UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

CTRL + ALT + STEVE

Absolvent
Iustin Aurelian Mitu

Coordonator științific
Lect. dr. Ștefan Popescu

București, iulie 2025

Rezumat

Această lucrare prezintă dezvoltarea unui joc 2D realizat în Unity, care folosește mai multe tehnologii la bază precum Blockly, Flask, Skulpt, etc, care sunt prezentate pe larg în cadrul capitolului de arhitectură. Acest joc are scop atât educativ cât și de divertisment, principalul țel al acestuia fiind dezvoltarea gândirii logice și algoritmice a utilizatorului. Publicul țintă al aplicației realizate sunt copiii de orice vârstă care cunosc vocabularul de bază al limbii engleze și care își doresc să se distreze și totodată să își facă o viziune de bază asupra programării în limbajul de programare Python. Jocul nu este unul obișnuit, deoarece modul de interacționare dintre utilizator și aplicație se face prin unirea de blocuri, care generează automat cod, vizibil utilizatorului, astfel încurajându-se dezvoltarea gândirii algoritmice.

Abstract

This paper presents the development of a 2D game made in Unity, which leverages several underlying technologies, such as Blockly, Flask, Skulpt, etc., which are presented in detail in the architecture chapter. This game has both an educational and entertainment purpose, its main goal being the development of the user's logical and algorithmic thinking. The target audience of the developed application is children of all ages with a basic understanding of English vocabulary and who want to have fun and at the same time gain a basic understanding of programming in the Python programming language. The game is not an ordinary one, as the interaction mode between the user and the application is done by connecting blocks, which automatically generate visible code for the user, thus encouraging the development of algorithmic thinking.

Cuprins

Capitolul 1

Introducere

1.1 Contextul general al lucrării

Din punct de vedere al domeniului, lucrarea se încadrează în dezvoltarea de jocuri. Acest domeniu se află în plină dezvoltare, fiind o ramură care a apărut încă din anii 70. Dezvoltarea acestei ramuri ale programării se reflectă în calitatea și experiența oferite de jocuri, ambele aspecte progresând de-a lungul anilor.

Printre primele jocuri apărute se numără **Spacewar!**, **Computer Space** și **Pong**. În anii 80 încep să apară consolele și computerele personale, iar între anii 1990 - 2000 această industrie are parte de o creștere semnificativă. Creșterea a avut loc în urma mai multor factori precum, tranziția de la grafica 2D la cea 3D (au apărut jocuri precum **Virtua Fighter** (1993) și **Super Mario 64** (1996)), dar și apariția noilor console precum Sony PlayStation (1994) sau Nintendo 64 (1996).

Un alt factor ce a contribuit la evoluția industriei de jocuri în aceeași perioada a fost diversificarea genurilor de jocuri. De exemplu, existau jocuri de tip shooter la persoana întâi cum ar fi **Doom** (1993), jocuri de strategie în timp real precum **Command & Conquer** (1995), sau jocuri de tip fantasy precum **Ultima Online** (1997). În aceeași perioadă încep să apară și primele motoare de jocuri, unele dintre cele mai importante fiind **Quake engine** (1996), **Unreal Engine** (1998), iar în 2005 apare **Unity**.

Până în anul 2024 această industrie a fost estimată la o valoare de aproximativ 221 miliarde de dolari, conform [4], și se estimează să ajungă la o valoare de 424 miliarde până în anul 2033.

1.2 Motivația alegerii temei

Alegerea temei a fost dificilă, deoarece am vrut să combin utilul cu plăcutul, mi-am dorit să descopăr tehnologii noi și să reușesc să proiectez o arhitectură cu un mod de lucru bine pusă la punct. Totodată mi-am dorit ca la finalizarea acestui proiect să rămân cu

un produs folositor.

Principalul motiv pentru care am ales această temă a fost dorința de a realiza un proiect complex care îmbină mai multe tehnologii și totodată mai multe limbaje de programare. Pe parcursul facultății am acumulat experiență în domeniul dezvoltării jocurilor și în tehnologii precum **Unity**, **Unreal Engine 5**, cod scris de la zero în **Python** cu ajutorul [5], sau chiar **C++** cu ajutorul **SFML** [8]). Experiența a provenit și din fostele locuri de muncă, și odată cu ele și pasiunea de a lucra cu copii. Mi-am dorit ca piatra de temelie a lucrării să fie ceva cunoscut, în care îmi place să-mi desfășor activitatea, dar care este suficient de complexă să pot adăuga tehnologii noi. Motivul pentru care am ales **Unity**, și nu alt motor de jocuri, a fost strict o preferință personală.

În continuarea acestui motiv, tema a fost aleasă ținând cont inclusiv de viitorul meu academic. Consider că această abilitate de a îmbina tehnologii este o abilitate esențială pentru un inginer software. Îmi doresc să continui studiile la master-ul de Inginerie Software din cadrul Facultății de Matematică și Informatică din Universitatea București. În prezent, lucrez ca Inginer Software și am fost pus de nenumărate ori în ipostaza de a trebui să realizez o modificare sau o implementare a unei tehnologii noi, tehnologie pe care de cele mai multe ori nu o cunoșteam. Procesul de familiarizare cu o aplicație sau cu modul de lucru al acesteia, cercetarea din spate a tehnologiilor și abilitatea de a înțelege un proiect fac parte din viața unui Inginer Software. Consider că lucrarea mea de licență îmbină destul de bine toate aceste aspecte și totodată este un proiect potrivit pentru a fi prezentat la interviul din cadrul master-ului de Inginerie Software.

1.3 Scopul și obiectivele lucrării

Având în vedere ca mi-am dorit ca la finalul proiectului să rămân cu un produs folositor, jocul final se află în directă corelație cu această afirmație. Secțiunea curentă răspunde la întrebările "Cu ce mă ajută acest joc? De ce este folositor?".

Răspunsul se află în tema educațională adoptată de această lucrare. Așa cum am menționat și în rezumat, publicul țintă, sau utilizatorii, sunt copiii de orice vârstă care cunosc vocabularul de bază al limbii engleze, care își doresc să se distreze și totodată să își facă o viziune de bază asupra programării în limbajul de programare **Python**. Prin urmare, scopul lucrării este să ofere utilizatorului o viziune generală asupra informaticii și a programării, în cazul acesta, folosindu-ne de limbajul **Python**. Precis, utilizatorii ar trebui să înțeleagă mai ușor cum funcționează (în special) variabilele, structurile de decizie (if - else) și structurile repetitive (for, while).

Jocul poate fi folosit inclusiv ca material didactic, dar poate fi jucat și singur de către un utilizator, fără a avea nevoie de un profesor sau o persoană specializată care să asiste.

Pe parcursul jocului am implementat mai multe referințe informatice, care au scopul de a familiariza utilizatorul cu diferite subdomenii ale programării. Spre exemplu am

explicat în diferite nivele conceptul de bug, conceptul de conflict, conceptul de virus, etc. Pe lângă explicația accesibilă a fiecărui concept, aplicația oferă și o metodă interactivă de a lucra cu aceste concepte. Spre exemplu, există nivele în care există fișiere corupte, și pentru a le detecta, utilizatorul este nevoit să le scaneze întâi, folosind blocul potrivit. Un alt exemplu sunt entitățile din joc care reprezintă blocuri de date criptate, iar pentru a le decripta este nevoie ca utilizatorul să aplice forță brută. În practică trebuie să folosească un anumit bloc de 5 ori, fapt ce simbolizează o căutare exhaustivă și totodată încurajează utilizatorul să folosească o structură repetitivă.

Un alt mod prin care jocul sugerează subtil folosirea de structuri repetitive sau structuri de decizie este construcția hărților. Concret există porțiuni de hartă unde se pot observa anumite modele ce se repetă, astfel, utilizatorului îi poate veni ideea sa folosească o structură repetitivă, reducând numărul de acțiuni de tip drag and drop.

Pe lângă aceste aspecte, jocul prezintă și codul creat dinamic de către utilizator. Prin dinamic, mă refer la faptul că este actualizat la fiecare modificare a codului. Scopul aplicației nu este să învețe un utilizator limbajul **Python**, acesta a fost mai mult ales pentru a conecta metoda interactivă de a plasa blocuri, cu partea de cod puțin mai teoretică din spate. **Python** este un limbaj de programare ușor de înțeles, și a fost ales datorită sintaxei ușoare, dar în practică putea fi ales absolut orice limbaj de programare ce are în spate o paradigmă imperativă.

Capitolul 2

Preliminarii

2.1 Povestea jocului

Povestea jocului este inspirată din desenele animate pentru copii care au fost difuzate la televizor în perioada copilăriei mele, aproximativ 2003 - 2015. Câteva mențiuni foarte clare sunt **Adventure Time - Guardians of Sunshine** și **Regular Show - Exit 9B**. În ambele episoade, personajele principale sunt transpuse într-un joc video și sunt nevoite să îndeplinească anumite sarcini pentru a trece peste o anumită situație din viața reală. Majoritatea desenelor animate din anii 2010 au câte un episod asemănător. Din punctul meu de vedere, aceste episoade erau cele mai interesante deoarece deschideau extrem de multe oportunități în universul personajelor, dar și pentru că era nevoie să fie schimbat stilul animației pentru un singur episod, ceea ce îl făcea să fie diferit. În Figura 2.1 se poate observa o perspectivă generală asupra graficii jocului.

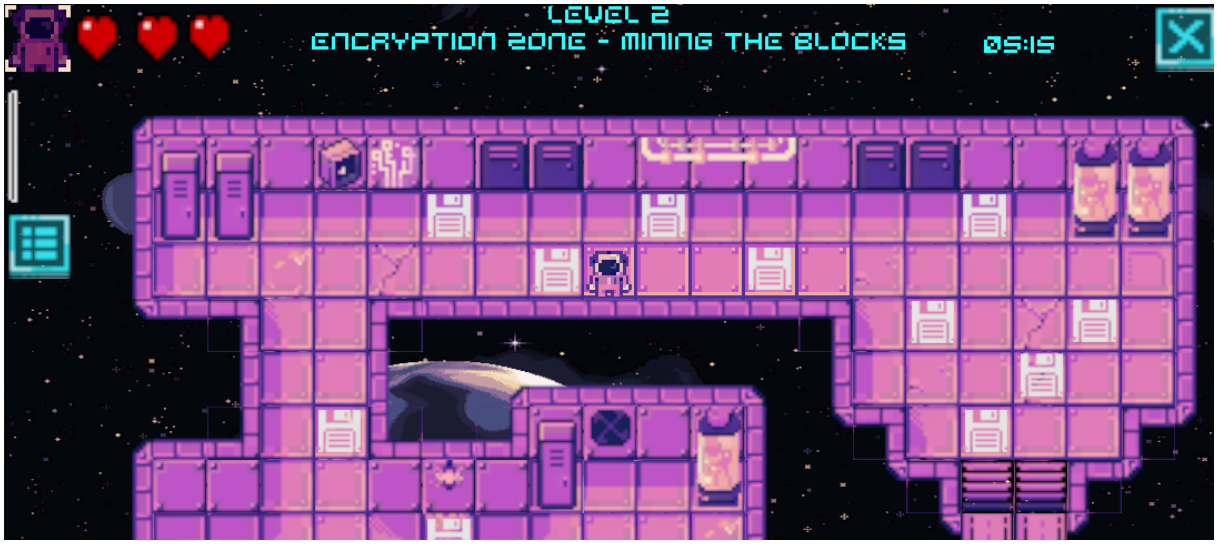

Figura 2.1: Mediul jocului

Povestea din acest joc este una asemănătoare. La nivel de expozițione, tu ești un

copil obișnuit, care știe câte ceva despre calculatoare, și ai un prieten pe nume Steve, care devine imediat personajul principal din această poveste.

Într-o zi, încercați împreună, fără succes, să rezolvați o eroare. După multe ore de chin, îți vine ideea să îl transformi pe Steve într-un șir de biți, și să îl trimiți în calculator pentru a rezolva eroarea, folosind un dispozitiv magic.

Neștiind atât de multe despre programare, Steve nu știe să navigheze singur prin calculator, deci scopul tău este să îl controlezi pentru a repara împreună problema. Și cum altfel să îl controlezi, dacă nu prin scris de cod, care pentru ușurința utilizatorului este transformat în a uni bloculețe de programare.

Împreună cu Steve, trebuie să parcurgeți diferite zone ale calculatorului, să eliminați bug-uri, să scanați fișiere corupte, să decriptați date, etc. pentru a scăpa de eroare. În Figura 2.2 se pot observa inamicii care trebuie eliminați.

Figura 2.2: Inamici

2.2 Desfășurarea jocului

Interacțiunea pe care o are utilizatorul cu jocul și mecanicile acestuia diferă de un joc normal. De obicei, într-un joc te miști folosind tastele W, A, S, D, săgețile, sau poate mouse-ul. În acest joc, utilizatorul este nevoit să creeze cod pentru a-l mișca pe Steve. Crearea codului se face prin glisarea blocurilor de cod. Blocurile sunt împărțite în următoarele categorii:

- Player – blocuri pentru mișcare, atac, scanare etc.
- Logic – structuri de decizie, operatori logici
- Loops – buclele `for` și `while`

- Math – operații matematice
- Variables – blocuri pentru variabile și manipularea lor

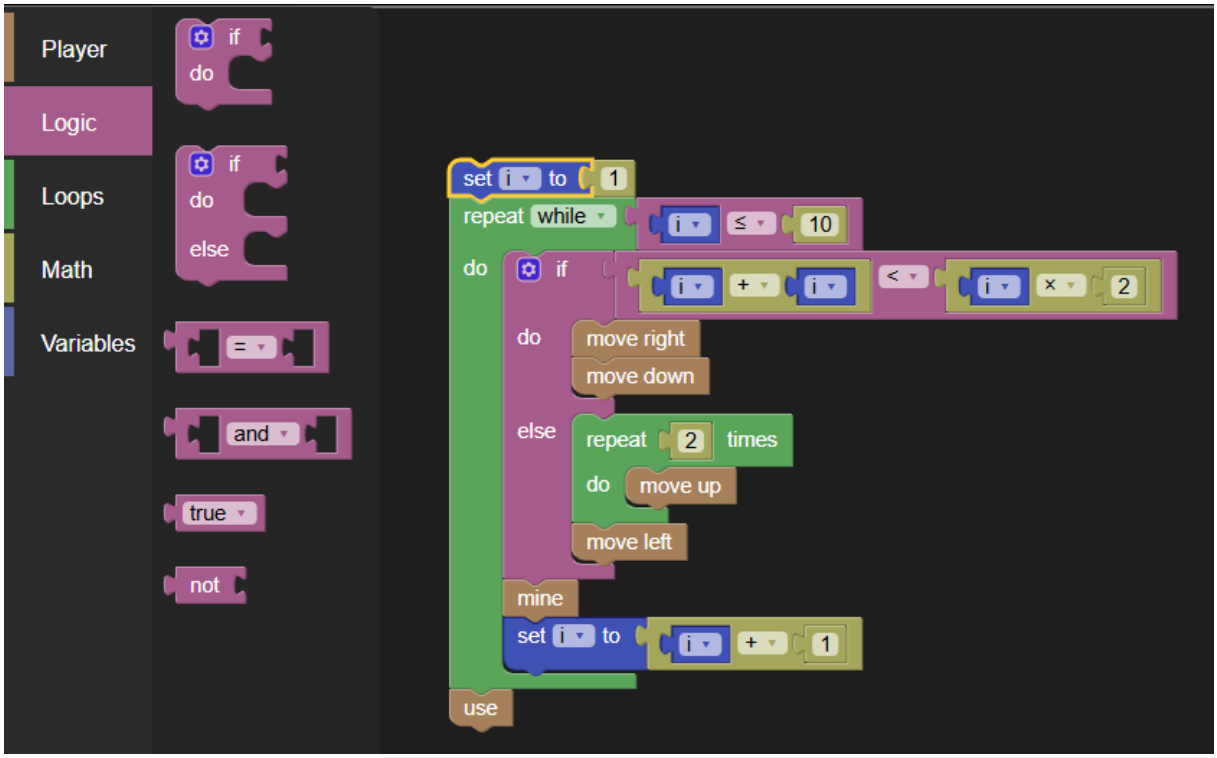

Figura 2.3: Împărțirea pe categorii ale bloculeților

După ce utilizatorul glisează și îmbină aceste bloculeț, apăsă pe butonul **Generate Code**. Acest buton va prelua bloculeț și va genera codul **Python** corespunzător.

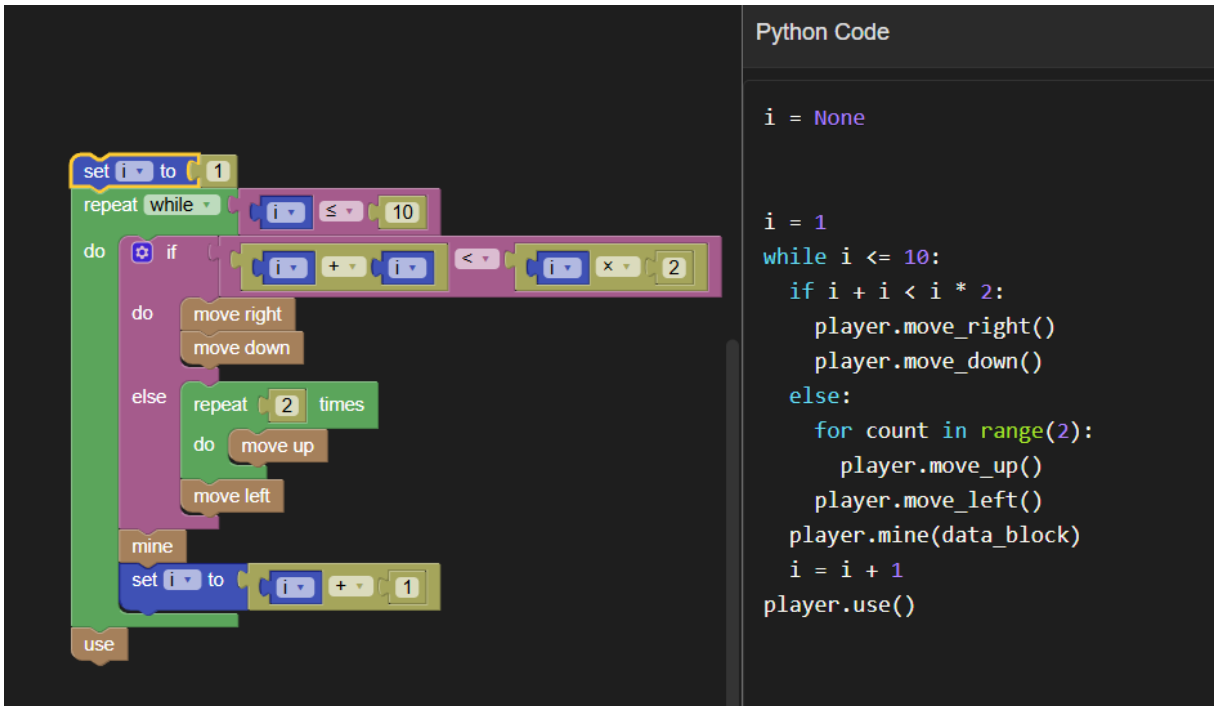

Figura 2.4: Codul Python generat

La final, după ce codul este generat, utilizatorul apăsă pe butonul **Run Code**, iar Steve va executa comenzile generate.

2.3 Tehnologiile folosite

În această secțiune voi detalia fiecare tehnologie care a fost folosită în cadrul acestei lucrări. De asemenea, sunt incluse limbajele de programare folosite.. Implementarea acestor tehnologii, și codul proiectului poate fi găsit la următoarele link-uri:

Aplicație Web

Proiect Unity

2.3.1 Unity

Unity [9] este un motor de jocuri gratis apărut în 2005, și este unul dintre cele mai cunoscute motoare, alături de **Unreal Engine**. Cu ajutorul acestuia se pot dezvolta jocuri 2D, 3D, VR, etc. Motorul dispune inclusiv de editoare vizuale interactive. Popularitatea motorului apare datorită faptului că este **cross-platform**, are o interfață grafică intuitivă, fizica integrată, sistem de particule, etc. Partea de scripting se face în **C#**, un limbaj destul de versatil și modern. **Unity** dispune de o comunitate foarte largă. Acestea sunt câteva dintre motivele pentru care am ales această tehnologie. Câteva jocuri cunoscute care au fost făcute în Unity sunt: **Among Us** (2018), **Monument Valley** (2014), **Beat Saber** (2018), etc.

2.3.2 Blockly

Blockly [1] este o bibliotecă **open-source** creată de **Google** cu ajutorul căreia utilizatorii pot construi cod folosind blocuri programabile. Este folosit la educația informatică deoarece oferă o interfață prietenoasă în care este ușor să creezi cod prin glisarea blocurilor de programare. Am ales această tehnologie datorită personalizabilității sale, care permite crearea oricărui tip de blocuri, și pentru faptul că este un editor de cod integrabil, putând fi inclus în pagini web. Un alt avantaj pe care îl are **Blockly** este exportarea de cod real, el poate genera cod în mai multe limbaje de programare, precum **JavaScript**, **Python**, **Lua**, **Dart**, **PHP**, etc. Acesta oferă accesibilitate utilizatorilor fără experiență de programare, și reduce problemele de sintaxă pe care le poate avea un programator la început de drum. Un exemplu concret în care este folosit este **Scratch**.

2.3.3 Flask

Flask [6] este un **framework** scris în **Python** care are la bază 2 componente, **Werkzeug** (folosit la manipularea cererilor HTTP) și **Jinja2** (folosit pentru template-uri HTML), astfel este un candidat foarte bun pentru **backend-ul** unei aplicații. Fiind scris în **Python**, oferă simplitate și flexibilitate dezvoltatorului la capitolul implementare. Această tehnologie a fost aleasă pentru a gestiona cererile API. Câteva exemple de utilizare a acestui framework din industrie sunt: **Netflix** în micro-servicii de configurare, **Reddit** în componente de automatizare, **Pinterest** în dezvoltarea unor servicii interne sau chiar **MIT** în proiecte educaționale.

2.3.4 Skulpt

Skulpt [2] este un **interpretor** de **Python** scris în **JavaScript**, care permite rularea codului direct în browser, și acesta a fost motivul principal pentru care l-am ales. Acesta este un proiect **open-source** care include pachete esențiale din **Python** precum math, random, etc. Avantajul principal al acestuia este că rulează în totalitate în browser, deci nu necesită componente din zona de server. Un avantaj este faptul că se poate integra foarte ușor cu tehnologii web precum **HTML**, **CSS**, **JavaScript** sau biblioteci de UI precum **Blockly**. Este des întâlnit în aplicații educaționale precum **edX**, **CodeSkulptor**, **trinket.io** sau în prototipuri în care este necesară execuția de cod **Python** într-un browser.

2.3.5 Prism

Prism [12] este o librărie **open-source** de **JavaScript** care este folosită pentru a evidenția sintaxa codului în mai multe limbaje de programare, precum **Python**, **JavaScript**, **C++**, etc. Această librărie are o dimensiune foarte redusă și o viteză de procesare

mare. **Prism** oferă funcționalități precum evidențierea statică a codului, suport pentru linii numerotate, personalizarea evidențierilor prin clase de **CSS** și chiar completare automată a codului cu ajutorul unor plugin-uri suplimentare. În general, acesta este folosit pentru aplicații web interactive pentru editarea codului și blog-uri sau site-uri tehnice precum **MDN Web Docs** sau **Stack Overflow**. Am ales această tehnologie pentru a evidenția și a da un aspect mai plăcut codului de **Python** generat.

2.3.6 Embedded Browser

Embedded Browser [3] este un pachet disponibil în **Unity Asset Store**, dezvoltat de **Zen Fulcrum LLC** care permite integrarea unui browser într-un proiect **Unity**. Acest pachet este necesar deoarece **Unity** nu suportă nativ cod **JavaScript**, deci nu se poate integra nativ un browser. Cu toate acestea, există totuși o altă variantă prin care se poate integra un browser în **Unity**, mai exact prin **WebView**. Din nefericire, **WebView** nu este **cross-platform**, și poate funcționa doar pe **macOS**. **Embedded Browser** oferă inclusiv interacționarea cu obiectele **DOM**. Afișarea paginii se face în timp real. Interacțiunea dintre **aplicația web** și **Unity** se face de fapt prin apeluri de funcții. Un exemplu de aplicație **open-source** care foloseste **Embedded Browser** este **WAX Unity SDK**, proiect care permite autentificarea și semnarea tranzacțiilor direct într-o aplicație **Unity**, folosind un portofel digital.

2.3.7 Python

Python [7] este unul dintre cele mai populare limbaje de programare. Caracteristicile de bază ale acestui limbaj sunt: sintaxa simplă, ceea ce îl face ideal pentru programatorii începători, interpretat și portabil, deci poate fi rulat direct fără să necesite compilare și este un limbaj multi-paradigmă (permite programare procedurală, orientată pe obiecte și funcțională) făcându-l astfel disponibil pentru o gamă variată de proiecte. Din punctul meu de vedere, avantajul principal al acestui limbaj este reprezentat de bibliotecile extinse făcându-l astfel disponibil în foarte multe domenii (precum dezvoltarea de aplicații web, inteligență artificială, analiză de date, automatizare, dezvoltare de jocuri, etc.). Un alt avantaj masiv este comunitatea foarte numeroasă și activă, oferind astfel o documentație excelentă și multe resurse educaționale. Câteva exemple de aplicații care folosesc **Python** sunt: **Instagram** a folosit **Python** pentru **Backend** în **Django**, **Spotify** utilizează **Python** pentru analiza datelor, machine learning și servicii de **Backend**, **YouTube** a folosit și a dezvoltat aplicația la început în **Python** pentru a accelera dezvoltarea platformei, etc.

Capitolul 3

Arhitectura și modul de lucru

La nivel general, proiectul poate fi împărțit în 3: Blockly, Aplicația Web și Unity. În această secțiune voi detalia cum interacționează aceste componente între ele și care este modul de lucru din spatele aplicației. Pentru claritatea explicațiilor scrise, putem începe prin prezentarea figurii 3.1 care prezintă interfața grafică.

Figura 3.1: Interfața grafică

Așa cum am precizat în capitolul precedent, **Blockly** este o bibliotecă **open-source** creată de **Google**. Cu ajutorul acestei componente, utilizatorul creează bucăți de cod necesare pentru a avansa într-un nivel. **Blockly** oferă posibilitatea de a face interactiv acest proces de creare de cod prin interfața minimalistă și prietenoasă. Utilizatorul poate glisa bloculețe, le poate îmbina ca pe niște piese de puzzle, poate da zoom-in și zoom-out și totodată poate vedea schimbările în codul de **Python** în timp real. Practic la fiecare modificare, sau la orice mișcare, script-ul de **JavaScript** reactualizează codul de **Python**. Un alt avantaj pe care îl oferă **Blockly** este că poate fi integrat într-o pagină web. Un alt aspect foarte important, este că **Blockly** permite crearea de bloculețe personalizabile, și

chiar suprascrierea celor deja existente. Concret, în proiectul meu, am creat mai multe bloculețe specifice acțiunilor jucătorului (cele din secțiunea **Player**), și am suprascris bloculețele specifice structurilor de decizie pentru a se încadra în cromatica proiectului.

Următoarea componentă este **Aplicația Web**. Aceasta integrează **Blockly** și este structurată astfel: componenta de **Backend** rulează pe **Flask**, iar componenta de **Frontend** folosește **HTML**, **CSS** și **JavaScript**.

Componenta de **Backend** este responsabilă de cererile **API**. Totodată, aceasta poate fi extinsă pentru o aplicație web care să ruleze un astfel de joc direct în browser. Spre exemplu se poate face o cerere **API** pe ruta **/run**, și server-ul va prelua codul de **Python** generat de **Blockly**, și îl va rula sau va întoarce un **JSON** cu toate comenzile care trebuiesc executate de către caracter. Acest **/run** este implementat într-un mod foarte minimalist și are strict rol exemplificativ pentru cum se poate extinde proiectul.

Componenta de **Frontend** este locul unde se întâmplă magia. Structura paginilor web este făcută cu **HTML**, stilizată cu **CSS**, iar partea dinamică este gestionată de **JavaScript**. Atunci când este apăsat butonul de **Run Code**, se invocă o funcție creată în **C#** din proiectul de **Unity** numită **takeActionList()** și are ca parametru o listă de șiruri de caractere. Pentru a înțelege cum interacționează **Aplicația Web** cu **Unity** este necesar să înțelegem cum este creat acest parametru.

La apăsarea butonul de generare, este apelată o funcție care verifică dacă utilizatorul a făcut modificări în blocuri (deci posibil a făcut modificări și în cod). În caz afirmativ, codul nou creat este preluat și este evidențiat în pagina web cu ajutorul bibliotecii **Prism**.

Următorul pas este **preinterpretarea codului**. Pentru eficiență și ușurință, utilizatorului îi este afișată doar o parte de script, cea intuitivă. De fapt, înainte de interpretarea codului (care este preluat ca un șir de caractere) este inserat un alt script scris manual, tot în **Python**. Acest script inserat nu face nimic altceva decât să creeze o clasă numită **Player**, să inițializeze o variabilă de tip listă de șiruri de caractere (pentru acțiuni), și să inițializeze un obiect de tip **Player** numit **player**. Metodele din această clasă, au același nume cu bloculețele customizabile menționate anterior în această secțiune, adică cele specifice acțiunilor jucătorului. Aceste funcții nu fac nimic altceva decât să adauge la lista de acțiuni, un șir nou de caractere de tipul specific acțiunii apelate. Un exemplu clar este: metoda **move left** doar adaugă la lista de acțiuni **actions** șirul de caractere **moveLeft**. Acest cod inserat împreună cu codul generat de utilizator este concatenat, și apoi trece la etapa de **interpretare**.

Interpretarea codului se realizează cu ajutorul **Skulpt**. Acest proces are 2 etape, prima etapă se ocupa de configurarea mediului de interpretare a codului, și cea de a doua etapă se ocupa de interpretarea propriu-zisă a codului, și preluarea anumitor valori globale. Concret, cele doua etape lucrează împreună pentru a prelua valoarea variabilei **actions** de la finalul interpretării. Această valoare este mai apoi atribuită unei variabile **JavaScript**. Mai apoi, această variabilă este transmisă ca parametru la **Unity** la apăsarea butonului

de **Run Code**, așa cum am menționat anterior.

În continuare totul se simplifică. Trecerea către **Unity** este făcută de către **Embedded Browser**. Acest pachet este absolut necesar deoarece face posibilă comunicarea dintr-o **Aplicație Web** și **Unity** prin apeluri de funcții. **Unity** nu rulează nativ **JavaScript**, din acest motiv a fost necesar acest artificiu.

În final, lista de acțiuni ajunge în **Unity** într-un format **JSON** destul de complex. Extragerea efectivă a acțiunilor este făcută cu puțin **Regex**, iar apoi fiecare acțiune este rulată pe rând de către funcția specifică.

Arhitectura din **Unity** este una complexă. Principalele elemente sunt obiectele, componentele, scenele și script-urile. Un obiect reprezintă o entitate de bază în **Unity** și poate reprezenta orice. Acestei entități i se atribuie componente. Aceste componente oferă funcționalitate entității precum masă, gravitație, culoare, textură, un comportament specific în anumite situații, etc. Scenele reprezintă o colecție de astfel de obiecte și de obicei o scenă reprezintă un nivel, sau un meniu. Script-urile sunt tot niște componente, scrise de obicei în **C#** și oferă flexibilitate dezvoltatorului pentru a controla anumite aspecte ale jocului, sau pentru a crea un anumit comportament unui obiect.

Unity oferă foarte multă flexibilitate deoarece are integrate inclusiv editoare și mecanisme prin care se pot modifica sau crea animații, efecte sonore, particule, etc. În această aplicație am folosit de exemplu animații pentru caracter. Pentru a realiza asta, am fost nevoit sa creez un **Animator**, un **Animator Controller** și câteva animații. **Animator** este obiectul animat (de exemplu, caracterul principal) iar **Animator Controller** este componenta care se ocupă de tranzițiile dintre animații. Aici se definesc tranziții pe baza unor condiții (și a unor parametrii controlați din script-uri). Pentru a generaliza, așa funcționează în general procesul de dezvoltare a unui joc în **Unity**, se creează un obiect, se adaugă componentele necesare, se definesc niște script-uri pentru comportamentul obiectului respectiv, iar apoi se ajustează parametrii în funcție de nevoile aplicației. Țin să menționez că toate imaginile folosite au fost preluate de pe site-ul oficial **Unity Asset Store** [10], iar imaginile de fundal au fost generate automat de către un o unealtă gratuită [11].

Capitolul 4

Concluzii

4.1 Limitări și probleme întâmpinate

Această secțiune evidențiază atât limitările aplicației, cât și câteva probleme întâmpinate pe parcursul dezvoltării. Concret, voi prezenta capacitățile aplicației, și cât de capabilă este arhitectura construită.

O primă limitare este cauzată de **Skulpt**. Așa cum am menționat anterior, această librărie include anumite pachete de **Python**, care sunt mai des întâlnite, deci în cazul în care se dezvoltă aplicația și se dorește să fie implementate bloculețe mai complicate, sau mai speciale, dezvoltatorul ar trebui să se asigure că poate obține respectiva funcționalitate cu pachetele deja existente în **Skulpt**.

O altă limitare, poate fi reprezentată de performanța **Unity**. Această limitare se datorează faptului că de obicei acest motor de jocuri generează fișiere foarte mari, chiar și pentru proiectele simple. Acest lucru poate duce consumul ridicat de resurse a dispozitivului de pe care se rulează, și pot crește timpii de încărcare. Consider că această limitare este una destul de generală la proiectele **Unity**, dar este o limitare necesară. Fiind un motor de jocuri, este clar faptul că dezvoltatorii **Unity** au oferit o gamă largă de posibilități, și din acest motiv într-o aplicație finală există și multe plug-in-uri care poate nu ajută cu nimic aplicația respectivă. Cu toate acestea, consider ca este o limitare necesară. În schimbul acestui posibil consum mai mare de resurse, **Unity** compensează cu o varietate foarte mare de funcționalități.

Limitarea specifică **Flask** este că, în primul rând, necesită rulare pe server dedicat, adică nu poate rula direct din **Unity**. Un alt motiv pentru care **Flask** este limitat este faptul că aplicația în sine este **single-threaded**, ceea ce înseamnă că nu este gândit pentru a susține simultan un număr mare de utilizatori, deci nu este scalabil fără un server de producție. Totodată, **Flask** nu oferă suport nativ pentru autentificare, baze de date sau validări, spre deosebire de alte freamework-uri precum **Django**.

Una dintre problemele întâmpinate a fost legată de **Skulpt**. Având în vedere ca

acesta este un interpretor de **Python**. el poate primi ca și input orice fel de cod **Pyhton**. Problema apare atunci când utilizatorul încearcă să construiască logică eronată, și trimite către **Skulpt** pentru interpretare. Concret, utilizatorul poate da ca input câteva linii de cod care formează o buclă infinită. Din moment ce am ales să integrez un interpretor, și nu am făcut propriul meu interpretor, acest fapt poate duce la un comportament neașteptat în aplicație, de exemplu se poate bloca. Soluția la această problemă a fost una destul de simplistă, doar că nu una perfectă. Având în vedere ca **Skulpt** nu are mecanisme de a detecta bucle infinite, am fost nevoit să folosesc un **Web Worker**. Acesta îmi permite să rulez interpretarea într-un fir de execuție separat de cel în care rulează aplicația web. Dacă acest fir de execuție nu transmite înapoi un semnal în mai puțin de o secundă, considerăm că interpretarea codului a dat greș din cauza unei bucle infinite. O soluție mai elegantă poate fi de exemplu să se implementeze un nou interpretor foarte minimalist, care să funcționeze strict pentru această aplicație, și să gestioneze corespunzător astfel de incidente neplăcute.

O altă problemă întâmpinată a fost legată de complexitatea arhitecturii. Fiind foarte multe tehnologii implicate, automat au apărut o multitudine de limbaje de programare care trebuiau folosite. Un exemplu exact a fost atunci când aveam o mică eroare la aplicație, și nu știam exact ce componentă nu funcționa. În astfel de momente este necesar să verifici orice componentă, iar acest proces poate dura foarte mult. Încercând să fac debugg-ing (în script-urile de **C#** din **Unity**, cele de **Python** din **Backend** dar și cele de **JavaScript** care încercau să interpreteze codul), încercam să afișez tot felul de variabile pe ecran sau în console pentru a-mi da seama unde este problema. Din neatenție, am confundat limbajele de programare, și într-un script **JavaScript** am scris **print(variabilă)**, (funcție ce bineînțeles ar fi funcționat în **Python**), în loc de **console.log(variabilă)**. Acest incident a avut și o parte umoristică, deoarece pe lângă faptul că aveam acea eroare, acum, dintr-un motiv pe care nu îl puteam înțelege atunci, browser-ul încerca să îmi printeze fizic, o pagină web. Practic, script-ul căuta o imprimantă fizică pentru a printa foi cu aplicația web. Acest incident m-a făcut să mă simt extrem de confuz, și totodată, deși era o oră destul de târzie și eram obosit, m-a făcut să zâmbesc la final, când am înțeles care era de fapt problema.

4.2 Concluzii și extinderea proiectului

Consider că proiectul are potențial. Poate, în prezent nu este un joc atât de atrăgător publicului larg, dar consider că prin muncă poate deveni mult mai interesant.

O sugestie care poate contribui masiv la extinderea proiectului poate fi implementarea de mai multe mecanici. În prezent, structurile de decizie nu sunt atât de utile, adică funcționează, doar ca se pot evita. Ar fi ideal de implementat anumite mecanici noi care să necesite în mod special astfel de blocuri.

O altă sugestie în imediata apropiere de ideea menționată anterior, este poate implementarea mai multor moduri de joc. Spre exemplu se poate implementa un mod cu provocări, care să reprezinte niște nivele de dimensiuni mai mici, dar care necesită o singură rulare de cod. Cu acest mod se pot induce, și se pot sugera subtil descoperirea de pattern-uri de către utilizatori. Eventual, aceste nivele nu au un timp, se pot realiza în cât timp este nevoie, și reprezintă ceva mai mult de exercițiu algoritmic.

Un alt mod de joc, ar putea fi unul în care să fii într-o lume liberă. Adică să te poți plimba pe o hartă care nu face parte dintr-un nivel. Aici poți descoperi personaje noi, inamici, etc. Acest mod de joc ar scăpa utilizatorul de sentimentul provocator de a trebui să completeze un nivel într-un anume interval de timp, și ar putea fi ideal pentru un mediu de învățare. Poate include chiar și un tutorial despre ce face fiecare bloculeț în parte.

O altă idee care se poate implementa ar fi crearea de niveluri personalizate. Această funcționalitate ar permite utilizatorului să creeze propriile nivele, și poate chiar să le distribuie cu prietenii sau cu alte persoane. Această idee poate fi implementată destul de ușor din moment ce tot jocul se bazează pe un grid.

O idee de extindere mai tehnică poate fi integrarea bazelor de date precum **SQL** sau **PostgreSQL**. Această funcționalitate ar permite practic salvarea nivelelor create de utilizator, salvarea progresului în joc, salvarea codului de **Blockly**, etc. Altă alternativă pentru această implementarea ar fi crearea unor script-uri **C#** care să folosească bazele de date deja folosite și gestionate automat de **Unity**. Această soluție nu ar fi deloc practică, deoarece ar necesita foarte multă muncă și nu ar fi garantat să funcționeze corect, așa că integrarea unei baze de date poate fi o idee destul de bună.

O altă idee ar fi implementarea unor conturi de utilizatori, practic aplicația îți va cere să se autentifici, și pe baza rezultatelor poate te va pune într-un clasament cu alți jucători, încurajând astfel competiția într-un joc educativ.

În concluzie, consider că această lucrare are un potențial semnificativ și merită efortul necesar pentru a fi îmbunătățită. De asemenea, cred că își poate atinge cu succes scopurile propuse. Pe parcursul dezvoltării, mi-am îmbunătățit abilitatea de a îmbina diverse tehnologii și, totodată, m-am familiarizat cu multe tehnologii noi.

Bibliografie

[1] Google, URL: https://developers.google.com/blockly.

[2] Scott Graham, URL: https://skulpt.org/.

[3] Zen Fulcrum LLC, URL: https://assetstore.unity.com/packages/tools/gui/embedded-browser-55459?srsltid=AfmBOoptC2Ej1CHnxncl_q1ThSiZF-1a9Hqgnl1TtfEkSGJzsWRCCtCA.

[4] GLOBE NEWSWIRE, *Global Gaming Market Size to Hit USD 424.23 Billion by 2033 | Straits Research*, 2024, URL: https://www.globenewswire.com/news-release/2024/10/08/2960033/0/en/Global-Gaming-Market-Size-to-Hit-USD-424-23-Billion-by-2033-Straits-Research.html?utm_source=chatgpt.com (accesat în 31.5.2025).

[5] python.org, *tkinter — Python interface to Tcl/Tk*, URL: https://docs.python.org/3/library/tkinter.html.

[6] Armin Ronacher, URL: https://flask.palletsprojects.com/en/stable/.

[7] Guido van Rossum, URL: https://www.python.org/.

[8] sfml-dev.org, *Simple and Fast Multimedia Library*, URL: https://www.sfml-dev.org/.

[9] Unity Technologies, URL: https://unity.com/.

[10] Unity, URL: https://assetstore.unity.com/.

[11] Unity, URL: https://assetstore.unity.com/packages/2d/environments/animated-pixel-art-backgrounds-free-255208.

[12] Lea Verou, URL: https://prismjs.com/.

(Acest document se leagă împreună cu lucrarea de licență)

Declarație

Subsemnatul/Subsemnata MITU IUSTIN-AURELIAN, candidat la examenul de licență, sesiunea IUNIE-IULIE 2025, la Facultatea de Matematică şi Informatică, programul de studii Informatică, declar pe propria răspundere că lucrarea de faţă este rezultatul muncii mele, pe baza cercetărilor mele și pe baza informaţiilor obţinute din surse care au fost citate şi indicate, conform normelor etice, în note și în bibliografie. Declar că nu am folosit în mod tacit sau ilegal munca altora și că nici o parte din teză nu încalcă drepturile de proprietate intelectuală ale altcuiva, persoană fizică sau juridică. Declar că lucrarea nu a mai fost prezentată sub această formă vreunei instituții de învățământ superior în vederea obţinerii unui grad sau titlu științific ori didactic.

Data	Semnătura
15.06.2025