#### UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

Where’s Perry?

Joc platformer 2D tip puzzle

#### propusă de

**Moișanu-Costinescu Ștefan**

Sesiunea: *iulie, 2021*

#### Coordonator științific

Lect. Dr. Moruz Alex

#### UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

**FACULTATEA DE INFORMATICĂ**

Where’s Perry?

Joc platformer 2D tip puzzle

**Moișanu-Costinescu Ștefan**

Sesiunea**:** *iulie, 2021*

#### Coordonator științific

***Lect. Dr. Moruz Alex***

Avizat,

Îndrumător Lucrare de Licență,

Lect. Dr. Moruz Alex.

Data Semnătura

#### DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul **Moișanu-Costinescu Ștefan** cu domiciliul în **România, jud. Iași, mun. Iași, str. Ion Creangă, nr. 8,** născut la data de **27 decembrie 1999**, identificat prin CNP **1991227226761**, absolvent al Universității „Alexandru Ioan Cuza” din Iași, **Facultatea de informatică** specializarea **informatică**, promoția **2021**, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: **“Where’s Perry? Joc platformer 2D tip puzzle”** elaborată sub îndrumarea **Lect. Dr. Moruz Alex**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ………………………… Semnătură student: ……………………

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul „*Where’s Perry? Joc platformer 2D tip puzzle*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Moișanu-Costinescu Ștefan**

Data: Semnătura:

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, **Moișanu-Costinescu Ștefan***.*

Încheierea acestui acord este necesară din următoarele motive:

*[Se explică de ce este necesar un acord, se descriu originile resurselor utilizate în realizarea*

*produsului-program (personal, tehnologii, fonduri) și aportul adus de fiecare resursă.]*

Iași, *data*

Decan Conf. Dr. Iftene AdrianAbsolvent *Moisanu-Costinescu Stefan*

(semnătura în original) (semnătura în original)

Cuprins

[1 Introducere 8](#_Toc75376795)

[1.1 Context 8](#_Toc75376796)

[1.2 Motivație 10](#_Toc75376797)

[2 Aplicații similare 12](#_Toc75376798)

[2.1 Portal 2 12](#_Toc75376799)

[2.2 FireBoy and WaterGirl 14](#_Toc75376800)

[3 Tehnologii utilizate 15](#_Toc75376801)

[3.1 Unity 15](#_Toc75376802)

[3.2 Limbajul de programare C# 15](#_Toc75376803)

[3.3 Firebase 16](#_Toc75376804)

[3.3.1 Autentificare 16](#_Toc75376805)

[3.3.2 Bază de date în timp real 16](#_Toc75376806)

[3.4 Adobe Illustrator 17](#_Toc75376807)

[4 Prezentarea aplicației 18](#_Toc75376808)

[4.1 Ferestre 18](#_Toc75376809)

[4.1.1 Schema de navigare 18](#_Toc75376810)

[4.1.2 Meniul principal 19](#_Toc75376811)

[4.1.3 Clasament online 19](#_Toc75376812)

[4.1.4 Mapa nivelelor 19](#_Toc75376813)

[4.1.5 Autentificare / Înregistrare 20](#_Toc75376814)

[5 Detalii de implementare 21](#_Toc75376815)

[5.1 Introducere 21](#_Toc75376816)

[5.2 Baza de date 22](#_Toc75376817)

[5.2.1 Autentificare / Înregistrare 23](#_Toc75376818)

[5.2.2 Clasament online 24](#_Toc75376819)

[5.3 Mediul 25](#_Toc75376820)

[5.3.1 Buton și maneta 26](#_Toc75376821)

[5.3.2 Laser 27](#_Toc75376822)

[5.3.3 Platforma mișcătoare 28](#_Toc75376823)

[5.3.4 Ușa 29](#_Toc75376824)

[5.4 Caractere 29](#_Toc75376825)

[5.4.1 Perry 31](#_Toc75376826)

[5.4.2 Maiorul Monogramă 32](#_Toc75376827)

[5.4.3 Inamic 34](#_Toc75376828)

[6 Analiza nivelelor 36](#_Toc75376829)

[6.1 Nivelul 1 36](#_Toc75376830)

[6.1.1 Nivelul 2 37](#_Toc75376831)

[6.1.2 Nivelul 3 38](#_Toc75376832)

[6.1.3 Nivelul 4 39](#_Toc75376833)

[6.1.4 Nivelul 5 40](#_Toc75376834)

[6.1.5 Nivelul 6 41](#_Toc75376835)

[7 Concluzii finale 43](#_Toc75376836)

[7.1 Studiu 43](#_Toc75376837)

[7.2 Direcții de viitor 45](#_Toc75376838)

[7.3 Concluzie 46](#_Toc75376839)

[8 Bibliografie 48](#_Toc75376840)

[9 Anexe 50](#_Toc75376841)

[9.1 Clasa FirebaseHandler. Funcții relevante 50](#_Toc75376842)

[9.2 Clasa Leaderboard 52](#_Toc75376843)

[9.3 Funcția de calculare a mișcării platformei 54](#_Toc75376844)

# Introducere

## Context

Primele concepte din istoria umanității ce pot fi încadrate în categoria puzzle sunt ghicitorile, iar acestea datează încă din era mitologiei grecești. Se consideră că prima enigmă creată de om este “The Riddle of the Sphinx” (Ghicitoarea Sfinxului) ce se traduce în felul următor: Ce creatură are o voce dar devine patrupedă, apoi bipedă iar apoi trepidă? În procesul de a găsi răspunsul, cel ce încearcă să deslușească misterul se conformează unor reguli și ipoteze inițiale; având acest punct de plecare, se folosește de un proces de gândire logică și sistematică, bazată pe încercare și eșec până converge spre soluție: omul are o voce, iar cele 3 categorii fac referire la 3 etape din viață acestuia: copilăria timpurie, când modalitatea de deplasare se aseamănă patrupedelor, viața de adult și bătrânețea când se folosește un baston pentru sprijin. De atunci și până în ziua de astăzi puzzle-urile s-au aflat în continuă expansiune, căutând să exploateze din ce în ce mai multe arii, cum ar fi dificultatea, mecanica, logica, matematica, iar odată cu tangența cu tehnologia si informatica, au apărut jocurile video, dând astfel glas acestei lucrări de licență.

Jocurile video au devenit, fără îndoială, o parte semnificativă din viață omului modern, aducând astfel un aport considerabil de influență în ceea ce privește modul de dezvoltare al unei persoane, cu atât mai mult la vârste fragede.

|  |
| --- |
| Fig. 1 - Procentul persoanelor ce dețineau un calculator in funcție de nivelul de educație |
|  |

După cum se poate observa în Fig. 1, de la începutul până la finalul anilor 90, procentul persoanelor cu studii superioare ce dețineau un calculator s-a dublat sau chiar triplat în această perioadă de timp. De asemenea, conform Fig. 2, procentul majoritar de persoane care erau pasionate de jocuri video în anul 2020 în Statele Unite se regăsește în intervalul de vârstă de la 18 la 34 de ani. Interpretând aceste mulțimi de date, pot trage concluzia că persoanele trecute de perioada adolescenței în anul 2020 au avut acces la un calculator și primele tangențe cu jocurile video în copilărie, datorită faptului că părinții lor se aflau într-o perioadă de stabilitate financiară în momentul în care acestea au devenit disponibile și accesibile pe piață. Procentul majoritar pe care aceștia îl reprezintă relevă aderarea lor la acest hobby și păstrarea în continuare în viața adultă a acestei activități.

|  |
| --- |
| Fig. 2 - Distribuția după vârsta a utilizatorilor de jocuri video in Statele Unite, în 2020 |

Propun prin intermediul lucrării mele să contribui la dezvoltarea industriei de jocuri video într-o direcție educativă ce aduce potențiale beneficii intelectuale utilizatorului, îmbinând conceptul de puzzle într-un joc de tip platformer 2D pentru sistemul de operare Windows. Consider că provocările pe care le-am înglobat în diversele nivele pe care le-am conceput în jocul meu au posibilitatea de a antrena și de a îmbunătăți latura creierului uman ce se ocupă cu funcțiile executive. Acestea reprezintă o mulțime de procese mentale care ne ajută să conectăm experiența anterioară cu o acțiune din prezent. Exersarea acestei capacități contribuie mai departe la optimizarea unor abilități esențiale, cum ar fi: atenția, planificarea, memoria, selectarea informațiilor senzoriale relevante, monitorizarea și interpretarea stimulilor interni și externi, auto-controlul, acționarea și luarea de decizii. În urma unui studiu realizat de Adam C. Oei si Michael D. Patterson in 2014, a fost trasă concluzia că un joc video puzzle bazat pe mecanici ale fizicii are capacitatea de a îmbunătăți funcțiile executive, spre deosebire de alte categorii de jocuri video. Subiecții au fost expuși înainte de experiment la teste ce vizează schimbarea aleatorie de sarcini, inhibarea răspunsurilor și abilitatea de a filtra stimuli; datele arată că în urma a 20 de ore de joc, persoanele care au interacționat cu aplicația “Cut the Rope”[[1]](#footnote-1) au prezentat cele mai semnificative diferențe de aptitudini când au fost rugați să refacă testele anterioare.

Acest experiment este un punct de plecare bun spre a putea răspunde la întrebarea: Este posibil ca jocurile video de tip puzzle să aibă drept efect antrenarea anumitor aptitudini mentale? Fiind inspirat de această problemă, am decis ca prin această lucrare să-mi setez același punct de plecare, dar privit dintr-o altă perspectiva: Se poate proiecta un joc care să aibă în vedere și să vizeze în mod direct antrenarea unei anumite funcții executive? Astfel, am conceput un joc video de tip platformer 2D ce înglobează printr-o interfață tematică șase nivele de dificultăți variate. Mecanica jocului constă în mișcarea alternativă a două personaje, unul terestru și unul aerian, ce au posibilitatea de a interacționa unul cu celălalt sau cu mediul înconjurător. Scopul unui nivel este de a colabora în vederea eliberării obstacolelor ce nu permit ieșirea din încăpere. Utilizatorul este invitat să accepte provocarea nivelelor de a-și îmbunătăți atenția, capacitatea de planificare și rezolvare de probleme, memoria și gândirea logică, toate sub presiunea unui cronometru. Drept stimulent pentru concentrare și motivație de a obține cel mai bun rezultat, am adăugat un element competitiv; jocul stochează online cele mai bune rezultate pentru fiecare nivel completat cu succes al utilizatorului autentificat, oferind astfel un clasament global al tuturor scorurilor obținute de către toți jucătorii.

## Motivație

Tematica jocului este inspirată din desenul meu animat preferat din copilărie, Phineas și Ferb, ce mi-a captat atenția de la primele episoade vizionate deoarece creatorii au dat dovadă, în opinia mea, de o creativitate și imaginație impresionantă, trăsături pe care consider că mi le-au pasat și mie într-o oarecare măsură prin produsul lor animat. O narativă secundară a serialului se învârte în jurul animalului de companie al celor două personaje principale, un ornitorinc, care în secret este un agent special ce lupta împotriva unui antagonist. Atuul acestui caracter negativ este bagajul de cunoștințe despre știință, ce îi permite să inventeze mașinării și capcane, în afara oricăror reguli ale naturii sau fizicii, ironic, având în vedere profesia sa. Inspirat de relația acestor două personaje, pasiunea pentru puzzle-uri, curiozitatea tehnologiei ce stă în spatele creării jocurilor video și motivat de rezultatul obținut de studiul prezentat anterior, am decis să îmbin cunoștințele de programare dobândite in facultate cu ingeniozitatea și posibilitățile oferite de acest desen animat într-un produs personal original, ce aspiră spre a putea oferi utilizatorului satisfacția și beneficiile aduse de rezolvarea cu succes a unui puzzle prin intermediul tehnologiei.

# Aplicații similare

Există o multitudine de jocuri video ce exploatează diferite mecanici cu scopul de a construi sisteme complexe care plasează jucătorul în scenarii ce propun găsirea unei soluții la problema expusă. Se creează o strategie de abordare a nivelelor folosind gândirea logică pentru a alcătui un plan compus dintr-o înlănțuire de acțiuni, fiecare eșec dezvăluind o nouă informație relevantă. După cum se poate observa, aceeași esență stă atât la baza primului tip de puzzle, ghicitoarea, cât și la baza jocurilor video, iar diferite companii specializate în dezvoltarea jocurilor video au avut inspirația să valorifice acest fapt; prin urmare, în acest capitol voi enunța proiectele care m-au inspirat în a alege această tema.

## Portal 2

Portal 2 este un joc “first-person” 3D de tip platformer, realizat de compania Valve Corporation, a cărui scop principal este părăsirea unei camere, inițial închisă, prin deschiderea unei uși folosind diverse unelte. Mecanica pentru care acest joc și-a creat un renume este reprezentată de o armă ce are capacitatea de a plasa două portaluri interconectate, doar pe anumite suprafețe, ce au proprietăți asemănătoare conceptului de gură de vierme[[2]](#footnote-2). Orice obiect trece prin oricare din cele două terminale își continuă mișcarea prin celălalt, păstrându-și toate proprietățile fizice, cum ar fi direcția de mișcare relativă la portal, viteză, accelerația sau inerția.

|  |  |
| --- | --- |
| Fig. 3 – Ilustrarea modului de funcționare al portalurilor | Un scenariu interesant ce devine posibil profitând de această mecanică este ilustrat în Fig. 3; platforma din dreapta este inițial inaccesibilă, dar utilizând această armă, jucătorul are capacitatea de a se folosi într-un mod ingenios de inerția și viteza ce sunt dobândite prin cădere liberă. Plasând cele două terminale pe suprafețe perpendiculare, direcția de deplasare este rotită cu 90°, continuându-și astfel căderea urmând o traiectorie asemănătoare unui glonț. |

Alte unelte folosite în rezolvarea puzzle-urilor sunt următoarele:

* “Thermal Discouragement Beams” este un laser, ce are drept punct de plecare un emițător fix, care transportă o cantitate de energie necesară pentru a activa alte elemente din mediul înconjurător, necesare pentru finalizarea nivelului.
* “Excursion Funnel” reprezintă o armă ce îi oferă posesorului capacitatea de a manevra prin tractare, mutare, sau împingere obiecte aflate la distanță.
* “Hard Light Bridge” este o platformă ce se extinde de la un emițător pe o traiectorie rectilinie pâna la primul obstacol.
* “Redirection Cube” denotș un cub alcătuit din suprafețe reflectorizante utilizate pentru a manipula direcția laserelor.

Toate aceste elemente plasate strategic în calea jucătorului creează un orizont larg de posibilități și combinații în care pot fi utilizate împreună, motiv pentru care Portal 2 a fost, precum “Cut the Rope”, subiectul unui studiu ce viza același obiectiv: posibilitatea îmbunătățirii anumitor aptitudini din sfera funcțiilor executive. Experimentul desfășurat de către Valerie J. Shute, Matthew Ventura și Fengfeng Ke în anul 2014 scoate în evidență efectele pozitive identificate la participanți, în urma unei sesiuni de opt ore de joc asupra abilităților precum rezolvarea de probleme, aptitudini spațiale și persistența. Testul a fost realizat având drept sistem de referință programul Lumosity[[3]](#footnote-3), față de care Portal 2 a înregistrat îmbunătățiri semnificative în toate cele 3 arii menționate anterior.

## FireBoy and WaterGirl

FireBoy and WaterGirl*,* apărut în 2009,este un joc de tip platformer 2D dezvoltat de Oslo Albet pentru browser-uri web folosind Adobe Flash[[4]](#footnote-4), publicat prima dată pe site-ul Armor Games.Asemănător Portal 2, scopul jocului este de a rezolva puzzle-uri pentru a părăsi camera ce alcătuiește nivelul curent. Jucătorul are posibilitatea de a controla simultan două personaje ce diferă doar la nivel estetic, dar modul în care acestea interacționează cu mediul le deosebește; obiectele alcătuite din același element sunt complet inofensive pentru caracterul corespunzător, în timp ce pentru celălalt sunt fatale.

|  |  |
| --- | --- |
| Fig. 4 – Nivelul 5, WaterGirl and FireBoy in The Forest Temple | Acest concept simplu, ilustrat în Fig. 4, îmbinat cu butoane, manete, balanțe, portaluri, lasere și oglinzi utilizate folosind cooperare, alcătuiesc un produs excelent și educativ pentru cei tineri. Capcanele gândite de dezvoltator necesită gândire logică și în avans, planificare bine pusă la punct, coordonare și cooperare. |

Motivul principal pentru care am ales acest joc drept punct principal de plecare pentru lucrarea mea este faptul că, deși are capacitatea de a pune în dificultate utilizatorii chiar și când este jucat în doi, odată acceptată provocarea de a înfrunta nivelele singur, experiența crește exponențial atât în dificultate, cât și în beneficii. Consider că abilitatea de a controla simultan cele două caractere, evitând pericolele, în timp ce este compus și un plan de rezolvare a puzzle-urilor, poate constitui un antrenament mental de coordonare și multifuncționalitate, chiar și pentru cei mai pricepuți. O dovadă a existenței îmbunătățirilor aduse acestor abilități în urmă interacțiunii cu jocul este faptul că, inițial, orice jucător va controla alternativ caracterele, dar, după o perioada de antrenament, va atinge performanța de a putea fi atent în două locuri diferite pe ecran, coordonându-și mâinile în același timp pentru a controla simultan cele două caractere diferite.

# Tehnologii utilizate

## Unity

Unity este una dintre cele mai bine consolidate unelte alese de dezvoltatorii de jocuri video pentru a-și transforma creațiile din concept în realitate. În calitate de motor și mediu de creare a jocurilor video, Unity furnizează utilizatorilor majoritatea uneltelor necesare pentru realizarea unui astfel de produs, cu alte cuvinte, programatorii nu trebuie să reinventeze roata, ci din potrivă, pot revoluționa piața cu produse de calitate superioară, folosindu-se de o linie de asamblare performantă și complet automată. Pentru a oferi și un sens denotativ acestei analogii, aduc completarea că nu este nevoie să fie calculate și implementate, de fiecare dată, aceleași module ce simulează, de exemplu, modul în care interacționează obiectele în lumea reală, respectând legile fizicii, sau modul în care lumina se reflectă pe diferite suprafețe. Aceste soluții sunt oferite atât pentru crearea aplicațiilor 2D cât și 3D, pentru o multitudine de platforme, cum ar fi Linux, Mac, Windows, Android, IOS, Xbox sau PlayStation.

O altă caracteristică extrem de utilă este magazinul de componente (“Asset Store”); acesta este un spațiu unde oamenii își pot încarca creațiile, punându-le la dispoziția tuturor, contra-cost sau gratuit. Astfel, este alcătuită o comunitate ce promovează ingeniozitatea, creativitatea și generozitatea, unde participanții își pot împărtăși viziunile și contribuie colectiv la dezvoltarea acesteia prin împărtășirea bunurilor precum componente vizuale 2D sau 3D, texturi, materiale, animații, interfețe, biblioteci, efecte speciale și audio.

Unity este în aceeași măsură și un mediu de dezvoltare integrat (Integrated Development Environment) ce are rolul de a centraliza toate aceste instrumente într-un singur editor, cu o interfață grafică extrem de intuitivă și ușor de manipulat. Acesta a ajuns la performanța în care o simplă interacțiune între obiecte poate fi simulată doar prin adăugarea în scenă a elementelor deja puse la dispoziție, urmată de ajustarea proprietăților acestora, fără a tasta o singură linie de cod.

## Limbajul de programare C#

C# este un limbaj de programare modern, orientat-obiect ce permite dezvoltarea produselor software într-o manieră naturală, remarcându-se prin simplitate, utilitate generală și productivitate sporită. Furnizează programatorului uneltele necesare pentru construirea unei aplicații robuste și de durată, precum colectorul de gunoi pentru gestionarea eficientă a memoriei, tipul de dată null pentru variabile ce nu au asociate obiecte instanțiate, gestionarea excepțiilor, expresii lambda, clase, suport pentru operații asincrone și multe altele.

Deși Unity are puterea de a realiza produse impresionant de complexe fără a scrie cod, pentru a-mi atinge scopul în acest proiect am avut prilejul de a mă folosi de utilitatea acestui limbaj pentru a descrie clase și interacțiunea dintre acestea, comportament ce se propagă mai departe la obiectele propriu-zise din joc cărora le sunt atașate aceste segmente de cod. În ciuda faptului că nu am mai avut tangențe cu C# în trecut, similaritatea acestuia cu limbajul de programare Java și ușurința și naturalețea aplicării gândirii POO (Programare orientată obiect) în cadrul acestuia mi-au facilitat aclimatizarea în acest mediu nou, putând chiar să afirm că nu am întâmpinat probleme care să aibă drept cauza lipsa cunoștințelor de sintaxa și utilizare specifice limbajului.

## Firebase

Firebase este o platformă susținută de Google destinată dezvoltării de produse software ce permite utilizatorilor să creeze aplicații concepute pentru sisteme de operare precum IOS, Android sau pentru Web. Sfera de întrebuințări ale acestui proiect se extinde vast, dar în capitolele următoare voi prezenta succint funcționalitățile pe care le-am folosit eu.

### Autentificare

Prin acest serviciu, Firebase oferă dezvoltatorului posibilitatea de a pune la dispoziție în aplicația sa un sistem de autentificare sigur ce îmbunătățește experiența utilizatorului. Sunt disponibile multiple modalități de demonstrare a identității, cum ar fi, atât un sistem intern de gestionare a conturilor create prin email și parolă, cât și prin apelarea la terți precum Google, Facebook, GitHub sau Twitter.

### Bază de date în timp real

Probabil unul dintre capacitățile principale ale acestei platforme este baza de date în timp real de tipul NoSQL[[5]](#footnote-5). Aceasta este stocată pe serverele lor, ceea ce oferă proprietatea de persistență a datelor salvate într-o anumită sesiune în cadrul aplicației țintă, chiar și când aceasta este inactivă. Cu atât mai mult, toate modificările realizate asupra bazei de date sunt sincronizate în timp real, astfel devenind disponibile tuturor utilizatorilor, la câteva secunde de la încheierea tranzacției.

## Adobe Illustrator

Adobe Illustrator este un program profesional de desenare și schițare a produselor vizuale folosind tehnologie bazată pe vectori. Acesta permite crearea elementelor precum postere, simboluri, sigle, iconițe și multe altele, având complexități de la cele mai simple elemente precum forme geometrice la întregi compoziții.

Această caracteristică principală a programului, grafica vectorială, merită menționată datorită posibilităților pe care această le deblochează. Toate ustensilele disponibile în panoul principal desenează folosind linii și curbe care au la bază aceste obiecte matematice numite vectori. Spre deosebire de grafica raster[[6]](#footnote-6), unde, spre exemplu, o roată de bicicletă ar fi redată prin setarea pixelilor din matrice într-o formă circulară, grafica vectorială asociază roata cu formulă matematică a cercului, colorând punctele corespunzătoare zonei interioare a acestuia. Astfel, redimensionarea roții în primul caz ar rezulta în pierderea calității prin manipularea pixelilor, pe când în al doilea caz se poate obține orice dimensiune dorită prin ajustarea parametrilor funcțiilor matematice, fără compromisuri. Vorbind în termeni generali, tehnologia este independentă de rezoluție, deci produsele pot fi redimensionate la orice mărime și afișate pe orice dispozitiv, fără a pierde calitate, motiv pentru care a reprezentat ustensila ideală pentru a realiza componentele vizuale din cadrul jocului meu.

# Prezentarea aplicației

## Ferestre

Jocul este compus din șapte scene, dintre care una reprezintă meniul compus din cinci ferestre inter-navigabile, iar restul șase sunt asociate fiecărui nivel. Consider că interfața unui joc are un rol destul de important în experiența utilizatorului, motiv pentru care nu am neglijat acest aspect, alocând timp design-ului, atât din punct de vedere estetic, cât și funcțional.

### Schema de navigare

|  |
| --- |
| Fig. 5 – Schema de navigare a jocului |

1. Play: Afișează ecranul de selecție a nivelului
2. Leaderboard: Afișează tabela de clasament
3. Sign In: Afișează meniul de autentificare în aplicație
4. Sign Up: Afișează meniul de creare a unui cont nou
5. Play: Încarcă nivelul corespunzător selecției realizate de utilizator

6, 7, 8, 9. Back: Revine la meniul principal

### Meniul principal

|  |
| --- |
| Fig. 6 – Meniul principal |

Meniul principal (Fig. 6) este primul ecran cu care interacționează jucătorul în urma rulării executabilului. Odată încărcat, utilizatorul este întâmpinat de titlul jocului asociat cu sigla acestuia, urmate de opțiuni clare de a începe jocul, de a vizualiza tabela de clasament online, închiderea aplicației, autentificare sau crearea unui cont.

### Clasament online

|  |
| --- |
| Fig. 7 – Clasament online |

Am adăugat opțiunea unui clasament online (Fig. 7) pentru a oferi o motivație în plus de a obține o performanță cât mai bună. Odată ce un nivel este completat cu succes, timpul este încărcat în baza de date Firebase de unde este accesată de scriptul ce populează tabela. Prima pagină, intitulată “Global”, furnizează o ordonare crescătoare a sumei timpilor obținuți la toate cele 6 nivele. Navigând în continuare prin butoanele numerotate de la 1 la 6, se pot vizualiza, de asemenea, și clasamentele individuale pentru fiecare nivel.

### Mapa nivelelor

|  |
| --- |
| Fig. 8 – Meniul de selecție a nivelului |

Odată apăsat butonul “Play”, este afișat meniul de selecție a nivelului dorit (Fig. 8). Am conceput pentru acesta un design ingenios ce înglobează elemente regăsite și în cadrul jocului. La încărcarea ecranului se deschid trapele corespunzătoare tuturor nivelelor deja completate, inclusiv cel curent. Odată deschise, utilizatorul are posibilitatea de a selecta pe cel dorit.

### Autentificare / Înregistrare

|  |  |
| --- | --- |
| Fig. 9 – Meniu creare cont | Fig. 10 – Meniu autentificare |

Există 2 ipostaze în care jucătorul poate parcurge nivelele: mod anonim sau autentificat; prima varianta presupune începerea jocului fără autentificare, ceea ce conduce la lipsa înregistrării scorului sau a prezenței în tabela de clasament. Singură memorare ce are loc este progresul acumulat, reprezentat de nivelul curent necompletat, care va fi stocat local. În cel de-al doilea caz, utilizatorul beneficiază de experiența completă a jocului, ambele procese de autentificare și creare a unui cont nou fiind însoțite de mesaje ajutătoare corespunzătoare pentru situații în care sunt completate câmpurile greșit, de exemplu, parola și verificarea acesteia nu coincid, emailul nu are un format valid, absența unor informații sau absența unui cont asociat cu datele furnizate.

# Detalii de implementare

## Introducere

În cele ce urmează voi prezenta părțile componente ale jocului și modul în care interacționează toate obiectele. În procesul de creație, m-am folosit de diverse unelte pe care le pune la dispoziție Unity, dintre care cele ce merită menționate sunt cele fundamentale, care se află la baza majorității elementelor ce presupun interacțiune sau dinamism:

|  |
| --- |
| Fig. 11 – Ordinea de executare a funcțiilor |

* **MonoBehaviour** este numele clasei de baza pe care o va moșteni orice script care urmează să fie atașat unui obiect. Acesta furnizează o infrastructură pentru programarea obiectelor, ușor de înțeles și de utilizat. Pune la dispoziție funcțiile necesare pentru manevrarea proprietăților în orice stare s-ar afla instanța respectivă, având o ordine clară de executare. În Fig. 11 se observă funcțiile de inițializare, Awake() și Start() având loc la instanțierea obiectului, iar OnEnable() la activarea obiectului în scenă. Update() și FixedUpdate() se repetă la fiecare cadru, respectiv la un interval de timp constant; ele sunt utilizate în special pentru evenimentele ce țin de logica jocului, cum ar fi inputul de la utilizator, și respectiv gestionarea fizicii, cum ar fi aplicarea constantă a unei forțe. Analog inițializării, au loc funcțiile OnEnable() și OnDestroy() când obiectul și-a îndeplinit scopul și nu mai este necesar.
* **Componentele** sunt atașate obiectelor și au rolul de a gestiona diferite tipuri de interacțiuni. Printre cele mai importante pe care le-am folosit se numără, de exemplu, Rigidbody care oferă posesorului capacitatea de a se supune tuturor legilor fizicii, precum gravitația, inerția, viteza, accelerația sau momentul de rotație. De asemenea, pentru ca orice dou obiecte să poată interacționa, au nevoie de un Collider care gestionează modul în care coliziunea dintre acestea are loc. Din punct de vedere vizual, elementare sunt SpriteRenderer și Animator ce se folosesc de elemente grafice precum animații și sprite-urile[[7]](#footnote-7) din care acestea sunt compuse pentru a simula mișcare, dând astfel viată obiectelor.
* **Conceptele de programare orientată obiect** care m-au ajutat cel mai mult și pe care le-am exploatat în mod repetat sunt moștenirea si Singleton Design Pattern[[8]](#footnote-8). Astfel, am avut posibilitatea să reutilizez fragmente substanțiale de cod pentru obiecte cu comportamente asemănătoare, cum ar fi butoanele, manetele, ușile sau caracterele principale. De asemenea, piesele care se remarcă prin proprietatea de unicitate sunt atât gestionatorul bazei de date, cât și personajele principale, toate acestea fiind accesibile global în toată aplicația.

## Baza de date

Mediatorul dintre baza de date și aplicație este un script denumit FirebaseHandler, accesibil global printr-o funcție statică care returnează singură instanța a acestei clase, urmând design pattern-ul menționat în capitolul anterior. Am conceput această clasa să înglobeze în câteva funcții cu scop clar toate operațiile necesare aplicației pentru a autentifica sau crea un cont utilizatorului și pentru a salva progresul și timpii record obținuți de acesta la fiecare nivel, astfel respectând principiul de încapsulare a programării orientate obiect.

|  |
| --- |
| Fig. 12 – Diagrama structurii bazei de date |

Arhitectura bazei de date fiind de tip NoSQL, este organizată sub forma unor dosare imbricate. După cum se poate observa în Fig. 12, fiecărui cont creat în interiorul aplicației îi este asociat un nod “User” care conține informații precum un identificator unic (“localId”), generat automat de platformă la înregistrare, date personale, progresul în joc și o lista de noduri de tip “LevelData”. Acestea stochează informațiile despre fiecare nivel în parte, cum ar fi numele acestuia, indexul și timpul în care a fost completat cu succes, atât sub formă de număr rațional, care arată numărul de secunde și milisecunde, cât și structurat ca șir de caractere de forma: “minute : secunde . milisecunde”.

Un impediment pe care l-am întâmpinat încercând să realizez comunicarea între baza de date și aplicația mea este faptul că Firebase pune la dispoziție utilitare SDK[[9]](#footnote-9) ușor integrabile cu Unity doar pentru platformele Android și IOS. Pentru a rezolva această problema am importat o bibliotecă de pe Unity Store numită RestClient care facilitează trimiterea de metode HTTP[[10]](#footnote-10), astfel folosindu-mă de API-ul pus la dispoziție de Firebase pentru aplicațiile Web.

### Autentificare / Înregistrare

Pentru a realiza autentificarea și înregistrarea utilizatorului trimit comenzi de tip GET, respectiv POST către serverul Firebase, așteptând drept răspuns un cod de confirmare care poate fi 200: “OK”, sau 400: “Not supported”. Primul reprezintă confirmarea acestor două acțiuni, iar cel de-al doilea poate avea interpretări multiple, de exemplu, faptul că se încearcă accesarea unui cont inexistent sau crearea unuia deja existent, parola introdusă este greșită sau datele de intrare nu sunt valide. Odată ce comanda asociată butonului de “Sign Up” se realizează cu succes, noul cont va fi salvat și afișat în pagina denumită “Authentification” (Fig. 13), de unde se pot gestiona toate aceste intrări.

|  |
| --- |
| Fig. 13 – Meniul de gestionare a conturilor de pe pagina de autentificare din consola Firebase |

### Clasament online

La finalul fiecărui nivel am plasat un obiect denumit “NextSceneLoader”, care are atașate doar două componente, un collider și un script; cea de-a doua are rolul, pe lângă de a încarca următoarea scenă, de a salva în baza de date timpul în care jucătorul a completat nivelul. Ea ascultă prin intermediul primului atașament evenimentul de coliziune simultană cu cele două caractere pe care jucătorul le are în control, această fiind una din cerințele obligatorii pentru oprirea cronometrului. Odată urcate informațiile pe server, vor fi vizibile în panoul de gestionare a bazei de date din Fig. 14, în nodul care are drept denumire numărul de ordin al nivelului, asociat fișierului desemnat utilizatorului, de unde pot fi preluate și prelucrate în timp real pentru a alcătui clasamentul online. Acest eveniment are loc doar în cazul în care jucătorul este autentificat, altfel doar progresul curent va fi salvat local în memoria aplicației. Pentru a fi posibilă și securizată scrierea, Firebase dispune și de un sistem de gestionare a permisiunilor de a realiza o anumită acțiune. În contextul aplicației mele consider că setarea compatibilă este cea în care citirea poate fi realizată de către oricine, pentru a putea consulta clasamentul în orice moment, dar am configurat ca scrierea să fie posibilă doar de către un jucător autentificat, pentru a preveni modificarea neautorizată a datelor altui utilizator.

|  |
| --- |
| Fig. 14 – Panoul de administrare a bazei de date în cadrul consolei Firebase |

O provocare des întâlnită în comunicarea cu baze de date este potențialul timp îndelungat de așteptare pe care îl poate genera această interacțiune; soluția se regăsește într-o latura interesantă a limbajelor precum C#, programarea asincronă. Ea presupune rularea în paralel a unor secvențe de cod, separat de firul principal de execuție, având control asupra progresului acesteia. Beneficiile ce derivă sunt îmbunătățirea performanței și a timpului de răspuns a aplicației, fiind cel mai ușor observabile la nivelul interfeței cu care interacționează utilizatorul deoarece devine mereu funcțională și interactivă, eliminând posibilitatea ca aceasta să se blocheze pentru a aștepta terminarea unei anumite secvențe de instrucțiuni. Se poate observa în anexa 8.1că toate interogările realizate prin librăria RestClient sunt de tip asincron, codul introdus în blocul Then() fiind executat odată ce răspunsul de la serverul Firebase este recepționat. O problema pe care am întâmpinat-o datorită acestui comportament este faptul că nu aveam posibilitatea de a obține o lista cu informații despre utilizator apelând o funcție publică din exteriorul clasei, deoarece aceasta ajungea la instrucțiunea de return, înainte de a executa codul din blocul Then(). În urma unui proces de cercetare, am descoperit posibilitatea de a accepta ca parametru pentru o metodă un obiect de tip Action<T>, prin care să furnizez la apel o funcție anonimă care să fie executată odată ce informațiile necesare au fost extrase din baza de date. Astfel, clasa denumită Leaderboard din anexa 8.2 poate realiza toate operațiile necesare populării bazei de date prin secvența de instrucțiuni pasată funcției ce le obține. Pentru o analiză mai detaliată a scorurilor, pe lângă clasamentul global (Fig. 15), am pus la dispoziție și pagini asociate fiecărui nivel, cu timpii respectivi sortați în ordine crescătoare (Fig. 16).

|  |  |
| --- | --- |
| Fig. 15 – Clasament global | Fig. 16 – Clasament specific nivelului 1 |

## Mediul

Interacțiunea personajelor cu mediul constă în activarea și dezactivarea a anumitor elemente dinamice cum ar fi butoane și manete, care sunt asociate în mod intuitiv prin aceeași culoare cu obiectele precum lasere, platforme mișcătoare și uși. Deși au scopuri și proprietăți diferite, ele se aseamănă prin faptul că toate realizează o mișcare în momentul în care primesc un semnal. După cum se poate observa în diagramă de clase UML din Fig. 17, toate moștenesc clasa Trigger care definește metoda TriggerFunction() ce inițiază respectiva mișcare.

|  |
| --- |
| Fig. 17 – Diagrama UML de interacțiune între obiectele cu care poate interacționa jucătorul |

### Buton și maneta

Butonul și maneta sunt elementele cu care jucătorul interacționează direct prin coliziune. Fiecare stochează în lista de obiecte de tip Trigger referințe către una sau mai multe instanțe cărora le va apela funcția de activare în momentul în care propria lor funcție de activare este apelată. Prin arhitectura pe care am conceput-o am pus în valoare regula de substituție Liskov din suita de principii SOLID[[11]](#footnote-11) ale programării orientate obiect, care presupune posibilitatea de a folosi orice clasă în locul unei clase părinte a acesteia, păstrând aceeași funcționalitate fără a implica modificări adiționale. Astfel, procesul de proiectare a nivelelor mi-a fost ușurat substanțial odată ce am creat câte un prefab[[12]](#footnote-12) pentru fiecare dintre aceste obiecte, urmând ca apoi să stabilesc perechi de tipul “element\_declansator – element\_declansat” în care pot asigna declanșatorului orice tip și oricâți moștenitori ai clasei Trigger, cum ar fi uși ce se deschid sau închid, platforme ce se deplasează între două puncte fixe sau lasere care se activează și dezactivează. Ambele parcurg lista denumită “triggers” și apelează funcția TriggerFunction() pentru fiecare în parte, dar funcționalitatea ce le deosebește este una subtilă, dar care amplifică posibilitățile de proiectare a puzzle-urilor; butonul (Fig. 18) acționează perechea sa, atât la apăsare, cât și la eliberare, cu posibilitatea de a avea și un temporizator ce amână ridicarea sa din momentul în care jucătorul nu mai este în contact cu el, în timp ce maneta (Fig. 19) se poate schimba între pozițiile pornit sau oprit, asemănător unui comutator.

|  |  |
| --- | --- |
| Fig. 18 - Butoane | Fig. 19 - Manete |

### Laser

Laserul presupune un pretext ingenios și estetic de a adauga încă o restricție, un impediment în procesul de rezolvare a puzzle-ului. Asemănător conceptului din Portal 2 prezentat în unul din capitolele anterioare, l-am realizat din trei părți componente ilustrate în Fig. 20: un obiect ce are atașată o componentă de tip LineRenderer[[13]](#footnote-13) și alte două identice, oglindite, unul ce joacă rol de emițător, iar celălalt de receptor.

|  |
| --- |
| Fig. 20 – Laserul compus din emițător, LineRenderer, respectiv receptor |

Am atașat obiectului și un collider pentru a putea înregistra momentul în care unul dintre caracterele controlabile de jucător intră în contact cu acesta, cu scopul de a declanșa resetarea scenei, utilizatorul fiind prevenit de consecințele greșelii. Întrucât am dorit să am posibilitatea de a plasa emițătorul sau receptorul pe obiecte mișcătoare, m-am lovit de problema că marginile ce alcătuiesc colliderul atașat laserului sunt generate de Unity la instantierea obiectului, rămânând constante pe parcursul rulării. Drept consecință, deși din punct de vedere vizual linia se extindea și se contracta corespunzător mișcării platformei de care era atașat, limitele nu se actualizau. Soluția pe care am găsit-o a fost de a atașa scriptul denumit LazerCollisionController din diagrama UML plasată la începutul capitolului, prin intermediul căruia restabilesc dinamic, la fiecare cadru, cele patru puncte ale paralelogramului.

### Platforma mișcătoare

Platformele mișcătoare reprezintă o modalitate de transport a personajelor controlate de jucător, în special cel terestru, care permite ajungerea în anumite zone inițial inaccesibile, printr-o mișcare rectilinie, previzibilă ce se bazează pe funcția Lerp() din biblioteca Vector3. Aceasta are drept fundament matematic conceptul de interpolare liniară între două puncte care determina poziția unui al treilea punct, date fiind primele două și un procent de distanțare față de primul, plecând de la presupunerea că există un segment de dreapta ce le intersectează pe toate trei:

Considerând în formula anterioară variabilele x, a și b vectori, prin varierea valorii lui t, un scalar aflat în intervalul [0,1], putem simula glisarea lui x, pornind de la punctul inițial a, pentru care t = 0, până la destinația b, pentru care t = 1. Pentru a obține o mișcare cursivă, inițial accelerată și decelerată spre final, apelez metoda CalculatePlatformMovement() în funcția FixedUpdate() pentru a afla noua poziție la care trebuie să translez obiectul la fiecare interval fix de timp, observabil în anexa 8.3. După cum am menționat și în capitoul introductiv, am optat pentru această funcție a clasei MonoBehaviour, în locul celeilalte variante, Update(), pentru a mă asigura că mișcarea este una constantă, independenta de numărul de cadre pe secundă, deoarece acesta poate fi influențat de mai mulți factori, cum ar fi specificațiile tehnice și puterea de procesare a componentelor sistemului pe care rulează utilizatorul jocul. Particularizând formula anterioară în situația mea, poziția curentă a platformei la momentul apelului joacă rolul variabilei a, iar destinația este capătul traiectoriei, setat în inspector. Proprietățile mișcării sunt redate de modul de calculare a procentului t: îl setez cu valoarea 0 la momentul inițierii deplasării, iar la fiecare iterație adun la valoarea anterioară diferența de timp dintre apelul curent și cel precedent, înmulțit cu viteza pe care o doresc, totul divizat la distanță rămasă până la punctul final. Odată aflată valoarea lui t, introdusă în funcția Vector3.Lerp() împreună cu plasarea curentă și destinația, obțin o mișcare plăcută din punctul de vedere al experienței de joc deoarece are viteză maximă in jurul mijlocului distanței de deplasare, iar spre capete prezintă o accelerare ușoară dintr-o poziție staționară, respectiv o decelerare lentă până la oprire, ceea ce permite îmbarcarea și părăsirea relaxată a mijlocului de transport.

|  |
| --- |
| Fig. 21 – Reprezentarea mișcării orizontale a platformei |

### Ușa

Ușile ilustrate în Fig. 22 sunt o altă modalitate prin care m-am gândit să restricționez posibilitățile de mișcare ale jucătorului, având situații când au o orientare atât verticală, cât și orizontală. Motricitatea lor este asemănătoare cu cea a platformelor, cu diferența că, spre deosebire de acestea, parametrul modificabil în inspector reglează durata de timp necesară pentru închiderea sau deschiderea completă în loc de viteză, pentru a spori gradul de ajustabilitate, în combinație cu eventualul temporizator al butoanelor.

|  |
| --- |
| Fig. 22 - Uși |

## Caractere

Mecanica de bază a jocului constă în controlarea alternativă a două caractere, dintre care unul este terestru, reprezentat de un ornitorinc agent secret, intitulat Perry, care are posibilitatea de a păși pe platformele din spațiul bidimensional, de a sări sau ataca. Drona poate zbura, fiind un personaj adjuvant care desemnează un ecran ce afișează în timp real un apel video cu Maiorul Monogramă, șeful agenției secrete. Întrucât au relativ multe funcționalități comune, cum ar fi funcțiile de ascultare a input-ului, aplicarea forțelor pentru mișcare sau preluarea/pasarea controlului, clasele ce controlează sprite-urile moștenesc PlayerController (Fig. 23), având astfel din nou avantajul de a mă putea referi la oricare dintre ei în interacțiunea cu alte obiecte prin acest părinte. De asemenea, având în vedere proprietatea de unicitate pe care o au amândoi în contextul jocului, am implementat clasele urmând Singleton Design Pattern ceea ce mi-a facilitat procesul de creație a nivelelor, astfel încât pot accesa din orice loc instanțele lor prin intermediul funcției statice GetInstance(), fără să mai fiu nevoit să asignez manual o referință fiecărui element în parte care interacționează în scenă cu ei. Un exemplu relevant de astfel de situație este în cazul gestionatorului de control, care are simpla întrebuințare de a asculta comanda de comutare a personajelor lansată de jucător prin tasta Shift, urmând ca apoi să dezactiveze și să activeze clasele corespunzătoare ce moștenesc PlayerController, astfel încât exact una dintre ele este întotdeauna activă la un moment dat.

|  |
| --- |
| Fig. 23 – Diagrama de clase UML a caracterelor controlate de jucător |

### Perry

Perry este caracterul principal asupra căruia jucătorul are controlul la începutul fiecărui nivel. Având în vedere că am conceput nivelele să aibă o orientare verticală, începutul fiind mereu în partea de jos a ecranului, iar finalul în partea de sus, scopul principal este întotdeauna crearea unui traseu prin care el să poată ajunge la platforma finală, din moment ce drona o poate accesa mult mai ușor, având capacitatea de a zbura. Obiectul are atașat scriptul GroundPlayerController din Fig. 23 care se ocupă de interacțiunea cu mediul și mișcarea acestuia în funcție de comenzile trimise de utilizator. Prin atributele sale pot gestiona viteza de deplasare, numărul de sărituri consecutive, puterea saltului sau punctele de viață.

Animațiile, efectele speciale și aspectul unui joc au un rol important în experiența utilizatorului și nu pot fi neglijate, altfel poate apărea confuzia sau lipsa de interes, ceea ce poate duce mai departe la abandonare. Acestea fiind spuse, am acceptat provocarea de a experimenta și acest capitol al dezvoltării jocurilor video și am realizat personal aproape toate componentele vizuale și dinamice în Adobe Illustrator. Exceptând dezavantajul că este extrem de costisitor din punct de vedere temporal, consider că am avut multe de câștigat întrucât necesită un nivel de atenție la detalii sporit, cum ar fi potrivirea culorilor sau a stilului de desenare, dimensiunea obiectelor relativă una la cealaltă și setările de importare în Unity sau alinierea perfectă a cadrelor animațiilor din Fig. 24.

|  |
| --- |
| Fig. 24 – Alinierea imaginilor pentru animația de atac |

Unity dispune de un sistem de controlare a animațiilor extrem de intuitiv; clasa GroundPlayerController conține o referință către o componentă de tip Animator atașată obiectului prin intermediul căreia pot trimite din cod semnale către un gestionator de tipul Animation Controller[[14]](#footnote-14), în care am definit animațiile și condițiile necesare pentru a face o tranziție între ele.

Schema din Fig. 25 arată toate conexiunile pe care le-am realizat între toate stările în care se poate afla Perry. În mod implicit, fără stimuli externi, el va rula la infinit animația “idle”; o tranziție din starea generală “Any state”, generată automat și obligatoriu de Unity, se poate realiza în orice moment dacă este îndeplinită condiția de tranziție. Spre exemplu, în această figura am selectat trecerea spre animația de atac, care se poate observa în fereastra Inspector din dreapta, ce are loc doar în momentul în care variabila isAttacking are valoarea adevărat; în momentul în care jucătorul apasă click, prin simplul apel al unei funcții o setez, iar Unity se ocupă de afișarea animației, prin schimbarea sprite-ului din componenta SpriteRenderer2D, la fiecare cadru.

|  |
| --- |
| Fig. 25 – Schema de tranziție intre animațiile lui Perry |

### Maiorul Monogramă

Conceptul care m-a fascinat cel mai mult la jocul FireBoy & WaterGirl este ideea de cooperare între două entități controlate de o singură minte care are o privire de ansamblu asupra întregii camere ce reprezintă nivelul curent. Având acest punct de plecare, am eliminat din ideea originală controlul lor simultan și acele acumulări de lavă sau apă care permit sau nu trecerea și am adus o notă personală prin crearea unei drone ce poate zbura, în loc de două personaje identice. Mecanica de zbor am obținut-o prin simpla eliminare a efectului gravitației asupra acesteia și aplicând o forță în direcția de deplasare introdusă de jucător. Pentru a simula mișcarea de frânare spre oprire, am mărit factorul forței de frecare cu aerul care încetinește progresiv corpul în momentul în care nu mai sunt forțe aplicate asupra sa.

În general, coliziunea între caractere este dezactivată, pentru a nu exista situații în care s-

|  |
| --- |
| Fig. 26 – Perry stând pe drona |

ar putea încurca unul pe celălalt. Singura ipostază în care ele pot interacționa este când Perry se află deasupra dronei ca în Fig. 26, astfel având posibilitatea de a staționa pe ea, făcând accesibile animalului terestru anumite zone inițial blocate datorită înălțimii. Am reușit să implementez acest comportament cu ajutorul componentei PlatformEffector2D atașate unui obiect copil dronei, ce se ocupă în mod special de restricționarea posibilității de coliziune doar pentru anumite grupuri de obiecte, doar din anumite unghiuri de incidența. O problema interesantă pe care am întâmpinat-o este faptul că ambele fiind obiecte dinamice, cu masă, care se supun regulilor fizicii simulate de Unity, în momentul coliziunii cădeau spre pământ datorită gravitației aplicate asupra lui Perry. Soluția găsită a fost să setez corpul dronei în componenta RigidBody2D de tip “kinematic” pe durata coliziunii, ceea ce dezactivează efectul întregului modul de simulare a fizicii lumii reale asupra sa. Având în vedere că această setare se întâmplă prea târziu în momentul înregistrării coliziunii, iar asupra obiectul zburător avea loc o translație semnificativă până să intre în vigoare modificarea, am atașat clasa POnPCollisionController din diagrama UML. Rolul ei este de a prezice și realiza configurațiile necesare înainte de contact, folosind funcția Physics2D.OverlapCircle() care, având date că parametrii o poziție în scenă, o rază de acțiune și un obiect de tip LayerMask[[15]](#footnote-15), returnează o valoare booleana care informează dacă există o suprapunere între cercul definit de primii doi parametrii și orice obiect aparținând stratului definit de cel de-al treilea. Așadar, apelând metoda la fiecare cadru și verificând dacă direcția de mișcare a lui Perry are componenta y negativă, semnificând o cădere, în momentul tangenței cu un cerc aflat deasupra dronei pot anticipa coliziunea celor două.

În marea majoritate a timpului, cele două personaje principale vor avea sarcini diferite, în zone diferite ale nivelului. Cu toate acestea, sunt momente în care se deplasează pe același drum, în aceeași direcție, iar datorită controlului alternativ, fiecare dintre ele trebuie condusă pe rând în punctul respectiv. Pentru a eficientiza procesul, am utilizat o biblioteca intitulată AStarPathfindingProject ce simulează și implementează algoritmul de căutare A\*. Acesta este des folosit în jocuri în crearea inteligenței artificiale, datorită eficienței sale de furnizare a celui mai scurt drum, dat fiind un graf pentru care există cel puțin un lanț de la punctul de plecare până la destinație. Specific acestui algoritm de căutare este faptul că deși la baza sa este un algoritm de tip căutare în lățime, frontiera acestuia nu se extinde în mod consistent în toate direcțiile, ci prioritizează căile care par să ducă mai aproape de locul dorit. Face acest lucru folosind un sistem de alegere a următorului nod ce combină alegerea drumului cu cost minim cu o euristică ce estimează costul lanțului ce urmează a fi descoperit din punctul curent până la destinație. Odată ce am atașat clasa numită Pathfinder unui obiect plasat în scenă, proiectul îmi generează automat un graf alcătuit din noduri interconectate ce înglobează toate drumurile posibile între toate zonele accesibile, odată ce am setat un obiect de tip LayerMask prin care specific care sunt obstacolele. În Fig. 27 se poate observă cum platformele, ușile, manetele, bila și cutia sunt ocolite, iar în zona albastră este calculat cel mai scurt drum de la drona la Perry; odată aflată această cale, clasa FollowController atașată obiectului zburător aplică forța necesară deplasării de la un nod la altul, până ajunge la destinație.

|  |
| --- |
| Fig. 27 – Ilustrarea celui mai scurt drum calculat cu ajutorul bibliotecii AStarPathfindingProject |

### Inamic

O problemă comună ce poate apărea în contextul jocurilor de tip puzzle este lipsa elementelor dinamice. Având în vedere că se pune accentul de planificare și gândirea în avans a mișcărilor, este posibil ca scena să devină destul de statică, ceea ce poate duce la pierderea interesului. Pentru a combate acest aspect, am adăugat un inamic cu care Perry va fi nevoit să susțină o luptă, pentru a primi cea de-a treia pălărie colectabilă, necesară avansării la nivelul următor. Inamicul simulează un comportament ghidat de o inteligență artificială. După cum se poate observa în Fig. 28, este compus din mai multe stări pe care le accesează în funcție de proximitatea personajelor principale, sau în funcție de mediul înconjurător. Clasa StateMachine gestionează tranzițiile între stări, declanșate pe baza unor evenimente înregistrate de implementările concrete ale ipostazelor: E1\_IdleState, E1\_MoveState, E1\_AttackState, etc. Această arhitectură oferă o fluiditate și naturalețe comportamentului inamicului, stările fiind clar ilustrate prin animații sugestive, obținute de pe Unity Asset Store. Inițial patrulează platforma pe care se află, întorcându-se dacă identifică o margine, un perete sau orice alt obstacol. Dacă jucătorul intră în raza lui vizuală, și rămâne pentru o perioadă scurtă de timp, va fugi spre el până se află la distanța minimă necesară pentru a ataca. Când jucătorul părăsește câmpul sau de acțiune, se va întoarce de 3 ori pentru a-l cauta, revenind la patrulă dacă nu îl găsește. Un avantaj extrem de benefic al acestei structuri de clase este faptul că respectă principul deschis/închis al programării orientate obiect deoarece se poate extinde ușor comportamentul inamicului adăugând stări noi, fără a modifica pe cele deja existente. Odată implementată logica circumstanței respective in funcțiile suprascrise LogicUpdate(), PhysicsUpate(), Start() și Exit() pe care le posedă orice stare, StateMachine se va ocupa de tranziția între ele.

|  |
| --- |
| Fig. 28 – Diagrama de clase UML a inamicului |

# Analiza nivelelor

Cea mai plăcută și interesantă parte din toate etapele dezvoltării jocului a fost proiectarea nivelelor. Odată ce am creat toate obiectele prefabricate necesare, mi-am configurat mediul de lucru astfel încât unele nivele au putut fi construite doar prin acțiuni de tip drag&drop, fără a mai fi nevoie să scriu linii de cod. Divizarea muncii în această manieră mi-a permis să-mi aloc toată concentrarea în a găsi echilibrul perfect între o experiență de joc cursivă, cu obiective, reguli și mijloace puse la dispoziție clare, dar și o provocare pentru jucător de depășire a limitelor proprii, de a reuși să elibereze un drum spre ieșire într-un mediu inițial necunoscut și intimidant. În următoarele subcapitole analizez fiecare nivel în parte pentru a prezenta soluțiile acestora, scoțând în evidență capcanele pe care le-am conceput si modul în care elementele mediului înconjurător se îmbină în mod strategic, construind un labirint, cu una sau mai multe căi de scăpare.

## Nivelul 1

Nivelul 1 este conceput pentru a introduce jucătorul în contextul aplicației și pentru a-i oferi un prim contact cu elementele cu care va avea de interacționat. Am ilustrat în Fig. 28, în ordinea apariției, toate lucrurile esențiale pe care le află, cum ar fi toate controalele de care are nevoie, posibilitatea de a sări pe drona, modul de acționare a manetelor și a butoanelor cu temporizator, obligativitatea adunării tuturor obiectelor colectabile și modul în care poate profita de mediu, mutând bile și cutii. Pentru a transmite aceste indicații într-o manieră tematică, am adăugat în partea de jos un bloc de text alăturat unei poze cu Maiorul Monogramă, ce simulează din punct de vedere vizual o notificare primită pe un dispozitiv de comunicare oarecare. Textul afișat se schimbă în funcție de poziția în nivel la care se află Perry, de fiecare dată oferind informații relevante pentru următoarele mișcări pe care jucătorul le are de făcut.

|  |
| --- |
| Fig. 29 – Analiza nivelului 1 |

### Nivelul 2

La nivelul 2 jucătorul poate face o primă observație, faptul că în partea din stânga se află o pălărie într-o zonă inaccesibilă în faza incipientă. Este un indiciu al faptului că trebuie să ajungă la cutia corespunzătoare punctului 2 din Fig. 29 și să o plaseze pe butonul roșu, care va deschide calea către acea pălărie. Pentru a putea ajunge acolo, va avea nevoie de ajutorul dronei care va activa butonul de la marcajul 1, în timp ce Perry se va afla deja pe platforma mișcătoare. Maneta de la punctul 3 deblochează ușa ce eliberează drumul spre bilă, dar în același timp blochează calea obiectului rotativ spre butonul de la mișcarea cu numărul 5. Din nou, acest impas poate fi depășit doar prin colaborare, Perry având rolul de a comuta manetă de oricâte ori este nevoie, în timp ce drona împinge bila spre canalul ce trebuie să fie liber, altfel greșeala este pedepsită prin pierderea bilei într-o fundătură, ceea ce duce la resetarea nivelului. După ce drona conduce mingea de la punctul 5 spre buton, ultima provocare este de a reuși să parcurgă drumul de la butonul albastru la poziția indexată cu numărul 6, cât timp temporizatorul se scurge, pentru a oferi suport animalului terestru pentru a putea ajunge în zona stâng superioară a nivelului, orice greșeală rezultând în coliziunea cu laserul, ceea ce provoacă resetarea scenei.

|  |
| --- |
| Fig. 30 – Analiza nivelului 2 |

### Nivelul 3

Nivelul 3 necesită un spirit de observație sporit; primul lucru în contradictoriu cu așteptările jucătorului este faptul că toate manetele, exceptând una, se vor reseta în poziția inițială la mai puțin de o secundă după atingere. În acest moment, jucătorul este pus în situația de a cauta indicii, primul lucru care iese în evidență fiind mulțimea de săgeți din zona marcată cu 0 în Fig. 30. Poziția fiecărui tablou are o valoare a componentei verticale unică, ceea ce este un indiciu că manetele trebuie acționate în ordinea redată de înălțimea săgeților, de la înalt la scund, orice abatere de la regulă rezultând în resetarea tuturor ușilor. În următoarea secțiune, după acționarea manetei de la punctul 1 și înfrângerea inamicului, Perry este îndemnat să se așeze pe butonul de la punctul 4 deoarece am plasat o pălărie acolo; este defapt o capcană, drona fiind cea care trebuie să dezactiveze laserul pentru a elibera calea lui Perry spre zona 5. Obiectul zburător se va întoarce pe același drum pe care a venit deoarece laserul se reactivează odată ce părăsește butonul și joacă din nou rol de adjuvant în accesarea zonelor inaccesibile lui Perry in pozițiile numerotate cu cifra 6.

|  |
| --- |
| Fig. 31 – Analiza nivelului 3 |

### Nivelul 4

Nivelul 4 necesită ingeniozitate și o capacitate de a înțelege proprietățile fizice ale obiectelor cu scopul de a ajusta corespunzător forța aplicată asupra obiectelor pentru a obține rezultatul dorit. La început, jucătorul se regăsește într-o scenă aparent simplă, pentru care prima mișcare intuitivă, fără planificare în avans, este de a activa unul din cele două declanșatoare portocalii din Fig. 31.Făcând asta, va realiza că orice variantă de activare a platformei alege, va exista un element blocant ce nu permite avansarea spre punctul 5:

·       Dacă alege să plaseze drona pe buton, va realiza că Perry nu poate sări spre punctul 5 fără ajutor, iar orice mișcare a Maiorului Monogramă va conduce la întoarcerea platformei în poziția inițială;

·       Dacă alege să comute maneta, platforma va rămâne, de dată aceasta, în poziția favorabilă, dar drona va fi blocată în spațiul închis marcat cu cifra 2, deci nu va putea oferi suportul necesar pentru a realiza săritura;

Soluția constă în exploatarea modului de funcționare a declanșatoarelor; există posibilitatea de a inversa pozițiile platformei relativ la poziția manetei: Perry se poziționează pe butonul de la punctul 1 după ce permite accesul dronei în zona 2 acționând butonul albastru. Odată ce zburătorul comută maneta, când Perry va elibera butonul, va provoca defazarea platformei și astfel vor fi amândoi liberi și platforma staționată în partea dreapta. Mai rămâne doar că drona să aducă platforma acționând butonul, pentru ca în continuare să se realizeze saltul asistat la pozițiile numerotate cu 3. În continuare, o decizie trebuie luată: pe ce direcție trebuie să meargă cutia și bila, amândouă putând lua ambele traiectorii în funcție de forța aplicată asupra lor, dar traseul mai lung ar trebui să fie un indiciu că acela este cel destinat obiectului rotativ. Butonul de la punctul 5 are un temporizator, suficient pentru a putea urca cu Perry și a împinge cutia, până la redeschiderea ușii albastre. Aceasta va închide trapă roșie, completând traseul mingii către butonul verde.

|  |
| --- |
| Fig. 32 – Analiza nivelului 4 |

### Nivelul 5

La nivelul 5 jucătorul poate fi indus în eroare de simetria modului în care sunt dispuse platformele fixe. Primul pas este de a coopera cu scopul de ajunge Perry în partea superioară, unde are sarcina de a împinge mingea de la punctul 2 spre butonul albastru; această acțiune este una contraintuitivă deoarece jucătorul s-ar aștepta ca obiectele mici să fie împinse de drona, dar judecând după experiența nivelelor anterioare, are posibilitatea de a prezice că Maiorul Monogramă nu va avea destulă putere. Faptul că nivelul este aproape oglindit pe direcția verticală ar putea sugera utilizatorului faptul că ambele butoane blocate de câte o trapă trebuie acționate cu o bilă. Dar, după cum se observă în Fig. 32, la poziția 4 butonul este apăsat de drona, după ce Perry eliberează calea la punctul 3. Acum, drumul este liber pentru ca animalul terestru să rostogolească bila peste butonul albastru.

|  |
| --- |
| Fig. 33 – Analiza nivelului 5 |

### Nivelul 6

Provocarea finală are dificultatea cea mai mare deoarece necesită o planificare în avans și o sincronizare a mișcărilor destul de precisă. Cu atât mai mult, aspectul nivelului este unul intimidant la prima vedere având multe mingi care pot fi direcționate pe diverse canale intersectate. Soluția este clar descrisă în Fig. 33, la care vin cu precizarea că orice greșeală poate duce la blocarea sau pierderea unei mingi, ceea ce este un factor motivant pentru jucător de a gândi mișcările în avans. De exemplu, este absolut necesar că mișcarea 4 să aibă loc cu ușa portocalie în poziția opusă față de cea surprinsă în cadru, deoarece bila 5 va acționa butonul turcoaz ce are un timp de dezactivare de 1.5 secunde; cât timp aceasta se află în cădere, Perry are sarcina de a activa maneta de la punctul 1, sincronizând căderea mingii cu intervalul de 1.5 secunde cât va fi deschisă trapa turcoaz. De asemenea, pentru cazul ultimului obiect rotativ este necesară acumularea unei inerții mai mari decât se generează dacă ar fi împinsă direct, motiv pentru care va fi rostogolită înapoi, apoi aplicată o forță continuă asupra ei pentru a putea urma calea ilustrată de marcajul 6.

|  |
| --- |
| Fig. 34 – Analiza nivelului 6 |

# Concluzii finale

## Studiu

Procedeul de dezvoltare a unui joc implică o multitudine de etape de testare, ședințe de dezbatere a ideilor și modificări realizate pe baza opiniilor oamenilor. Așadar, procesul de creare a unei astfel de aplicații, care prin definiție este un produs având drept grup țintă o mulțime considerabilă de oameni, din toate categoriile sociale, presupune o dificultate pentru dezvoltator să cunoască și să întâlnească așteptările, preferințele și plăcerile tuturor, fără a exista o comunicare între aceștia pe parcurs. De asemenea, este crucial ca în calitate de creator, să abordez o mentalitate flexibilă relativă la produsul meu, eliminând latura emoțională, pentru a avea capacitatea de a privi opiniile exterioare într-o manieră obiectivă, sugestiile negative fiind filtrate astfel în unele constructive. Un eveniment ce m-a determinat să obțin această performanță este faptul că în procesul de proiectare a nivelelor, in urma conversației cu o persoană, am ajuns la concluzia că eram incapabil de a estima personal dificultatea acestora; era imposibil să privesc soluțiile dintr-o perspectiva exterioară, odată ce eu cunoșteam toate etapele, modificările și testele ce au avut loc până să ajung la starea finală și funcțională a puzzle-ului. Acestea fiind spuse, am distribuit executabilul jocului mai multor persoane, având atașat un formular cu câteva întrebări menite atât să estimeze gradul de îndeplinire obiectivelor proiectului, cât și să îmi ofere o privire de ansamblu asupra elementelor ce necesită îmbunătățiri sau modificări.

|  |
| --- |
| Fig. 35 – Dificultatea nivelelor apreciată de jucători |

Formularul conține aceleași două întrebări pentru fiecare nivel în parte. Prima este o simplă estimare a dificultății redată pe o scară de la 1 la 5. Motivul pentru care am realizat această statistică este deoarece consider că jocurile de tip puzzle necesită o evoluție graduală a complexității, pentru a oferi timp utilizatorului de a se obișnui cu mecanicele și modul de gândire necesar pentru rezolvarea acestora. În Fig. 34 am centralizat răspunsurile pentru toate cele șase nivele, de unde se poate observa clar că se formează o tendința crescătoare a cantității de efort depusă pentru deslușirea puzzle-urilor.

|  |
| --- |
| Fig. 36 – Evaluarea experienței de joc a fiecărui nivel in parte |

Cea de-a doua interogare este referitoare la impresia generală creată jucătorului de către fiecare nivel. În Fig. 35 este surprinsă satisfacția pe care au simțit-o jucătorii, măsurată pe o scară relativ subiectivă, de la un sentiment complet negativ la o impresie plăcută și plină de entuziasm, variantele îmbunătățindu-se gradual în cinci trepte. Se poate observa că opinia globală este una pozitivă, întrucât rezultatele variază de la o părere cu conotație neutră spre o apreciere deplină. Concluzia pe care am extras-o din aceste procente este că munca mea nu a fost futilă și am reușit să construiesc un joc funcțional cu destule elemente atrăgătoare pentru a oferi o experiență plăcută, dar, că orice proiect atât de complex aflat în stare incipientă, mai necesită multe îmbunătățiri și rectificări.

Un alt aspect relevant în contextul aplicației pe care l-am avut în vedere este echilibrul. Pe de-o parte, am vrut să conțină puzzle-uri accesibile pentru marea majoritate a oamenilor, având la dispoziție într-o manieră intuitivă toate mijloacele necesare pentru găsirea soluției. In același timp, am vizat să includ și un factor provocator, care să pună in dificultate și să necesite un proces de gândire mai complex, fapt ce aduce și un plus de satisfacție în momentul realizării unei descoperiri. Din experiența personală pot afirma că o dovadă a existenței acestui echilibru este momentul în care un joc oferă un sentiment de bucurie spontană, al descoperirii, de claritate în urma confruntării cu o situație dificilă, deseori asociat cu exclamația lui Arhimede: “Evirca!”, care se traduce prin sintagma “Am aflat!”. În urma verificării rezultatelor, am avut plăcuta surpriză de a fi întâmpinat de un procent extrem de satisfăcător al persoanelor care au relatat că au experimentat acest sentiment, conform Fig. 36. De asemenea, un aspect asupra căruia am insistat și am triumfat în realizarea sa a fost de a găsi un mod în care jucătorul să afle o informație nouă si relevantă din greșelile pe care le face, care duc la resetarea scenei, dar de data aceasta cu un indiciu în plus.

|  |
| --- |
| Fig. 37 – Măsura in care jucătorii au avut un moment revelator sau au învățat din greșelile făcute |

Din fericire, un alt avantaj extrem de util din punctul de vedere al unui dezvoltator este eficacitatea procesului de testare; este mult mai probabil ca un număr substanțial de persoane, cu mentalități și moduri de gândire diferite, să abordeze jocul în modalități diferite, realizând diverse combinații de mișcări care pot scoate la iveală anumite erori de implementare. Special conceput pentru această situație, am adăugat în formular și un câmp pentru raportarea acestor probleme întâlnite, din care am avut posibilitatea de a extrage foarte multe informații prețioase, care m-au ajutat să consolidez codul și funcționalitatea aplicației mele.

## Direcții de viitor

În opinia mea, singura limitare a posibilităților unui joc este conturată de imaginația creatorului și de tehnologia disponibilă. Având în vedere că prima este o resursă infinită a creierului uman, iar cea de-a două se află în continuă expansiune, jocul meu are o multitudine de direcții în care poate evolua mai departe. Printre acestea, câteva pe care le consider eu mai relevante ar fi următoarele:

·       Introducerea de elemente interactive noi, cu scopul de a mari posibilitățile de creare a puzzle-urilor, cum ar fi portaluri, oglinzi, turbine de vânt sau suprafețe alunecoase.

·       Integrarea unor mini-jocuri în compoziția nivelelor, care să joace rol de sarcini ce trebuie rezolvate pentru a putea progresa, ceea ce aduce posibilități nenumărate: sudoku, sortare de obiecte, un labirint, ghicitori, realizarea unor conexiuni, construirea de obiecte și multe altele.

·       După cum am precizat în capitolele anterioare, odată ce aveam la dispoziție toate elementele necesare proiectării nivelelor, acțiunea propriu-zisă de construire presupunea doar plasarea obiectelor în scenă și interconectarea lor. O idee extrem de interesantă ce a reieșit din această realizare ar fi adăugarea unui modul care permite utilizatorului să își asume rolul de dezvoltator și să creeze el însuși nivele, într-un mediu de lucru intuitiv și ușor de folosit. Personal, pot confirma faptul că acțiunea de a conturi puzzle-uri este cel puțin la fel de captivantă si provocatoare ca rezolvarea lor, iar posibilitatea de a oferi spre testare celorlalți jucători mapele create, eventual și introducerea unui element competitiv bazat pe votare, ar putea crește substanțial valoarea aplicației.

## Concluzie

În capitolul introductiv am afirmat faptul că în urma lecturării experimentului relativ la potențialul jocurilor video de tip puzzle de a antrena unele funcții executive ale creierului uman, am propus pentru această lucrare să studiez dacă există posibilitatea de a crea un joc specific conceput pentru această cauza. Consider că am făcut un pas în direcția corectă, cu toate că mă aflu doar la început de drum, existând încă destul loc pentru aprofundare și experimentare mai amănunțită. Având în vedere că nu am dispus de resursele și cunoștințele necesare pentru a organiza un studiu care să poată returna date concrete și valide referitoare la reușita atingerii scopului propus, am apelat la simpla opinie a persoanelor ce au avut răbdarea de a juca jocul. În statistica ilustrată în Fig. 36 este rezumată părerea oamenilor referitoare la posibilitatea ca jocul “Where’s Perry?” să aibă un impact pozitiv, în orice măsură, asupra funcțiilor executive enumerate la baza fiecărei coloane. Din punctul meu de vedere, concluzia ce poate fi trasă analizând această reprezentare grafică are o conotație destul de pozitivă, având astfel plăcerea de a afirma cu încredere că mi-am îndeplinit cu succes țelul propus în lucrarea mea de licență.

|  |
| --- |
| Fig. 38 – Opiniile participanților la studiu referitoare la potențialul jocului de a antrena anumite funcții executive |

# Bibliografie

1. Adam C. Oei, Michael D. Patterson: “Playing a puzzle video game with changing requirements improves executive functions”, 2014.

Division of Psychology, Nanyang Technological University, HSS-04-13, 14 Nanyang Drive, Singapore 637332, Singapore

1. Valerie J. Shute, Matthew Ventura, Fengfeng Ke: The power of play: “The effects of Portal 2 and Lumosity on cognitive and noncognitive skills”, 2014

Florida State University, College of Education, 1114 West Call Street, Tallahassee, FL 32306-4453, USA

1. U.S. Departament of Labor, Bureau of Labor Statistics: “Computer Ownership Up Sharply in the 1990’s”, 4 March 1999.

2 Massachusetts Ave., NE, Washington, DC 20212-0001

1. ”What is Unity? Everyting you need to know”, 2021: <https://www.androidauthority.com/what-is-unity-1131558>
2. Information Technology, University of Washington: ”What is Illustrator?”

<https://itconnect.uw.edu/learn/workshops/online-tutorials/graphics-and-design-workshops/adobe-cs/illustrator/>

1. ”A tour of the C# language”, 2021: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp>
2. Ghid de programare a unei platforme mișcătoare în Unity: https://github.com/SebLague/2DPlatformer-Tutorial/tree/master/Platformer%20E08
3. Ghid de creare a unui laser în Unity: <https://www.youtube.com/watch?v=S6eRVwAtfOM&t=630s>
4. Ghid de creare a unui joc de tip platformer 2D în Unity: <https://github.com/Bardent/Platformer-Tutorial>
5. Documentație și manual de utilizare Firebase: https://firebase.google.com/docs/reference/rest/auth#section-api-usage
6. Domenico Rotolo, „Firebase Database in Unity with REST API”: https://medium.com/@rotolonico/firebase-database-in-unity-with-rest-api-42f2cf6a2bbf
7. Documentația si manualul de utilizare Unity: <https://docs.unity3d.com>
8. Biblioteca RestClient: <https://github.com/proyecto26/RestClient>
9. Biblioteca fullserializer: https://github.com/jacobdufault/fullserializer
10. Resursă grafică vetorială creată de vector\_corp utilizată pentru sigla jocului: https://www.freepik.com/free-vector/abstract-shape-with-watercolor-splash-banners\_2638050.htm
11. Freesound, site pentru efecte sonore fară drepturi de autor: https://freesound.org/
12. Efectul sonor pentru platforma mișcătoare: <https://www.youtube.com/watch?v=j6EbsG1lE0Y&list=OLAK5uy_mQFN3o7wHhhdY8mamTbo8U5yFwpUy9n9g&index=17>
13. Efectul sonor pentru sunetul de tastare pe tastatură: <https://www.youtube.com/watch?v=Z70MOUHQ3TI>
14. Platforma pe care am realizat diagramele UML: https://app.diagrams.net/

# Anexe

## Clasa FirebaseHandler. Funcții relevante

1. private void SignUpUser(string email, string password)
2. {
3. string userData = "{\"email\":\"" + email + "\",\"password\":\"" + password + "\",\"returnSecureToken\":true}";
5. RestClient.Post<SignInResponse>("https://www.googleapis.com/identitytoolkit/v3/relyingparty/signupNewUser?key=" + singletonInstance.authKey, userData)
6. .Then(response =>
7. {
8. CreateNewUserEntry(response.localId, response.idToken);
9. GameObject.FindGameObjectWithTag("DialogBox").GetComponent<DialogBoxCont   roller>().SetMessage("Account created. Go back and sign in to play !");
11. })
12. .Catch(err =>
13. {
14. var error = err as RequestException;
15. if (error != null)
16. {
17. GameObject.FindGameObjectWithTag("DialogBox").GetComponent<DialogBox  Controller>().SetMessage("Error creating account. Try again.");
18. Debug.Log(error);
19. }
20. });
21. }
23. private void SignInUser(string email, string password)
24. {
25. string userData = "{\"email\":\"" + email + "\",\"password\":\"" + password + "\",\"returnSecureToken\":true}";
27. RestClient.Post<SignInResponse>("https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=" + singletonInstance.authKey, userData)
28. .Then(response =>
29. {
30. singletonInstance.isLogged = true;
31. Debug.Log("Successfully signed in");
32. GameObject.FindGameObjectWithTag("DialogBox").GetComponent<DialogBoxControl  ler>().SetMessage("Successfully signed in!");
33. singletonInstance.idToken = response.idToken;
34. singletonInstance.localId = response.localId;
36. SetUsername();
38. })
39. .Catch(err =>
40. {
41. var error = err as RequestException;
42. if (error != null)
43. {
44. GameObject.FindGameObjectWithTag("DialogBox").GetComponent<DialogBoxCon   troller>().SetMessage("Username or password incorrect. Try again");
45. Debug.Log(error.Message);
46. }
47. });
48. }
49. public void PutLevelScore(int levelId, string levelName, float seconds)
50. {
51. if (singletonInstance.idToken != null)
52. {
53. // idToken == nul => player is not signed in so scores will not get saved
54. // idToken != null => player is signed in and the score will be updated if it is a highscore
56. RestClient.Get<User>(singletonInstance.databaseURL + "/userData/" + singletonInstance.localId + ".json").Then(response =>
57. {
58. float previousLevelTime = response.levelsData[levelId].seconds;
59. if (previousLevelTime > seconds || previousLevelTime == 0)
60. {
61. //there is a new highscore => update database entries
62. String timeString=TimeSpan.FromSeconds(seconds).ToString("mm':'ss'.'ff");
63. response.globalTime = response.globalTime - previousLevelTime + seconds;
64. response.levelsData[levelId].timeString = timeString;
65. response.levelsData[levelId].seconds = seconds;
66. response.levelsData[levelId].id = SceneManager.GetActiveScene().buildIndex;
67. response.levelsData[levelId].name = SceneManager.GetActiveScene().name;
68. if (response.currentLevel == levelId)
69. {
70. // store the level with the lowest index that the user did not complete yet
71. response.currentLevel = levelId + 1;
72. }
73. RestClient.Put(singletonInstance.databaseURL + "/userData/" + singletonInstance.localId + ".json?auth=" + singletonInstance.idToken, response).Catch(err =>
74. {
75. var error = err as RequestException;
76. if (error != null)
77. {
78. Debug.Log("Error updating level score: " + err.Message);
79. }
80. });
82. }
83. else
84. {
85. Debug.Log("Not a highscore");
86. }
87. }).Catch(err =>
88. {
89. var error = err as RequestException;
90. if (error != null)
91. {
92. Debug.Log(err.Message);
93. Debug.Log(err.Data);
94. }

97. });
98. }
99. }
100. public void GetUsersList(Action<List<User>> callback)
101. {
102. RestClient.Get(singletonInstance.databaseURL + "/userData.json").Then(response =>
103. {
105. fsData userDataJson = fsJsonParser.Parse(response.Text);
106. Dictionary<string, User> users = null;
107. singletonInstance.serializer.TryDeserialize(userDataJson, ref users);
108. singletonInstance.usersList.Clear();
109. foreach (var user in users.Values)
110. {
111. singletonInstance.usersList.Add(user);
112. }
113. callback(singletonInstance.usersList);
114. }).Catch(err =>
115. {
117. var error = err as RequestException;
118. if (error != null)
119. {
121. Debug.Log(err.Message);
123. }
124. });
125. }

## Clasa Leaderboard

1. public class Leaderboard : MonoBehaviour
2. {
3. [SerializeField]
4. protected Transform entryContainer;
5. [SerializeField]
6. protected Transform entryTemplate;
7. [SerializeField]
8. protected float templateHeight = 30f;
9. protected List<Transform> entryTansforms;
10. [SerializeField]
11. protected ScrollRect scroll;
12. private GameObject dialogBox;
13. private void Awake()
14. {
15. entryTansforms = new List<Transform>();
17. }
19. private void OnEnable()
20. {
21. dialogBox = GameObject.FindGameObjectWithTag("DialogBox");
22. if (dialogBox)
23. {
24. dialogBox.SetActive(false);
25. }
26. }
27. private void OnDisable()
28. {
29. if (dialogBox)
30. {
31. dialogBox.SetActive(true);
32. }
33. }
34. public void PopulateLeaderboard()
35. {
37. FirebaseHandler.GetInstance().GetUsersList((usersList) =>
38. {
39. SortUsers(usersList);
40. SendEmptyToEnd(usersList);
41. InstantiateEntries(usersList);
42. DisplayEntries();
43. ScrollToTop();
44. }
45. );
47. }
49. protected void ScrollToTop()
50. {
51. scroll.verticalNormalizedPosition = 0.5f;
52. Canvas.ForceUpdateCanvases();
53. }
55. protected virtual void SendEmptyToEnd(List<User> usersList)
56. {
58. }
60. protected virtual void SortUsers(List<User> usersList)
61. {
63. }
65. protected void InstantiateEntries(List<User> usersList)
66. {
67. int entryIndex = 1;
68. foreach (User user in usersList)
69. {
70. Transform entryTransform = Instantiate(entryTemplate, entryContainer);
71. RectTransform entryRectTransform = entryTransform.GetComponent<RectTransform>();
72. entryRectTransform.anchoredPosition = new Vector2(0, -templateHeight \* entryIndex + templateHeight / 2);
74. entryTransform.Find("Position").GetComponent<TextMeshProUGUI>().text = entryIndex.ToString();
75. entryTransform.Find("Username").GetComponent<TextMeshProUGUI>().text = user.username;
77. SetEntryValues(entryTransform, user);
78. entryTansforms.Add(entryTransform);
80. entryIndex++;
82. }
83. }
85. protected virtual void SetEntryValues(Transform entryTransform, User user)
86. {
88. }
90. protected void DisplayEntries()
91. {
93. foreach (var entry in entryTansforms)
94. {
95. entry.gameObject.SetActive(true);
96. }
97. }
99. public void ClearLeaderboard()
100. {
101. if (entryTansforms != null)
102. {
103. foreach (var entry in entryTansforms)
104. {
105. Destroy(entry.gameObject);
106. }
107. entryTansforms.Clear();
108. }
109. }
110. }

## Funcția de calculare a mișcării platformei

1. private void FixedUpdate()
2. {
3. if (isMoving || !isTriggerable)
4. {
5. transform.Translate(CalculatePlatformMovement());
6. UpdateGraph();
7. }
8. }
9. private Vector3 CalculatePlatformMovement()
10. {
11. Vector3 newPosition;
12. float distance;
14. // platform only moves when TriggerFunction() is called
15. if (isTriggerable)
16. {
17. Vector3 destination = (isTriggered) ? triggeredPosition : unTriggeredPosition;
18. distance = Vector3.Distance(transform.position, destination);
19. percentBetweenWaypoints += Time.deltaTime \* speed / distance;
21. newPosition = Vector3.Lerp(transform.position, destination, percentBetweenWaypoints);
22. if (1 - percentBetweenWaypoints < 0.1f) {
23. percentBetweenWaypoints = 0;
24. isMoving = false;
25. FindObjectOfType<SoundManager>().Play("platform\_init");
26. FindObjectOfType<SoundManager>().Stop((isTriggered) ? "platform\_go" : "platform\_come");
28. }
30. }
31. // platform moves all the time
32. else
33. {
34. int toWaypointIndex = fromWaypointIndex + 1;
35. distance = Vector3.Distance(transform.position, globalWaypoins[toWaypointIndex]);
36. percentBetweenWaypoints += Time.deltaTime \* speed / distance;
38. newPosition = Vector3.Lerp(transform.position, globalWaypoins[toWaypointIndex], percentBetweenWaypoints);
40. if (1 - percentBetweenWaypoints < 0.01f)
41. {
42. percentBetweenWaypoints = 0;
43. fromWaypointIndex++;
44. if (fromWaypointIndex >= globalWaypoins.Length - 1)
45. {
47. fromWaypointIndex = 0;
48. System.Array.Reverse(globalWaypoins);
49. }
50. isMoving = false;
52. }
53. }
54. return newPosition - transform.position;
56. }

1. Cut the Rope – joc video puzzle pentru Android bazat pe reguli ale fizicii ce necesită formularea si revizuirea unui plan de acțiune, imaginație și învățare din greșeli anterioare prin testare si eșuare. [↑](#footnote-ref-1)
2. Gaură de vierme - Reprezintă o metodă teoretică prin care două zone îndepărtate din univers sunt unite între ele printr-o scurtătură. În consecință, din punct de vedere teoretic, permite călătoria aproape instantanee dintr-un loc în altul. [↑](#footnote-ref-2)
3. Lumosity – Program computerizat de antrenare a abilitaților cognitive de baza precum memoria, atenția, viteza de procesare, flexibilitatea mentală, orientarea spațiala si rezolvarea de probleme. [↑](#footnote-ref-3)
4. Adobe Flash – Produs software folosit pentru a produce animații, aplicații web și desktop, jocuri și aplicații pentru mobil și playere video pentru browsere web. [↑](#footnote-ref-4)
5. NoSQL – Un tip de arhitectura de baze de date ce permite o viteza operationala ridicata si o flexibilitate mare din partea dezvoltatorului [↑](#footnote-ref-5)
6. Grafica raster – Grafica redata prin matrice dreptunghiulara de puncte ce reprezinta o grila de pixeli, usor reprezentabila pe monitoare. [↑](#footnote-ref-6)
7. Sprite – Obiect 2D simplu care are imagini grafice atasate numite texturi. [↑](#footnote-ref-7)
8. Singleton Design Pattern – O soluție bine documentată la o problemă des întâlnită în programarea orientată obiect care restricționează instanțierea la un singur exemplar a unui obiect ce aparine unei clase. [↑](#footnote-ref-8)
9. Software Development Kit – SDK este o mulțime de unelte software și programe folosite de dezvoltatori pentru a crea aplicații pentru anumite platforme. [↑](#footnote-ref-9)
10. Metode HTTP – Funcții special concepute pentru a realiza comenzi de ștergere, citire, creare și actualizare prin protocolul HTTP [↑](#footnote-ref-10)
11. SOLID – Acronim pentru o multime de principii care aduc solutii pentru probleme des intalnite in industria de dezvoltare de produse software, eliminand proprietatile de rigiditate, fragilitate si imobilitate ale codului. [↑](#footnote-ref-11)
12. Prefab – Un șablon ce poate fi salvat în Unity pentru a recrea, configura și adauga în scenă un obiect, menținându-i toate proprietățile precum componente, valori, și elemente copil. [↑](#footnote-ref-12)
13. LineRenderer – Componentă atașabilă obiectelor din Unity ce primește un vector de puncte și desenează o linie între fiecare, având posibilități multiple de configurare. [↑](#footnote-ref-13)
14. Animation Controller – Panou ce permite aranjarea și gestionarea unui set de animații și tranzițiile dintre acestea pentru un anumit caracter sau obiect. [↑](#footnote-ref-14)
15. LayerMask – Obiect ce stocheaza o filtrare a straturilor pe care sa se afle obiectele dorite a fi luate in considerare. [↑](#footnote-ref-15)