

if (loadRoomQueue.Count == 0)

{

Room bossRoom = loadedRooms[loadedRooms.Count - 1];

Room tempRoom = new Room(bossRoom.X, bossRoom.Y);

Destroy(bossRoom.gameObject);

var roomToRemove = loadedRooms.Single(room => room.X == tempRoom.X && room.Y == tempRoom.Y); //AICI MODIFIC CUM SE GENEREAZA CAMERA BOSSULUI

loadedRooms.Remove(roomToRemove);

LoadRoom("End", tempRoom.X, tempRoom.Y);

}

}

La început se setează valoarea variabilei spawnedBossRoom cu true, acest lucru semnalează că a fost generata și camera finală. Apoi, co-rutina așteaptă jumătate de secundă pentru a se asigura ca toate camere sunt încărcate. Dacă nu există camere rămase de încărcat și camera finală nu a fost încă generată, se ia ultima cameră încărcată, se generează o cameră temporară cu aceleași coordonate, se îndepărtează camera finală din lista de camere încărcate și se încarcă camera finală în locul camerei temporare.

# Capitolul 4 - Testarea aplicației și rezolvarea bug-urilor

Testarea aplicației și rezolvarea bug-urilor sunt etape critice în procesul de dezvoltare a oricărei aplicații software, iar în cazul jocului nostru video 2D în Unity, aceste etape sunt la fel de importante. În acest capitol, vom discuta despre diferitele probleme care au fost observate la testarea jocului de către mine și unii voluntari. Vom explora metodele utilizate pentru a identifica și raporta bug-urile, precum și modul în care acestea au fost rezolvate. De asemenea, vom discuta despre importanța testării continue și a actualizării jocului pentru a asigura o experiență de joc plăcută și fără probleme pentru utilizatori.

Metoda utilizată pentru testarea jocului a implicat o abordare mai puțin formală, în sensul că nu a fost o procedură standardizată și definită în avans. În locul acesteia, am invitat diverse persoane să joace versiunea curentă a jocului și să îmi comunice orice bug-uri, probleme sau nemulțumiri pe care le-au întâmpinat pe parcursul sesiunii de joc. Această metodă neformală de testare a fost aleasă pentru a obține o imagine cât mai detaliată a experienței de joc a utilizatorilor, oferindu-le acestora posibilitatea de a se exprima liber și de a înregistra feedback-ul lor în timp real.

În timpul sesiunilor de testare ale jocului, s-au identificat trei probleme majore semnalate de utilizatori. Prima problemă a constat în posibilitatea jucătorului de a trage proiectile prin pereții camerei, permițând astfel lovirea inamicilor fără ca aceștia să poată reacționa. Soluția implementată a constat în adăugarea unei componente de tip RigidBody fiecărui perete din cameră, care detectează contactul cu un proiectil și îl șterge. Această soluție simplă a dus la rezolvarea problemei într-un mod eficient și fără a afecta experiența de joc a utilizatorilor.

A doua problemă a constat în faptul că jucătorul putea să colecteze un număr mare de obiecte care îi regenerau punctele de viață, depășind astfel limita maximă de puncte de viață afișată pe ecran. Pentru a remedia această problemă, s-a introdus o secvență de cod in cadrul funcției HealPlayer care verifică dacă punctele de viață ale jucătorului ar depășii punctele maxime de viață, astfel încât jucătorul să poată colecta în continuare aceste obiecte pentru a crește scorul final al sesiunii de joc, dar să nu îi crească punctele de viață.

public static void HealPlayer(float healAmount)

{

if(health + healAmount > MaxHealth)

{

health = maxHealth;

}

else

{

health += healAmount;

}

}

A treia problema consta în abilitatea inamicilor de fi creați foarte aproape sau chiar peste ușile de intrare în cameră, făcându-i sa atingă și prin urmare sa lovească jucătorul fără ca acesta sa aibă timp sa reacționeze. Această problema a fost rezolvată prin modificarea matricei în care inamicii pot fi creați (fig.15) în așa fel încât sa nu poată apărea prea aproape de caracter.

După rezolvarea celor trei probleme semnalate, s-a observat o îmbunătățire semnificativă a experienței jocului și a jucabilității acestuia. Implementarea unei metode bine definite de testare a jocului ar putea ajuta la identificarea altor probleme într-un mod mai eficient și sistematic, îmbunătățind astfel calitatea generală a jocului. Este recomandat ca dezvoltatorii de jocuri să aloce timp și resurse pentru testarea și identificarea problemelor înainte de lansarea finală a jocului, deoarece acest lucru poate avea un impact semnificativ asupra reputației și succesului jocului. De asemenea, este important să se ia în considerare feedback-ul utilizatorilor și să se fie deschiși la sugestii și îmbunătățiri, astfel încât să se poată îmbunătăți și mai mult experiența jocului în viitor.

# Concluzii

## Concluzii

Dezvoltarea jocului video 2D în Unity a presupus pentru mine o serie de etape esențiale, incluzând planificarea, proiectarea, implementarea, testarea și rezolvarea problemelor. Prin aplicarea unei metodologii de dezvoltare riguroase, am reușit să creez un joc video 2D care îndeplinește standardele așteptate și care oferă o experiență plăcută și captivantă pentru jucători. În timpul testării jocului, am identificat și am reușit să rezolv cu succes o serie de probleme legate de gameplay și de performanța aplicației. Astfel, jocul final este o versiune îmbunătățită și optimizată, care poate fi utilizată ca punct de plecare pentru dezvoltarea de jocuri video mai avansate și mai complexe în viitor.

## Moduri de îmbunătățire

În ceea ce privește îmbunătățirea jocului proiectat, există mai multe modalități prin care acesta ar putea fi adus la nivelul următor și ar putea oferi o experiență de joc și mai captivantă.

În primul rând, se poate lua în considerare refacerea totală a texturilor jocului, în special dacă acestea sunt considerate depășite sau nu se potrivesc cu stilul și tematica jocului. Aceasta ar putea fi o sarcină pentru un artist experimentat care să poată crea texturi noi și mai atractive, care să sporească realismul și atmosfera jocului.

În al doilea rând, se poate lua în considerare adăugarea de noi obiecte interesante pe care jucătorul să le poată găsi și colecta în timpul jocului. Aceste obiecte ar putea avea un impact direct asupra jocului, oferind recompense sau deblocând noi caracteristici sau niveluri. De asemenea, aceste obiecte ar putea fi ascunse în locuri secrete sau dificil de accesat, ceea ce ar încuraja jucătorii să exploreze mai mult jocul.

În al treilea rând, adăugarea de inamici noi și interesanți ar putea îmbunătăți și mai mult experiența jocului. Acești inamici ar putea avea comportamente diferite, astfel încât jucătorii să fie nevoiți să găsească noi strategii pentru a-i înfrânge. De asemenea, inamicii ar putea fi însoțiți de noi sunete și efecte vizuale care să le facă să fie mai amenințători și mai captivanți.

În al patrulea rând, adăugarea de camere noi ar putea să le ofere jucătorilor o experiență nouă și diversificată în timpul jocului. Aceste camere noi ar putea să includă noi obstacole, inamici sau obiecte de colectat, ceea ce ar putea încuraja jucătorii să exploreze mai mult și să descopere noi aspecte ale jocului.

În final, adăugarea unui șef puternic și captivant la sfârșitul labirintului ar putea să fie un mod excelent de a încheia jocul și de a oferi jucătorilor o provocare finală și de neuitat. Acest șef ar putea avea puteri și abilități unice, astfel încât jucătorii să fie nevoiți să găsească noi strategii pentru a-l înfrânge.

În concluzie, prin adăugarea de noi obiecte, inamici, camere și un șef puternic și captivant, jocul proiectat poate fi îmbunătățit semnificativ și poate oferi o experiență de joc mai bogată și mai captivantă pentru jucători.

## Posibile noi direcții de abordare

Pentru a explora noi direcții de abordare, se poate lua în considerare integrarea de tehnologii noi și inovative, precum realitatea virtuală sau augmentată, care ar putea oferi o experiență de joc mai interactivă și mai captivantă. De asemenea, se poate explora dezvoltarea de jocuri video cu mai multe dimensiuni sau în spații virtuale mai complexe, ceea ce ar putea permite o mai mare libertate de mișcare și interacțiune pentru jucători. De asemenea, dezvoltarea de jocuri video multiplayer ar putea oferi oportunități noi pentru a aduce comunitățile de jucători împreună și pentru a spori competitivitatea. De asemenea, se poate explora integrarea de tehnologii inteligente, cum ar fi inteligența artificială și învățarea automată, pentru a crea inamici mai inteligenți și mai adaptabili și pentru a oferi jucătorilor o experiență mai personalizată. Toate aceste direcții de abordare ar putea deschide noi orizonturi pentru dezvoltarea de jocuri video captivante și inovative.

# Bibliografie

[1] Tom Wijman , Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018, <https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half>

[2] Ben Stegner , What Are Roguelike and Roguelite Video Games? , [https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/](https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/%20)

[3] \*\*\* , Explainer: What is a Dungeon Crawler game ?, <https://boardgamingparent.com/explainer-what-is-a-dungeon-crawler-game/>

[4] \*\*\*, Cuphead , [https://en.wikipedia.org/wiki/Cuphead#](https://en.wikipedia.org/wiki/Cuphead%23)

[5]\*\*\*,Mafia (party game), <https://en.wikipedia.org/wiki/Mafia_(party_game)>

[6]\*\*\*,The Thing (1982 film) , <https://en.wikipedia.org/wiki/The_Thing_(1982_film)>

[7]\*\*\*, UNREAL ENGINE REVIEW: PROS, CONS, AND SUITABILITY , <https://www.newgenapps.com/en/blogs/unreal-engine-review-pros-cons-and-suitability>

[8] \*\*\*,The Binding of Isaac (video game), <https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video_game)>

[9] Joe Hocking , Unity in Action: Multiplatform Game Development in C#

[10] Tracy Fullerton , Game Design Workshop: A Playcentric Approach to Creating Innovative Games

[11] Jason Gregory , Game Engine Architecture

[12] Robert Nystrom , Game Programming Patterns

[13]Jesse Schell, The Art of Game Design: A Book of Lenses

1. <https://unity.com> [↑](#footnote-ref-1)
2. <https://assetstore.unity.com> [↑](#footnote-ref-2)
3. <https://www.polygon.com/cuphead-guide/2017/10/3/16391938/king-dice-all-bets-are-off> [↑](#footnote-ref-3)
4. <https://www.ign.com/wikis/cuphead/Run_&_Gun-_Forest_Follies> [↑](#footnote-ref-4)
5. <https://www.infogamerhub.com/how-to-make-among-us-reactor-task-mini-game/> [↑](#footnote-ref-5)
6. <https://among-us.fandom.com/wiki/Voting> [↑](#footnote-ref-6)
7. <https://www.unrealengine.com/en-US> [↑](#footnote-ref-7)
8. <https://signup.buildbox.com> [↑](#footnote-ref-8)