#### 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](#_Toc74739916)

[1.1 Context 8](#_Toc74739917)

[1.2 Motivatie 11](#_Toc74739918)

[2 Aplicatii similare 12](#_Toc74739919)

[*2.1* *Portal 2* 12](#_Toc74739920)

[*2.2* *FireBoy and WaterGirl* 14](#_Toc74739921)

[3 Tehnologii utilizate 15](#_Toc74739922)

[3.1 Unity 15](#_Toc74739923)

[3.2 Limbajul de programare C# 15](#_Toc74739924)

[3.3 Firebase 16](#_Toc74739925)

[3.3.1 Authentification 16](#_Toc74739926)

[3.3.2 Realtime Database 16](#_Toc74739927)

[3.4 Adobe Illustrator 17](#_Toc74739928)

[4 Prezentarea aplicatiei 18](#_Toc74739929)

[4.1 Ferestre 18](#_Toc74739930)

[4.1.1 Schema de navigare 18](#_Toc74739931)

[4.1.2 Meniul principal 19](#_Toc74739932)

[4.1.3 Clasament online 19](#_Toc74739933)

[4.1.4 Mapa nivelelor 19](#_Toc74739934)

[4.1.5 Sign In / Sign Up 20](#_Toc74739935)

[5 Detalii de implementare 21](#_Toc74739936)

[5.1 Introducere 21](#_Toc74739937)

[5.2 Baza de date 22](#_Toc74739938)

[5.2.1 Autentificare / Inregistrare 23](#_Toc74739939)

[5.2.2 Clasament online 24](#_Toc74739940)

[5.3 Mediul 25](#_Toc74739941)

[5.3.1 Buton si maneta 26](#_Toc74739942)

[5.3.2 Laser 27](#_Toc74739943)

[5.3.3 Platforma miscatoare 28](#_Toc74739944)

[5.3.4 Usa 29](#_Toc74739945)

[6 Ghid de utilizare 49](#_Toc74739946)

# Introducere

## Context

Primele concepte din istoria umanitatii ce pot fi incadrate in categoria puzzle sunt ghicitorile, iar acestea dateaza inca din era mitologiei grecesti. Se considera ca prima enigma creata de om este “The Riddle of the Sphinx” (Ghicitoarea Sfinxului) ce se traduce in felul urmator: Ce creatura are o voce dar devine patrupeda, apoi bipeda iar apoi tripeda? In procesul de a gasi raspunsul, cel ce incearca sa desluseasca misterul se conformeaza unor reguli si ipoteze initiale; avand acest punct de plecare, se foloseste de un proces de gandire logica si sistematica, bazata pe incercare si esec pana converge spre solutie: omul are o voce, iar cele 3 categorii fac referire la 3 etape din viata acestuia: copilaria timpurie, cand modalitatea de deplasare se aseamana patrupedelor, viata de adult si batranetea cand se foloseste un baston pentru sprijin. De atunci si pana in ziua de astazi puzzleurile s-au aflat in continua expansiune, cautand sa exploateze din ce in ce mai multe arii, cum ar fi dificultatea, mecanica, logica, matematica, iar odata cu tangenta cu tehnologia si informatica, au aparut jocurile video, dand astfel glas acestei lucrari de licenta.

Jocurile video au devenit, fara indoaiala, o parte semnificativa din viata omului modern, aducand astfel un aport considerabil de influenta in ceea ce priveste modul de dezvoltare al unei persoane, cu atat mai mult la varste fragede.

|  |
| --- |
| Fig. 1 - Procentul personanelor ce detineau un calculator in functie de nivelul de educatie |

Dupa cum se poate observa in Fig. 1, de la inceputul pana la finalul anilor 90, procentul persoanelor cu studii superioare ce detineau un calculator s-a dublat sau chiar triplat in aceasta perioada de timp. De asemenea, conform Fig. 2, procentul majoritar de persoane care erau pasionate de jocuri video in anul 2020 in Statele Unite se regaseste in intervalul de varsta de la 18 la 34 de ani. Interpretand aceste multimi de date, pot trage concluzia ca persoanele trecute de perioada adolescentei in anul 2020 au avut acces la un calculator si primele tangente cu jocurile video in copilarie, datorita faptului ca parintii lor se aflau intr-o perioada de stabilitate financiara in momentul in care acestea au devenit disponibile si accesibile pe piata. Procentul majortar pe care acestia il reprezinta releva aderarea lor la acest hobby si pastrarea in continuare in viata adulta a acestei activitati.

|  |
| --- |
| Fig. 2 - Distributia dupa varsta a utilizatorilor de jocuri video in Statele Unite, in 2020 |

Propun prin intermediul lucrarii mele sa contribui la dezvoltarea industriei de jocuri video intr-o directie educativa ce aduce potentiale beneficii intelectuale utilizatorului, imbinand conceptul de puzzle intr-un joc de tip platformer 2D pentru sistemul de operare Windows. Consider ca provocarile pe care le-am inglobat in diversele nivele pe care le-am conceput in jocul meu au posibilitatea de a antrena si de a imbunatati latura creierului uman ce se ocupa cu functiile executive. Aceastea reprezinta o multime de procese mentale care ne ajuta sa conectam experienta anterioara cu o actiune din prezent. Exersarea acestei capacitati contribuie mai departe la optimizarea unor abilitati esentiale, cum ar fi: atentia, planificarea, memoria, selectarea informatiilor senzoriale relevante,monitorizarea si interpretarea stimulilor interni si externi, auto-controlul, actionarea si luarea de decizii. In urma unui studiu realizat in cadrul Universitatii Tehnologice Nanyang din Singapore a putut fi trasa concluzia ca un joc video puzzle bazat pe mecanici ale fizicii are capacitatea de a imbunatati functiile executive, spre deosebire de alte categorii de jocuri video. Subiectii au fost expusi inainte de experiment la teste ce vizeaza schimbarea aleatorie de sarcini, inhibarea raspunsurilor si abilitatea de a filtra stimuli; datele arata ca in urma a 20 de ore de joc, persoanele care au interactionat cu aplicatia “Cut the Rope”[[1]](#footnote-1) au prezentat cele mai semnificative diferente de aptitudini cand au fost rugati sa refaca testele anterioare.

Acest experiment este un punct de plecare bun spre a putea raspunde la intrebarea: Este posibil ca jocurile video de tip puzzle sa aiba drept efect antrenarea anumitor aptitudini mentale? Fiind inspirat de aceasta problema, am decis ca prin aceasta lucrare sa-mi setez acelasi punct de plecare, dar privit dintr-o alta perspectiva: Se poate proiecta un joc care sa aiba in vedere si sa vizeze in mod direct antrenarea unei anumite functii executive? Astfel, am conceput un joc video de tip platformer 2D ce inglobeaza printr-o interfata tematica 6 nivele de dificultati variate. Mecanica jocului consta in miscarea alternativa a doua personaje, unul terestru si unul aerian, ce au posibilitatea de a interactiona unul cu celalalt sau cu mediul inconjurator. Scopul unui nivel este de a colabora in vederea eliberarii obstacolelor ce nu permit iesirea din incapere. Utilizatorul este invitat sa accepte provocarea nivelelor de a-si imbunatati atentia, capacitatea de planificare si rezolvare de probleme, memoria si gandirea logica, toate sub presiunea unui cronometru. Drept stimulent pentru concentrare si motivatie de a obtine cel mai bun rezultat, am adaugat un element competitiv; jocul stocheaza online cele mai bune rezultate pentru fiecare nivel completat cu succes al utilizatorului logat, oferind astfel un clasament global al tuturor scorurilor obtinute de catre toti jucatorii.

## Motivatie

Tematica jocului este inspirata din desenul meu animat preferat din copilarie, Phineas si Ferb, ce mi-a captat atentia de la primele episoade vizionate deoarece creatorii au dat dovada, in opinia mea, de o creativitate si imaginatie impresionanta, trasaturi pe care consider ca mi le-au pasat si mie intr-o oarecare masura prin produsul lor animat. O narativa secundara a serialului se invarte in jurul animalului de companie al celor doua personaje principale, un ornitorinc, care in secret este un agent special ce lupta impotriva unui antagonist a carui atu este bagajul de cunostinte despre stiinta, ce ii permite sa inventeze masinarii si capcane, in afara oricaror reguli ale naturii sau fizicii, ironic, avand in vedere profesia sa. Inspirat de relatia acestor doua personaje, pasiunea pentru puzzleuri, curiozitatea tehnologiei ce sta in spatele crearii jocurilor video si motivat de rezultatul obtinut de studiul prezentat anterior, am decis sa imbin cunostinele de programare dobandite in facultate cu ingeniozitatea si posibilitatile oferite de acest desen animat intr-un produs personal original, ce aspira spre a putea oferi utilizatorului satisfactia si beneficiile aduse de rezolvarea cu succes a unui puzzle prin intermediul tehnologiei.

# Aplicatii similare

Exista o multitudine de jocuri video ce exploateaza diferite mecanici cu scopul de a construi sisteme complexe care plaseaza jucatorul in scenarii ce propun gasirea unei solutii la problema expusa. Se creeaza o strategie de abordare a nivelelor folosind gandirea logica pentru a alcatui un plan compus dintr-o inlantuire de actiuni, fiecare esec dezvaluind o noua informatie relevanta. Dupa cum se poate observa, aceeasi esenta sta atat la baza primului tip de puzzle, ghicitoarea, cat si la baza jocurilor video, iar diferite companii specializate in dezvoltarea jocurilor video au avut inspiratia sa valorifice acest fapt; prin urmare, in acest capitol voi enunta proiectele ce m-au inspirat in a alege aceasta tema.

## *Portal 2*

*Portal 2* este un joc “first-person” 3D de tip platformer, realizat de compania Valve Corporation, a carui scop principal este parasirea unei camere, initial inchisa, prin deschiderea unei usi folosind diverse unelte. Mecanica pentru care acest joc si-a creat un renume este reprezentata de o arma ce are capacitatea de a plasa doua portaluri interconectate, doar pe anumite suprafete, ce au proprietati asemanatoare concepului de gura de vierme[[2]](#footnote-2). Orice obiect trece prin oricare din cele doua terminale isi continua miscarea prin celalalt, pastrandu-si toate proprietatile fizice, cum ar fi directia de miscare relativa la portal, viteza, acceleratia sau inertia.

|  |  |
| --- | --- |
| Fig. 3 – Ilustrarea modului de functionare al portalurilor | Un scenariu interesant ce devine posibil profitand de aceasta mecanica este ilustrat in Fig. 3; platforma din dreapta este inital inaccesibila, dar folosind aceasta arma, jucatorul are capacitatea de a se folosi intr-un mod ingenios de inertia si viteza ce sunt dobandite prin cadere libera: plasand cele doua terminale pe suprafete perpendiculare, directia de deplasare este rotita cu 90°, continuandu-si astfel caderea urmand o traiectorie asemanatoare unui glont. |

Alte unelte folosite in rezolvarea puzzle-urilor:

* “Thermal Discouragement Beams” este un laser, ce are drept punct de plecare un emitator fix, care transporta o cantitate de energie necesara pentru a activa alte elemente din mediul inconjurator, necesare pentru finalizarea nivelului.
* “Excursion Funnel” reprezinta o arma ce ii ofera posesorului capacitatea de a manevra prin tractare, mutare, sau impingere obiecte aflate la distanta.
* “Hard Light Bridge” este o platforma ce se extinde de la un emitator pe o traiectorie rectilinie pana la primul obstacol.
* “Redirection Cube” denota un cub alcatuit din suprafete reflectorizante utilizate pentru a manipula directia laserelor.

Toate aceste elemente plasate strategic in calea jucatorului creeaza un orizont larg de posibiliatati si combinatii in care pot fi utilizate impreuna, motiv pentru care Portal 2 a fost, precum “Cut the Rope”, subiectul unui studiu ce viza acelasi obiectiv: posibilitatea imbunatatirii anumitor aptitudini din sfera functiilor executive. Experimentul desfasurat la Universitatea din Florida, SUA, documentat in lucrarea “*The power of play: The effects of Portal 2 and Lumosity on cognitive and noncognitive skills”* scoate in evidenta efectele pozitive identificate la participanti, in urma unei sesiuni de 8 ore de joc asupra abilitatilor precum rezolvarea de probleme, aptitudini spatiale si persistenta. Testul a fost realizat avand drept sistem de referinta programul *Lumosity*[[3]](#footnote-3), fata de care *Portal 2* a inregistrat imbunatatiri semnificative in toate cele 3 arii mentionate anterior.

## *FireBoy and WaterGirl*

*FireBoy and WaterGirl,* aparut in 2009,este un joc de tip platformer 2D dezvoltat de Oslo Albet pentru browsere web folosind Adobe Flash[[4]](#footnote-4), publicat prima data pe site-ul Armor Games.Asemanator *Portal 2*, scopul jocului este de a rezolva puzzleuri pentru a parasi camera ce alcatuieste nivelul curent. Jucatorul are posibilitatea de a controla simultan doua personaje ce difera doar la nivel estetic, dar modul in care acestea interactioneaza cu mediul le deosebeste; obiectele alcatuite din acelasi element sunt complet inofensive pentru caracterul corespunzator, in timp ce pentru celalalt sunt fatale.

|  |  |
| --- | --- |
| Fig. 4 – Nivelul 5, WaterGirl and FireBoy in The Forest Temple | Acest concept simplu, ilustrat in Fig. 4, imbinat cu butoane, manete, balante, portaluri, lasere si oglinzi utilizate folosind cooperare, alcatuiesc un produs excelent si educativ pentru cei tineri. Capcanele gandite de dezvoltator necesita gandire logica si in avans, planificare bine pusa la punct, coordonare si cooperare. |

Motivul principal pentru care am ales acest joc drept punct principal de plecare pentru lucrarea mea este faptul ca, desi are capacitatea de a pune in dificultate utilizatorii chiar si cand este jucat in doi, odata acceptata provocarea de a infrunta nivelele singur, experienta creste exponential atat in dificultate, cat si in beneficii. Consider ca abilitatea de a controla simultan cele doua caractere, evitand pericolele, in timp ce este compus si un plan de rezolvare a puzzleurilor, poate constitui un antrenament mental de coordonare si multifunctionalitate, chiar si pentru cei mai priceputi. O dovada a existentei imbunatatirilor aduse acestor abilitati in urma interactiunii cu jocul este faptul ca, initial, orice jucator va controla alternativ caracterele, dar, dupa o perioada de antrenament, va atinge performanta de a putea fi atent in doua locuri diferite pe ecran, coordonandu-si mainile in acelasi timp pentru a controla simultan cele doua caractere diferite.

# Tehnologii utilizate

## Unity

Unity este una dintre cele mai bine dezvoltate unelte alese de dezvoltatorii de jocuri video pentru a-si transforma creatiile din concept in realitate. In calitate de motor si mediu de creare a jocurilor video, Unity furnizeaza utilizatorilor majoritatea uneltelor necesare pentru realizarea unui astfel de produs, cu alte cuvinte, programatorii nu trebuie sa reinventeze roata, ci din potriva, pot revolutiona piata cu produse de calitate superioara folosindu-se de linia de asamblare performanta si complet automata. Pentru a oferi si un sens denotativ acestei analogii, aduc completarea ca nu este nevoie sa fie calculate si implementate, de fiecare data, aceleasi module ce simuleaza, de exemplu, modul in care interactioneaza obiectele in lumea reala, respectand legile fizicii, sau modul in care lumina se reflecta pe diferite suprafete. Aceste solutii sunt oferite atat pentru crearea aplicatiilor 2D cat si 3D, pentru o multitudine de platforme, cum ar fi Linux, Mac, Windows, Android, IOS, Xbox sau Play Station.

O alta caracteristica extrem de utila este magazinul de componente (“Asset Store”); acesta este un spatiu unde oamenii isi pot incarca creatiile, punandu-le la dispozitia tuturor, contra-cost sau gratuit. Astfel, este alcatuita o comunitate ce promoveaza ingeniozitatea, creativitatea si generozitatea, unde participantii isi pot impartasi viziunile si contribuie colectiv la dezvoltarea acesteia prin impartasirea bunurilor precum componente vizuale 2D sau 3D, texturi, materiale, animatii, interfete, biblioteci, efecte speciale si audio.

Unity este in aceeasi masura si un mediu de dezvoltare integrat (Integrated Development Environment) ce are rolul de a centraliza toate aceste instrumente intr-un singur editor, cu o interfata grafica extrem de intuitiva si usor de manipulat. Acesta a ajuns la performanta in care o simpla interactiune intre obiecte poate fi simulata doar prin adaugarea in scena a elementelor deja puse la dispozie, urmata de ajustarea proprietatilor aceastora, fara a tasta o singura linie de cod.

## Limbajul de programare C#

C# este un limbaj de programare modern, orientat-obiect ce permite dezvoltarea produselor software intr-o maniera naturala, remarcandu-se prin simplitate, utilitate generala si productivitate sporita. Furnizeaza programatorului uneltele necesare pentru construirea unei aplicatii robuste si de durata, precum colectorul de gunoi pentru gestionarea eficienta a memoriei, tipul de data null pentru variabile ce nu au asociate obiecte instantiate, gestionarea exceptiilor, expresii lamda, clase, suport pentru operatii asincrone si multe altele.

Desi Unity are o puterea de a realiza produse impresionant de complexe fara a scrie cod, pentru a-mi atinge scopul in acest proiect am avut prilejul de a ma folosi de utilitatea acestui limbaj pentru a descrie clase si interactiunea dintre acestea, comportament ce se propaga mai departe la obiectele propriu-zise din joc carora le sunt atasate aceste segmente de cod. In ciuda faptului ca nu am mai avut tangente cu C# in trecut, similaritatea acestuia cu limbajul de programare Java si usurinta si naturaletea aplicarii gandirii POO (Programare orientata obiect) in cadrul acestuia mi-au facilitat aclimatizarea in acest mediu nou, putand chiar sa afirm ca nu am intampinat probleme care sa aiba drept cauza lipsa cunostinelor de sintaxa si utilizare specifice limbajului.

## Firebase

Firebase este o platforma sustinuta de Google destinata dezvoltarii de produse software ce permite utilizatorilor sa creeze aplicatii concepute pentru sisteme de operare precum IOS, Android sau pentru Web. Sfera de intrebuintari ale acestui proiect se extinde vast, dar in capitolele urmatoare voi prezenta succint functionalitatile pe care le-am folosit eu.

### Authentification

Prin acest serviciu, Firebase ofera dezvoltatorului posibilitatea de a pune la dispozitie in aplicatia sa un sistem de autentificare sigur ce imbunatateste experienta de logare a utilizatorului. Sunt disponibile multiple modalitati de demonstrare a identitatii, cum ar fi, atat un sistem intern de gestionare a conturilor create prin email si parola, cat si prin apelarea la terti precum Google, Facebook, GitHub sau Twitter.

### Realtime Database

Probabil unul dintre capacitatile principale ale acestei platforme este baza de date in timp real de timpul NoSQL[[5]](#footnote-5). Aceasta este stocata pe serverele lor, ceea ce ofera proprietatea de persistenta a datelor salvate intr-o anumita sesiune in cadrul aplicatiei tinta, chiar si cand aceasta este inactiva. Cu atat mai mult, toate modificarile realizate asupra bazei de date sunt sincronizate in timp real, astfel devenind disponibile tuturor utilizatorilor, la cateva secunde de la incheiera tranzactiei.

## Adobe Illustrator

Adobe Illustrator este un program profesional de desenare si schitare a produselor vizuale folosind tehnologie bazata pe vectori. Acesta permite crearea elementelor precum postere, simboluri, sigle, iconite si multe altele, avand complexitati de la cele mai simple elemente precum forme geometrice la intregi compozitii.

Aceasta caracteristica principala a programului, grafica vectoriala, merita mentionata datorita posibilitatilor pe care aceasta le deblocheaza. Toate ustensilele disponibile in panoul principal deseneaza folosind linii si curbe care au la baza aceste obiecte matematice numite vectori. Spre deosebire de grafica raster[[6]](#footnote-6), unde, spre exemplu, o roata de bicicleta ar fi redata prin setarea pixelilor din matrice intr-o forma circulara, grafica vectoriala asociaza roata cu formula matematica a cercului, colorand punctele corespunzatoare zonei interioare a acestuia. Astfel, redimensionarea rotii in primul caz ar rezulta in pierderea calitatii prin manipularea pixelilor, pe cand in al doilea caz se poate obtine orice dimensiune dorita prin ajustarea parametrilor functiilor matematice, fara compromisuri. Vorbind in termeni generali, tehnologia este independenta de rezolutie deci produsele pot fi redimensionate la orice marime si afisate pe orice dispozitiv, indiferent de rezolutie, fara a pierde calitate, motiv pentru care care a reprezentat ustensila ideala pentru a realiza componentele vizuale din cadrul jocului meu.

# Prezentarea aplicatiei

## Ferestre

Jocul este compus din sapte scene, dintre care una reprezinta meniul compus din cinci ferestre inter-navigabile, iar restul sase sunt asociate fiecarui nivel. Consider ca interfata unui joc are un rol destul de important in experienta utilizatorului, motiv pentru care nu am neglijat acest aspect, alocand timp design-ului acesta, atat din punct de vedere estetic, cat si functional.

### Schema de navigare

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

1. Play: Afiseaza ecranul de selectie a nivelului
2. Leaderboard: Afiseaza tabela de clasament
3. Sign In: Afiseaza meniul de logare in aplicatie
4. Sign Up: Afiseaza meniul de creare a unui cont nou
5. Play: Incarca nivelul corespunzator selectiei 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 interactioneaza jucatorul in urma rularii executabilului. Odata incarcat, utilizatorul este intampinat de titlul jocului asociat cu sigla acestuia, urmate de optiuni clare de a incepe jocul, de a vizualiza tabela de clasament online, inchiderea aplicatiei, logare sau crearea unui cont.

### Clasament online

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

Am adaugat optiunea unui clasament online (Fig. 7) pentru a oferi o motivatie in plus de a obtine o perfomanta cat mai buna. Odata ce un nivel este completat cu succes, timpul este incarcat in baza de date Firebase de unde este accesata de script-ul ce populeaza tabela. Prima pagina, intitulata “Global”, furnizeaza o ordonare crescatoare al sumei timpilor obtinuti la toate cele 6 nivele. Navigand in continuare prin tab-urile numerotate de la 1 la 6, se pot vizualiza, de asemenea, si clasamentele individuale pentru fiecare nivel.

### Mapa nivelelor

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

Odata apasat butonul “Play”, este afisat meniul de selectie a nivelului dorit (Fig. 8). Am conceput pentru acesta un design ingenios ce inglobeaza elemente regasite si in cadrul jocului. La incarcarea ecranului se deschid trapele corespunzatoare tuturor nivelelor deja completate, inclusiv cel curent. Odata deschise, utilizatorul are posibilitatea de a selecta pe cel dorit.

### Sign In / Sign Up

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

Exista 2 ipostaze in care jucatorul poate parcurge nivelele: mod anonim sau autentificat; prima varianta presupune inceperea jocului fara logare, ceea ce conduce la lipsa inregistrarii scorului sau a prezentei in tabela de clasament. Singura memorare ce are loc este progresul acumulat, reprezentat de nivelul curent necompletat, care va fi stocat local. In cel de-al doilea caz, utilizatorul beneficiaza de exprerienta completa a jocului, ambele procese de logare si autentificare fiind insotite de mesaje ajutatoare corespunzatoare pentru situatii in care sunt completate campurile gresit, de exemplu, parola si verificarea acesteia nu coincid, mail-ul nu are un format valid, absenta unor informatii sau absenta unui cont asociat cu datele furnizate.

# Detalii de implementare

## Introducere

In cele ce urmeaza voi prezenta partile componente ale jocului si modul in care interactioneaza toate obiectele. In procesul de creatie, m-am folosit de diverse unelte pe care le pune la dispozitie Unity, dintre care cele ce merita mentionate sunt cele fundamentale, care se afla la baza majoritatii elementelor ce presupun interactiune sau dinamism:

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

* **MonoBehaviour** este numele clasei de baza pe care o va mosteni orice script care urmeaza sa fie atasat unui obiect. Aceasta furnizeaza o infrasctructura pentru programarea obiector, usor de inteles si de programat. Pune la dispozitie functiile necesare pentru manevrarea proprietatilor in orice stare s-ar afla instanta respectiva, avand o ordine clara de executare. Din Fig. 11 se observa functiile de initializare, Awake() si Start() avand loc la instantierea obiectului, iar OnEnable la la activarea obiectului in scena. Update() si FixedUpdate() se repeta la fiecare frame, respectiv la un interval de timp constant; ele sunt utilizate in special pentru evenimentele ce tin de logica jocului, cum ar fi input-ul de la utilizator, si respectiv gestionarea fizicii, cum ar fi aplicarea constanta a unei forte. Analog initializarii, au loc functiile OnEnable() si OnDestroy().
* **Componentele** sunt atasate obiectelor si au rolul de a gestiona diferite tipuri de interactiuni. Printre cele mai importante pe care le-am folosit se numara, de exemplu, Rigidbody care ofera posesorului capacitatea de a se supune tuturor legilor fizicii, precum grativatia, inertia, viteza, acceleratia sau momentul de rotatie. De asemenea, pentru ca orice 2 obiecte sa poata interactiona, au nevoie de un Collider care gestioneaza modul in care coliziunea dintre acestea are loc. Din punct de vedere vizual, elementare sunt SpriteRenderer si Animator ce se folosesc de elemente grafice precum animatii si sprite-urlie[[7]](#footnote-7) din care acestea sunt compuse pentru a simula miscare, dand astfel viata obiectelor.
* **Conceptele de programare orientata obiect** care m-au ajutat cel mai mult si pe care le-am exploatat in mod repetat sunt mostenirea si Singleton Design Pattern[[8]](#footnote-8). Astfel, am avut posibilitatea sa reutilizez fragmente substantiale de cod pentru obiecte cu comportamente asemanatoare, cum ar fi butoanele, manetele si usile. De asemenea, piesele care se remarca prin proprietatea de unicitate sunt atat gestionatorul bazei de date, cat si personajele principale, toate acestea fiind accesibile global in toata aplicatia.

## Baza de date

Mediatorul dintre baza de date si aplicatie este un script denumit FirebaseHandler, accesibil global printr-o functie statica care returneaza singura instanta a acestei clase, urmand design pattern-ul mentionat in capitolul anterior. Am conceput aceasta clasa sa inglobeze in cateva functii cu scop clar toate operatiile necesare aplicatiei pentru a autentifica sau crea un cont utilizatorului si pentru a salva progresul si timpii record obtinuti de acesta la fiecare nivel, astfel respectand principiul de incapsulare a programarii orientate obiect.

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

Arhitectura bazei de date fiind de tip NoSQL, aceasta este organizata sub forma unor dosare imbricate. Dupa cum se poate observa in Fig. 12, fiecarui cont creat in interiorul aplicatiei ii este asociat un nod “User” care contine informatii precum un identificator unic (“localId”), generat automat de platforma la inregistrare, date personale, progresul in joc si o lista de noduri de tip “LevelData”. Acestea stocheaza informatiile despre fiecare nivel in parte, cum ar fi numele acestuia, indexul si timpul in care a fost completat cu succes, atat sub forma de numar rational, care arata numarul de secunde si milisecunde, cat si structurat ca sir de caractere de forma “minute:secunde.milisecunde”.

Un impediment pe care l-am intampinat incercand sa realizez comunicarea intre baza de date si aplicatia mea este faptul ca Firebase pune la dispozitie utilitare SDK[[9]](#footnote-9) usor integrabile cu Unity pentru platformele Android si IOS. Pentru a rezolva aceasta problema am importat o biblioteca de pe Unity Store numita RestClient care faciliteaza trimiterea de metode HTTP[[10]](#footnote-10), astfel folosindu-ma de API-ul pus la dispozitie de Firebase pentru aplicatiile Web.

### Autentificare / Inregistrare

Pentru a realiza autentificarea si inregistrarea utilizatorului trimit comenzi de tip GET, respectiv POST catre serverul Firebase, asteptand drept raspuns un cod de confirmare care poate fi 200: “OK”, sau 400: “Not supported”. Primul reprezinta confirmarea acestor doua actiuni, iar cel de-al doilea poate avea interpretari multiple, de exemplu, faptul ca se incearca accesarea unui cont inexistent sau crearea unuia deja existent, parola introdusa este gresita sau datele de intrare nu sunt valide. Odata ce comanda asociata butonului de “Sign Up” se realizeaza cu succes, noul cont va fi salvat si afisat in pagina denumita “Authentification” (Fig. 13), de unde se pot gestiona toate aceste intrari.

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

### Clasament online

La finalul fiecarui nivel am plasat un obiect denumit “NextSceneLoader”, care are atasate doar doua componente, un collider si un script; cea de-a doua are rolul, pe langa de a incarca urmatoarea scena, de a salva in baza de date timpul in care jucatorul a completat nivelul. Ea asculta prin intermediul primului atasament evenimentul de coliziune simultana cu cele doua caractere pe care jucatorul le are in control, aceasta fiind una din cerintele obligatorii pentru oprirea cronometrului. Odata urcate informatiile pe server, vor fi vizibile in panoul de gestionare a bazei de date din Fig. 14, in nodul care are drept denumire numarul de ordin al nivelului, asociat fisierului desemnat utilizatorului, de unde pot fi preluate si prelucrate in timp real pentru a alcatui clasamentul online. Acest eveniment are loc doar in cazul in care jucatorul este autentificat, altfel doar progresul curent va fi salvat local in memoria aplicatiei. Pentru a fi posibila si securizata scrierea, Firebase dispune si de un sistem de gestionare a permisiunilor de a realiza o anumita actiune. In contextul aplicatiei mele consider ca setarea compatibila este cea in care citirea poate fi realizata de catre oricine, pentru a putea consulta clasamentul in orice moment, dar am configurat scrierea sa fie posibila doar de catre un jucator autentificat, pentru a preveni modificarea neautorizata a datelor altui utilizator.

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

O provocare des intalnita in comunicarea cu baze de date este potentialul timp indelungat de asteptare pe care il poate genera aceasta interactiune; solutia se regaseste intr-o latura interesanta a limbajelor precum C#, programarea asincrona. Ea presupune rularea in paralel a unor secvente de cod, separat de firul principal de executie, avand control asupra progresului acesteia. Beneficiile ce deriva sunt imbunatatirea performantei si a timpului de raspuns a aplicatiei, fiind cel mai usor observabile la nivelul interfetei cu care interactioneaza utilizatorul deoarece devine mereu functionala si interactiva, eliminand posibilitatea ca aceasta sa se blocheze pentru a astepta terminarea unei anumite secvente de instructiuni. Se poate observa in **ANEXA 1 ??** ca toate interogarile realizate prin libraria RestClient sunt de tip asincron, codul introdus in blocul Then() fiind executat odata ce raspunsul de la serverul Firebase este receptionat. O problema pe care am intampinat-o datorita acestui comportament este faptul ca nu aveam posibilitatea de a obtine o lista cu informatii despre utilizator apeland o functie publica din exteriorul clasei, deoarece aceasta ajungea la instructiunea de return, inainte de a executa codul din blocul Then(). In urma unui proces de cercetare, am descoperit posibilitatea de a accepta ca parmetru un obiect de tip Action<T>, prin care sa furnizez la apel o functie anonima care sa fie executata odata ce informatiile necesare au fost extrase din baza de date. Astfel, clasa denumita Leaderboard din **ANEXA 2 ??** poate realiza toate operatiile necesare popularii bazei de date prin secventa de instructiuni pasata functiei ce le obtine. Pentru o analiza mai detaliata a scorurilor, pe langa clasamentul global (Fig. 15), am pus la dispozitie si pagini asociate fiecarui nivel, cu timpii respectivi sortati in ordine crescatoare (Fig. 16).

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

## Mediul

Interactiunea cu mediul a personajelor consta in activarea si dezactivarea a anumitor elemente dinamice cum ar fi butoane si manete care sunt asociate in mod intuitiv prin aceeasi culoare cu obiectele precum lasere, platforme miscatoare si usi. Desi au scopuri si proprietati diferite, ele se aseamana prin faptul ca toate realizeaza o miscare in momentul in care primesc un semnal. Dupa cum se poate observa in diagrama de clase UML din Fig. 17, toate mostenesc clasa Trigger care defineste metoda TriggerFunction() ce initiaza respectiva miscare.

|  |
| --- |
| Fig. 17 – Diagrama UML de interactiune intre obiectele cu care poate interactiona jucatorul |

### Buton si maneta

Butonul si maneta sunt elementele cu care jucatorul interactioneaza direct prin coliziune. Fiecare stocheaza in lista de obiecte de tip Trigger referinte catre una sau mai multe instante carora le va apela functia de activare in momentul in care propria lor functie de activare este apelata. Prin arhitectura pe care am conceput-o am pus in valoare regula de substitutie Liskov din suita de principii SOLID[[11]](#footnote-11) ale programarii orientate obiect, care presupune posibilitatea de a folosi orice clasa in locul unei clase parinte a acesteia, pastrand aceeasi functionalitate fara a implica modificari aditionale. Astfel, procesul de proiectare a nivelelor mi-a fost usurat substantial odata ce am creat cate un prefab[[12]](#footnote-12) pentru fiecare dintre aceste obiecte, urmand ca apoi sa stabilesc perechi de tipul “element\_declansator – element\_declansat” in care pot asigna declansatorului orice si oricati mostenitori ai clasei Trigger, cum ar fi usi ce se deschid sau inchid, platforme ce se deplaseaza intre doua puncte fixe sau lasere care se activeaza si dezactiveaza. Ambele parcurg lista denumita “triggers” si apeleaza functia TriggerFunction() pentru fiecare in parte, dar functionalitatea ce le deosebeste este una subtila, dar care amplifica posibilitatile de proiectare a puzzle-urilor; butonul (Fig. 18) actioneaza perechea sa, atat la apasare, cat si la eliberare, cu posibilitatea de a avea si un temporizator ce amana ridicarea sa din momentul in care jucatorul nu mai este in contact cu el, in timp ce maneta (Fig. 19) se poate schimba intre pozitiile pornit sau oprit, asemanator unui comutator.

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

### Laser

Laserul presupune un pretext ingenios si estetic de a adauga inca o restrictie, un impediment in procesul de rezolvare a puzzle-ului. Asemanator conceptului din *Portal* 2 prezentat in unul din capitolele anterioare, l-am realizat din 3 parti componente ilustrate in Fig. 20 : un obiect ce are atasata o componenta de tip LineRenderer[[13]](#footnote-13) si alte doua identice, oglindite, unul ce joaca rol de emitator, iar celalalt de receptor.

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

Am atasat obiectului si un collider pentru a putea inregistra momentul in care unul dintre caracterele controlabile de jucator intra in contact cu acesta, cu scopul de a declansa resetarea scenei, utilizatorul fiind prevenit de consecintele greselii. Intrucat am dorit sa am posibilitatea de a plasa emitatorul sau receptorul pe obiecte miscatoare, m-am lovit de problema ca marginile ce alcatuiesc colliderul atasat laserului sunt generate de Unity la instantierea obiectului, ramanand constante pe parcursul rularii. Drept consecinta, desi din punct de vedere vizual linia se extindea si se contracta corespunzator miscarii platformei de care era atasat, limitele nu se actualizau. Solutia pe care am gasit-o a fost de a atasa scriptul denumit LazerCollisionController din diagrama UML plasata la inceputul capitolului, prin intermediul caruia restabilesc dinamic, la fiecare cadru, cele patru puncte ale paralelogramului.

### Platforma miscatoare

Platformele miscatoare reprezinta o modalitate de transport a personajelor controlate de jucator, in special cel terestru, care permite ajungerea in anumite zone initial inaccesibile, printr-o miscare rectilinie, previzibila ce se bazeaza pe functia Lerp() din biblioteca Vector3. Aceasta are drept fundament matematic conceptul de interpolare liniara intre doua puncte care determina pozitia unui al treilea punct, date fiind primele doua si un procent de distantare fata de primul, plecand de la presupunerea ca exita un segment de dreapta ce le intersecteaza pe toate trei:

Considerand in formula anterioara variabilele x, a si b vectori, prin varierea valorii lui t, un scalar aflat in intervalul [0,1], putem simula glisarea lui x, pornind de la punctul initial a, pentru care t = 0, pana la destinatia b, pentru care t = 1. Pentru a obtine o miscare cursiva, initial accelerata si decelerata spre final, apelez metoda CalculatePlatformMovement() in functia FixedUpdate() pentru a afla noua pozitie la care trebuie sa translez obiectul la fiecare interval fix de timp, observabil in **ANEXA 3??**. Dupa cum am mentionat si in capitoul introductiv, am optat pentru aceasta functie a clasei MonoBehaviour, in locul celeilalte variante, Update(), pentru a ma asigura ca miscarea este una constanta, independenta de numarul de cadre pe secunda, deoarece acesta poate fi influentat de mai multi factori, cum ar fi specificatiile tehnice si puterea de procesare a componentelor sistemului pe care ruleaza utilizatorul jocul. Particularizand formula anterioara in situatia mea, pozitia curenta a platformei la momentul apelului joaca rolul variabilei a, iar destinatia este capatul traiectoriei, setat in inspector. Proprietatile miscarii sunt redate de modul de calculare a procentului t: il setez cu valoarea 0 la momentul initierii deplasarii, iar la fiecare iteratie adun la valoarea anterioara diferenta de timp dintre apelul curent si cel precedent, inmultit cu viteza pe care o doresc, totul divizat la distanta ramansa pana la punctul final. Odata aflata valoarea lui t, introdusa in functia Vector3.Lerp() impreuna cu plasarea curenta si destinatia, obtin o miscare placuta din punctul de vedere al experientei de joc deoarece are viteza maxima in jurul mijocului distantei de deplasare, iar spre capete prezinta o accelerare usoara dintr-o pozitie stationara, respectiv o decelerare lenta pana la oprire, ceea ce permite imbarcarea si parasirea relaxata a mijlocului de transport.

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

Rezolvare problema rigidbody kinetic -> modificare parinte perrry

### Usa

Usile ilustrate in fig Fig. 22 sunt o alta modalitate prin care m-am gandit sa restrictionez posibilitatile de miscare ale jucatorului, avand situatii cand au o orientare atat verticala, cat si orizontala. Motricitatea lor este asemanatoare cu cea a platformelor, cu diferenta ca, spre deosebire de acestea, parametrul ajustabil in inspector regleaza durata de timp necesara pentru inchiderea sau deschiderea completa in loc de viteza, pentru a spori gradul de ajustabilitate, in combinatie cu eventualul temporizator al butoanelor.

|  |
| --- |
| Fig. 22 - Usi |

## Caractere

Mecanica de baza a jocului consta in controlarea alternativa a doua caractere, dintre care unul este terestru, reprezentat de un ornitorinc agent secret, intitulat Perry, care are posibilitatea de a pasi pe platformele din spatiul bidimensional, de a sari sau ataca. Drona poate zbura, fiind un personaj adjuvant care desemneaza un ecran ce afiseaza in timp real un apel video cu Maiorul Monograma, seful agentiei secrete. Intrucat au relativ multe functionalitati comune, cum ar fi functiile de ascultare a input-ului, aplicarea fortelor pentru miscare sau preluarea/pasarea controlului, clasele ce controleaza sprite-urile mostenesc PlayerController (Fig. 23), avand astfel din nou avantajul de a ma putea referi la oricare dintre ei in interactiunea cu alte obiecte prin aceast parinte. De asemeanea, avand in vedere proprietatea de unicitate pe care o au amandoi in contextul jocului, am implementat clasele urmand Singleton Design Pattern ceea ce mi-a facilitat procesul de creatie a nivelelor, astfel incat pot accesa din orice loc instantele lor prin intermediul functiei statice GetInstance(), fara sa mai fiu nevoit sa asignez manual o referinta fiecarui element in parte care interactioneaza in scena cu ei. Un exemplu relevant de astfel de situatie este in cazul gestionatorului de control, care are simpla intrebuintare e a asculta comanda de comutare a personajelor lansata de jucator prin tasta Shift, urmand ca apoi sa dezactiveze si sa activeze clasele corespunzatoare ce mostenesc PlayerController, astfel incat exact una dintre ele este intotdeauna activa la un moment dat.

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

### Perry

Perry este caracterul principal asupra caruia jucatorul are controlul la inceputul fiecarui nivel. Avand in vedere ca am conceput nivelele sa aiba o orientale verticala, inceputul fiind mereu in partea de jos a ecranului, iar finalul in partea de sus, scopul principal este intotdeauna crearea unui traseu prin care el sa poata ajunge la platforma finala, din moment ce drona o poate accesa mult mai usor, avand capacitatea de a zbura. Obiectul are atasat scriptul GroundPlayerController din Fig. 23 care se ocupa de interactiunea cu mediul si miscarea aceastuia in functie de comenzile trimise de utilizator. Prin atributele sale pot gestiona viteza de deplasare, numarul de sarituri consecutive, puterea saltului sau punctele de viata.

Animatiile, efectele speciale si aspectul unui joc au un rol important in experienta utilizatorului si nu pot fi neglijate, altfel poate aparea confuzia sau lipsa de interes, ceea ce poate duce mai departe la abandonare. Acestea fiind spuse, am acceptat provocarea de a experimenta si acest capitol al dezvoltarii jocurilor video si am realizat personal aproape toate componentele vizuale si dinamice in Adobe Illustrator. Exceptand dezavantajul ca este extrem de costisitor din punct de vedere temporal, consider ca am avut multe de castigat intrucat necesita un nivel de atentie la detalii sporit, cum ar fi potrivirea culorilor sau a stilului de desenare, dimensiunea obeictelor relativa una la cealalta si setarile de importare in Unity sau alinierea perfecta a cadrelor animatiilor din Fig. 24.

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

Unity dispune de un sistem de controlare a animatiilor extrem de intuitiv; clasa GroundPlayerController contine o referinta catre o componenta de tip Animator atasata obiectului prin intermediul careia pot trimite din cod semnale catre un gestionator de tipul Animation Controller[[14]](#footnote-14), in care am definit animatiile si conditiile necesare pentru a face o tranzitie intre ele.

Schema din Fig. 24 arata toate conexiunile pe care le-am realizat intre toate starile in care se poate afla Perry. In mod implicit, fara stimuli externi, el va rula la infinit animatia “idle”; o tranzitie din starea generala “Any state”, generata automat si obligatoriu de Unity, se poate realiza in orice moment daca este indeplinita conditia de tranzitie. Spre exemplu, in aceasta figura am selectat trecerea spre animatia de atac, care se poate observa in fereastra Inspector din dreapta, care are loc doar in momentul in care variabila isAttacking are valoarea adevarat; in momentul in care jucatorul apasa click, prin simplul apel al unei functii o setez, iar Unity se ocupa de afisarea animatiei, prin schimbarea sprite-ului din componenta SpriteRenderer2D, la fiecare cadru.

|  |
| --- |
| Fig. 25 – Schema de tranzitie intre animatiile lui Perry |

### Maiorul Monograma

Conceptul care m-a fascinat cel mai mult la jocul FireBoy & WaterGirl este ideea de cooperare intre doua entitati controlate de o singura minte care are o privire de ansamblu asupra intregii camere ce reprezinta nivelul curent. Avand acest punct de plecare, am eliminat din ideea originala controlul lor simultan si acele acumulari de lava sau apa care permit sau nu trecerea si am adus o nota personala prin crearea unei drone ce poate zbura, in loc de doua personaje identice. Efectul de zbor l-am obtinut prin simpla eliminare a efectului gravitatiei asupra acesteia si aplicand o forta in directia de deplasare introdusa de jucator. Pentru a simulara miscarea de franare spre oprire, am marit factorul fortei de frecare cu aerul care incetineste progresiv corpul in momentul in care nu mai sunt forte aplicate asupra sa.

In general, coliziunea intre caractere este dezactivata, pentru a nu exista situatii in care s-

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

ar putea incurca unul pe cealalt. Singura situatie in care ele pot interactiona este cand Perry se afla deaspura dronei ca in Fig. 26, astfel avand posibilitatea de a stationa pe ea, facand accesibile animalului terestru anumite zone initial blocate datorita inaltimii. Am reusit sa implementez acest comportament cu ajutorul componentei Platform Effector 2D atasate unui obiect copil dronei, ce se ocupa in mod special de restrictionarea posibilitatii de coliziune doar pentru anumite grupuri de obiecte, doar din anumite unghiuri de incidenta. O problema interesanta pe care am intampinat-o este faptul ca ambele fiind obiecte dinamice, cu masa, care se supun regulilor fizicii simulate de Unity, in momentul coliziunii cadeau spre pamant datorita gravitatiei aplicate asupra lui Perry. Solutia gasita a fost sa setez corpul dronei in componenta RigidBody2D de tip “kinematic” pe durata coliziunii, ceea ce dezactiveaza efectul intregului modul de simulare a fizicii lumii reale asupa sa. Avand in vedere ca aceasta setare se intampla prea tarziu in momentul inregistrarii coliziunii, iar asupra obiectul zburator avea loc o translatie semnificativa pana sa intre in vigoare modificarea, am atasat clasa POnPCollisionController din diagrama UML. Rolul ei este de a prezice si realiza configuratiile necesare inainte de contact, folosind functia Physics2D.OverlapCircle() care, avand date ca parametrii o pozitie in scena, o raza de actiune si un obiect de tip LayerMask[[15]](#footnote-15), returneaza o valoare booleana care informeza daca exista o suprapunere intre cercul definit de primii doi parametrii si orice obiect apartinand stratului definit de cel de-al treilea. Asadar, apeland metoda la fiecare cadru si verificand daca directia de miscare a lui Perry are componenta y negativa, semnificand o cadere, in momentul tangentei cu un cerc aflat deasupra dronei, pot anticipa coliziunea celor doua.

In marea majoritate a timpului, cele doua personaje principale vor avea sarcini diferite, in zone diferite ale nivelului. Cu toate aceastea, sunt momente in care se deplaseaza pe acelasi drum, in aceeasi directie, iar datorita controlului alternativ, fiecare dintre ele trebuie duse pe rand in punctul respectiv. Pentru a eficientiza procesul, am utilizat o biblioteca intitulata AStarPathfindingProject ce simuleaza si implementeaza algoritmul de cautare A\*. Accesta este des folosit in jocuri in crearea inteligentei artficiale, datorita eficientei sale de furnizare a celui mai scurt drum, dat fiind un graf pentru care exista cel putin un lant de la punctul de plecare pana la destinatie. Specific acestui algoritm de cautare este faptul ca desi la baza sa este un algoritm de tip cautare in latime, frontiera acestuia nu se extinde in mod consistent in toate directiile, ci prioritizeaza caile care par sa duca mai aproape de locul dorit. Face acest lucru folosind un sistem de alegere a urmatorului nod ce combina prioritizarea drumului cu cost minim cu o euristica ce estimeaza costul drumului ce urmeaza a fi descoperit din punctul curent pana la destinatie. Odata ce am atasat clasa numita Pathfinder unui obiect plasat in scena, proiectul imi genereaza automat un graf alcatuit din noduri interconectate ce inglobeaza toate drumurile posibile intre toate zonele accesibile, odata ce am setat un obiect de tip LayerMask prin care specific care sunt obstacolele. In Fig. 27 se poate observa cum platformele, usile, manetele, bila si cutia sunt ocolite, iar in zona albastra este calculat cel mai scurt drum de la drona la Perry; odata aflata aceasta cale, clasa FollowController atasata obiectului zburator aplica forta necesara deplasarii de la un nod la altul, pana ajunge la destinatie.

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

# Ghid de utilizare

1. Cut the Rope – joc video puzzle pentru Android bazat pe reguli ale fizicii ce necesita formularea si revizuirea unui plan de actiune, imaginatie si invatare din greseli anterioare prin testare si esuare. [↑](#footnote-ref-1)
2. Gaura de vierme - Reprezinta o metodă teoretică prin care două zone îndepărtate din Univers sunt unite între ele printr-o scurtătură. În consecinţă, din punct de vedere teoretic, permite calatoria aproape instantanee dintr-un loc în altul. [↑](#footnote-ref-2)
3. Lumosity – Program computerizat de antrenare a abilitatilor cognitive de baza precum memoria, atentia, viteza de procesare, flexibilitatea mentala, orientarea spatiala si rezolvarea de probleme [↑](#footnote-ref-3)
4. Adobe Flash – Produs software folosit pentru a produce animatii, aplicatii web si desktop, jocuri si aplicatii pentru mobil si 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 solutie bine documentata la o problema des intalnita in programea orientata obiect ce restrictioneaza instantierea la un singur exemplar a unui obiect ce apartine unei clase. [↑](#footnote-ref-8)
9. Software Development Kit – SDK este o multime de unelte software si programe folosite de dezvoltatori pentru a crea aplicatii pentru anumite platforme. [↑](#footnote-ref-9)
10. Metode HTTP – Functii special concepute pentru a realiza comenzi de stergere, citire, creare si 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 sablon ce poate fi salvat in Unity pentru a recrea, configura si adauga in scena un obiect, mentinandu-i toate proprietatile precum componente, valori, si elemente copil. [↑](#footnote-ref-12)
13. LineRenderer – Componenta atasabila obiectelor din Unity ce primeste un vector de puncte si deseneaza o linie intre fiecare, avand posibilitati multiple de configurare. [↑](#footnote-ref-13)
14. Animation Controller – Panou ce permite aranjarea si mentinerea [↑](#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)