Cubul Rubik

Autor: Panțu Mihnea-Andrei 331AB

An: 3

Introducere

Aplicația ce urmează a fi detaliată în paginile următoare este un joc ce presupune amestecarea și rezolvarea celebrului Cub Rubik, creat de Erno Rubik în anul 1974. Prin îmbinarea unei varietăți de concepte multimedia și algoritmică, aplicația urmărește stimularea laturii creative, matematice și orientarea în spațiu a utilizatorului, care va avea ca misiune potrivirea culorilor pe toate cele 6 fețe ale puzzle-ului. Fiecare strat al cubului se mișcă independent, iar utilizatorul poate analiza stadiul în care se află în timpul rezolvării. Acest aspect poate fi realizat fie prin rotirea cubului, fie prin analizarea unei hărți de culori ce reprezintă cubul în plan în timp real, fiind actualizată automat după fiecare mișcare. Mi-am propus ca utilizatorul să poată opta ca amestecarea cubului să fie realizată automat de către aplicație sau îl poate amesteca el însuși. De asemenea, urmăresc implementarea rezolvării automate a cubului de către aplicație în cazul în care jucătorul nu poate duce la bun sfârșit rezolvarea.

Prezentarea suportului tehnic

Pentru realizarea cubului am urmărit întâi schema de culori recunoscută de World Cube Association și anume, albul este opus lui galben, verde opusul culorii albastre, iar portocaliu opusul culorii roșii [2]. Toate aceste culori au fost atașate pe un fundal negru pentru a realiza cubul.

Pentru crearea jocului am utilizat platforma Unity, creată în anul 2005, fiind dezvoltată de Unity Technologies. Pentru implementarea culorilor și a aspectelor vizuale am utilizat paleta de culori RGB pusă la dispoziție de platforma Unity, iar pentru animațiile determinate de mișcările cubului, precum și logica sa de funcționare am redactat fișiere de cod în limbajul C# pe care le-am încărcat ulterior în platformă [1]. Fișierele de cod le-am scris prin intermediul platformei Visual Studio Code 2022.

O parte din conceptele din spatele jocului au fost urmărite din aplicațiile realizate de către membrii ai companiei Megalomobile [3].

Pentru implementarea rezolvării automate a cubului am utilizat algoritmul Kociemba, fiind una din soluțiile optime de rezolvare a puzzle-ului, propusă de către Herbert Kociemba în anul 1992 [4]. Algoritmul a fost transpus în fișiere de cod C# de către cei de la Megalomobile și le-am implementat în cadrul proiectului pentru a realiza rezolvarea automată a cubului [5].

Cubul Rubik are 6 straturi, U(up), D(Down), R(Right), L(Left), F(Front), B(Back). Totodată, aceste notații denotă și mișcările ce pot avea loc pe cub. U înseamnă rotirea stratului superior la 90 de grade în sensul acelor de ceas, U’ în sens invers acelor de ceas, iar U2 presupune rotirea stratului 180 de grade. La fel se aplică și pentru celelalte straturi. Algoritmii Cubului Rubik reprezintă îmbinări de astfel de mișcări. Algoritmul Kociemba este alcătuit din două faze. Dacă în rezolvarea cubului nu utilizăm mișcările R, R', L, L', F, F', B și B' vom genera un subset de mișcări ce aparține grupului G1 = U,D,R2,L2,F2,B2. În acest grup, orientarea colțurilor și a marginilor nu poate fi schimbată. Orientarea marginilor din stratul mijlociu U-D rămâne izolată în acel strat. Faza 1 presupune aducerea cubului în forma G1 de unde poate fi rezolvat folosind doar mutările acestui grup. Faza 2 presupune rezolvarea finală a cubului prin permutarea colțurilor și a marginilor în poziția corectă folosind mișcări ale grupului G1 [6].

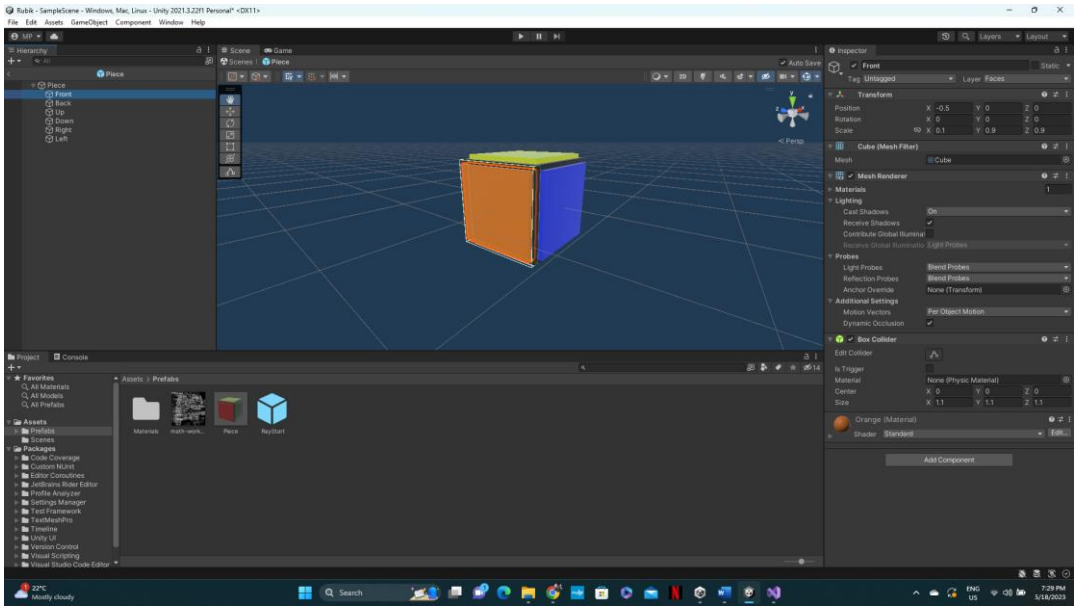
Etapa de implementare

Pentru crearea jocului am utilizat mai multe concepte de multimedia, precum culori din spectrul RGB, imagine de fundal în format jpg, animații și o melodie ce rulează pe fundal în timpul folosirii aplicației. De asemenea am folosit și concepte de algoritmică pentru rezolvarea cubului.

Implementarea am realizat-o în 8 etape prezentate în detaliu mai jos.

Pasul 1:

Am creat în Unity design-ul cubului, am realizat un cub mic pe care apoi l-am copiat de 26 de ori reușind în final să asamblez cubul mare. Am asamblat schema de culori a cubului respectând schema de culori standard utilizată de Asociația Mondială a Cubului Rubik, și anume albul este opus lui galben, verde lui albastru, roșu lui portocaliu și invers. Culorile au fost alese dintr-o paletă de culori RGB.



(1)

Pe parcursul construcției cubului am ales la întâmplare ca fața dinspre jucător să fie cea portocalie, astfel fața din spate va fi roșie, cea de sus galbenă, cea de jos albă, cea din dreapta albastră, iar cea din stânga verde.
Codurile RGB folosite pentru fiecare culoare:

Roșu: (255, 18, 18)

Portocaliu: (238, 126, 10)

Albastru: (18, 52, 229)

Verde: (24, 176, 55)

Alb: (255, 255, 255)

Galben: (223, 233, 23)

După aceea, am ales o imagine cu ecuații matematice pentru a o folosi ca fundal în cadrul jocului.

Pasul 2:

Ulterior cu ajutorul unui script in C# m-am apucat sa scriu codul pentru a putea realiza diverse operații cu cubul. În primul rând, m-am ocupat de rotirea cubului într-un sistem de axe Oxyz. Jucătorul poate roti cubul în jurul axelor apăsând click dreapta și mișcând mouse-ul. Întâi am studiat mișcarea cubului in axa Oy, dacă diferența între coordonatele axei Ox finale si inițiale este negativă, înseamnă că puzzle-ul trebuie rotit la stânga (90 de grade) in jurul Oy, în caz contrar trebuie rotit la dreapta. Inițial rotirea era foarte bruscă, așa că am implementat o tranziție pentru rotirea in jurul axei Oy a cărei viteza poate fi modificata după bunul plac. Am procedat asemănător si pentru mișcarea in jurul celorlalte axe.

Am repoziționat camera pentru a vedea cât mai bine cele 3 fețe care se află către jucător și am setat lumina astfel încât să nu existe umbre lăsate de cub sau pe cub și să se distingă culorile cubului cat mai bine.

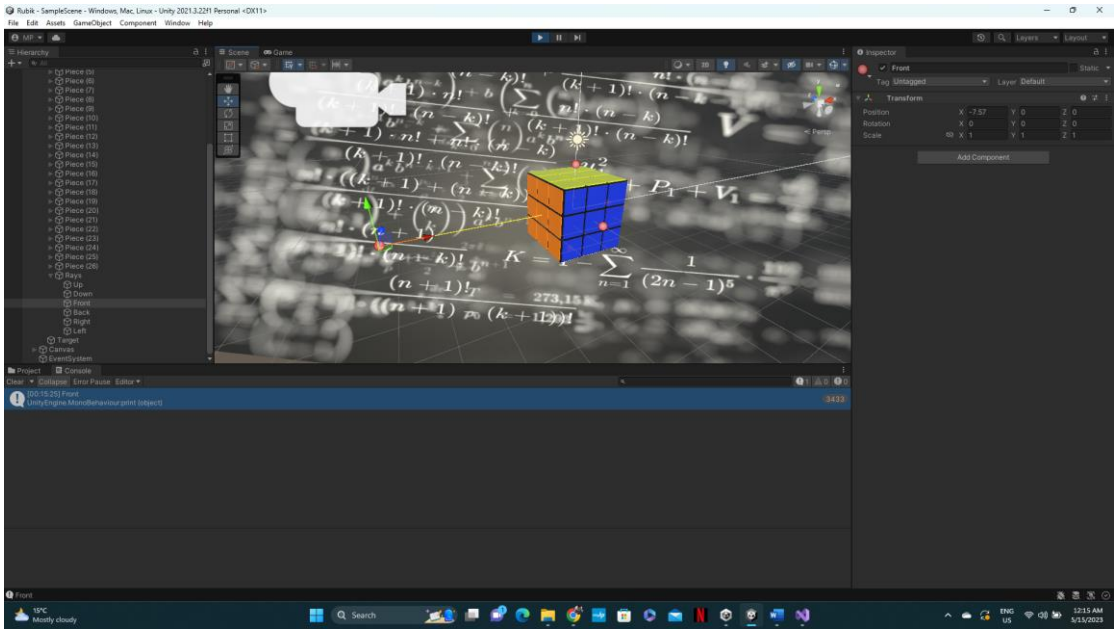
Cât timp butonul de la mouse este ținut apăsat, cubul se poate mișca în jurul axei sale centrale și putem analiza mai multe laturi ale acestuia, mișcând mouse-ul. Daca butonul mouse-ului este eliberat, cubul se va roti în direcția in care am deplasat mouse-ul.

Pasul 3:

Vom crea o hartă plană a cubului care ne va arăta ce culori din cub se regăsesc pe fiecare față. Astfel, când îl amestecăm și încercăm să îl rezolvăm, vom putea analiza și culorile care se află pe partea din spate a cubului uitând-ne pe această hartă. Am implementat un obiect numit Rays care are cate un obiect copil folosit pentru a citi fiecare față a cubului Rubik și pentru a putea ulterior să actualizăm harta (Up, Down, Front, Back, Right, Left). Am crescut dimensiunea fiecărui “sticker” al cubului pentru a fi sigur că razele citesc corect fiecare față a cubului.

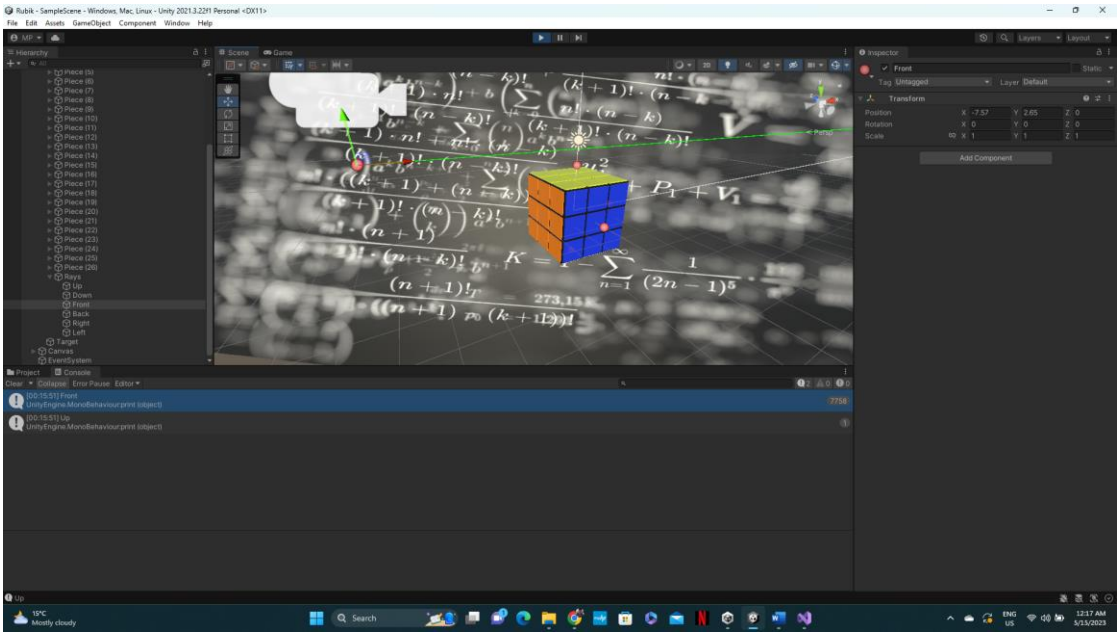
Am atribuit fiecărei fețe o latură (față, spate, dreapta etc.). Am creat un nou script numit Cube State care va înregistra stadiul în care este cubul sub formă de liste.

Am creat o nouă listă care va conține fețele ce au fost atinse de una din razele folosite pentru identificarea culorilor din cadrul harții. Razele pot citi doar fețele pe care le văd. Am implementat o funcție care să semnaleze dacă razele intersectează piese ale cubului, caz în care sunt galbene, iar în caz contrar sunt verzi.



(2)

Raza este galbenă când atinge cubul.

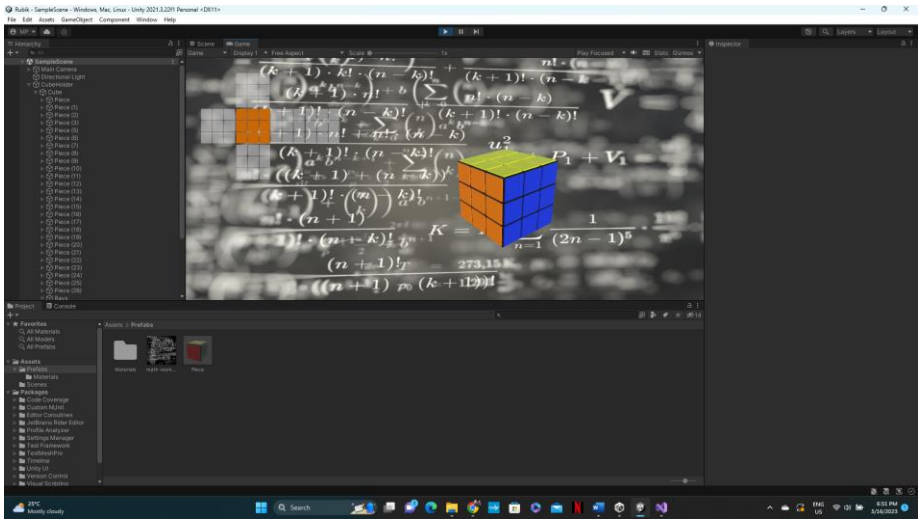


(3)

Raza este verde când nu intersectează cubul.

Am creat un nou script pentru harta cubului, pentru a actualiza culorile fiecărei fețe a hărții în funcție de culorile pe care le intersectează razele.

Am implementat o metoda de actualizare a fețelor harții in funcție de ceea ce intersectează razele descrise mai sus. Pentru a putea citi culorile fețelor, trebuie ca razele să caute culorile exacte în funcție de codurile RGB pe care le-am folosit de la bun început. Funcția utilizata folosește coduri RGBA cu domeniul de valori in [0;1] astfel trebuie împărțite valorile de mai sus la 255. Funcția se numește Color. Canalul Alpha a fost setat la 1. Pentru început am realizat citirea fetei dinspre jucător, si anume cea portocalie. Astfel urmărind pașii de mai sus am reușit sa citesc fata portocalie si sa o trec in harta cubului.



(4)

Pasul 4:

Inițial am folosit o singură rază, acum voi folosi 9 raze dispuse în forma uneia dintre laturile cubului, începând numerotarea de la raza 0 din partea stanga-sus a unei laturi până la raza 8 din partea dreapta-jos a unei laturi. Distribuția razelor pe cub ar arata în felul următor:
|0|1|2|

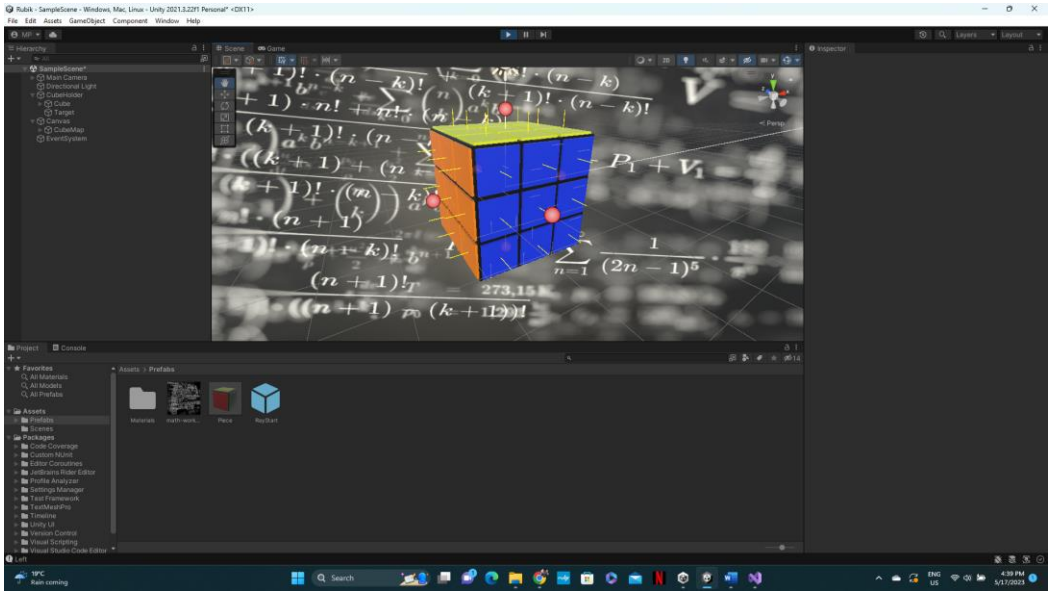
|3|4|5|

|6|7|8|
Acestea sunt practic punctele de intersecție ale razelor cu fețele cubului.

Pentru a determina coordonatele fiecărei celule, pornind din centru (0,0), care este |4|, |0| ar fi de coordonate (-1,1), |1| de coordonate (0,1) și așa mai departe. Am create cele 9 raze folosindu-mă de aceste coordonate.

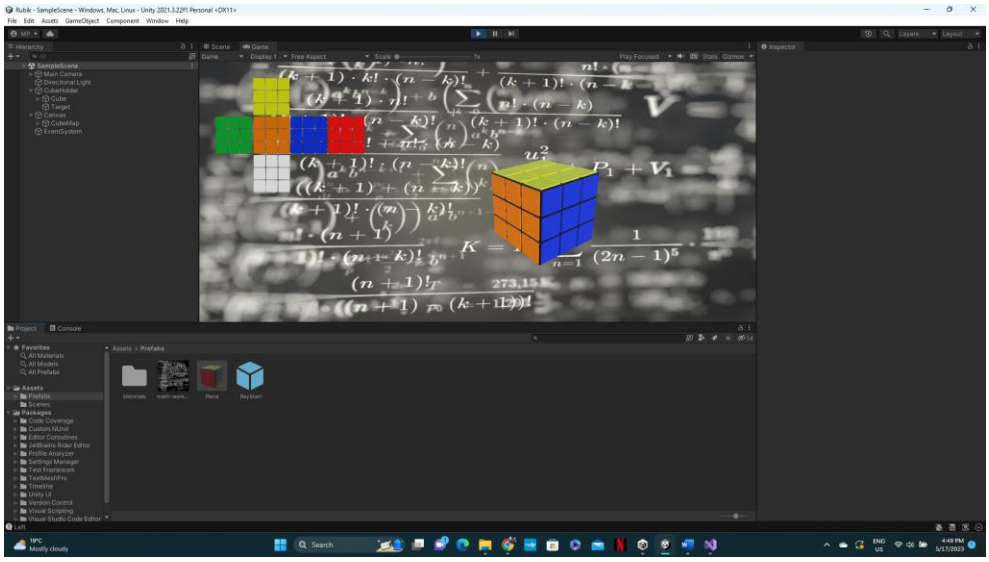
Am creat 6 liste, câte una pentru fiecare fată a cubului și fiecare listă va conține câte 9 raze, pentru fiecare componentă a unei fețe. Ulterior am completat listele cu raze pentru fiecare strat al cubului pentru a putea citi culorile și pentru a le putea reda în hartă. După aceea am implementat o metodă prin care să realizez citirea folosindu-mă de listele de raze de razele și de coordonatele menționate. Metoda implementată citește un singur strat al cubului drept urmare am implementat o nouă metoda pentru a citi toate fețele cubului, folosindu-mă de metoda anterioară.

Astfel , acum avem raze care să citească culorile fiecărei componente a fiecărui strat în parte.



(5)

Și cu ajutorul razelor, citind culorile pieselor am reușit să transpun cubul și culorile pieselor în harta acestuia:



(6)

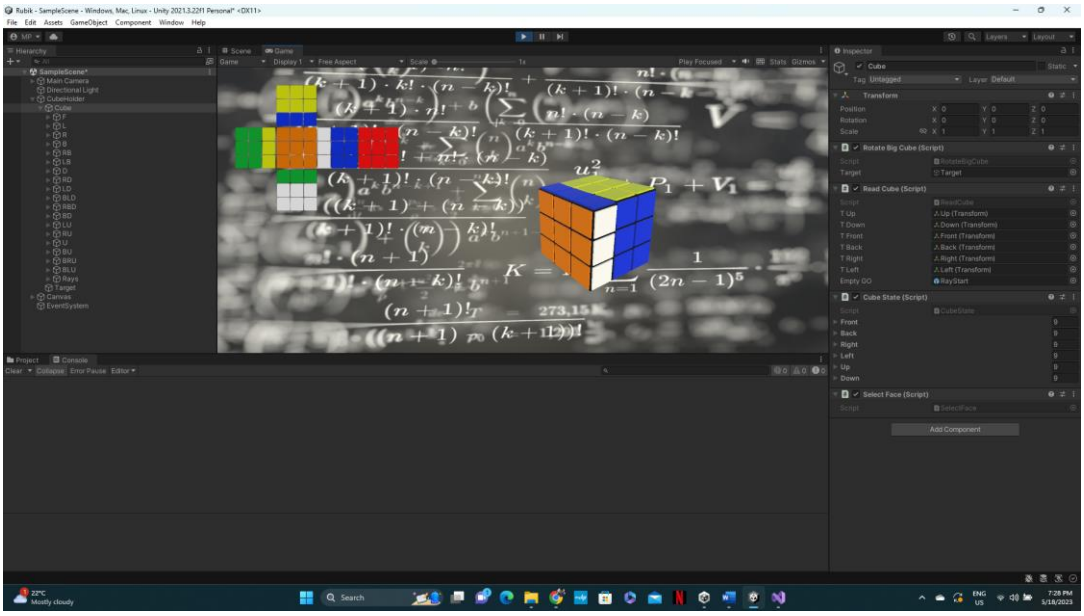
Pasul 5:

Următorul pas este rotirea propriu-zisă a straturilor cubului. Am început prin implementarea unui nou script care să selecteze câte o față și piesele de pe acea față a cubului. Ulterior toate piesele feței respective vor deveni copiii piesei de centru, ca atunci când centrul se rotește, să mute împreună cu el toate piesele din jurul lui.

Straturile unui cub se învârt în jurul a 6 pivoți, fiecare pivot fiind reprezentat de centrul fiecărui strat. Am create un nou script numit PivotRotation pentru a realiza mișcările cubului.

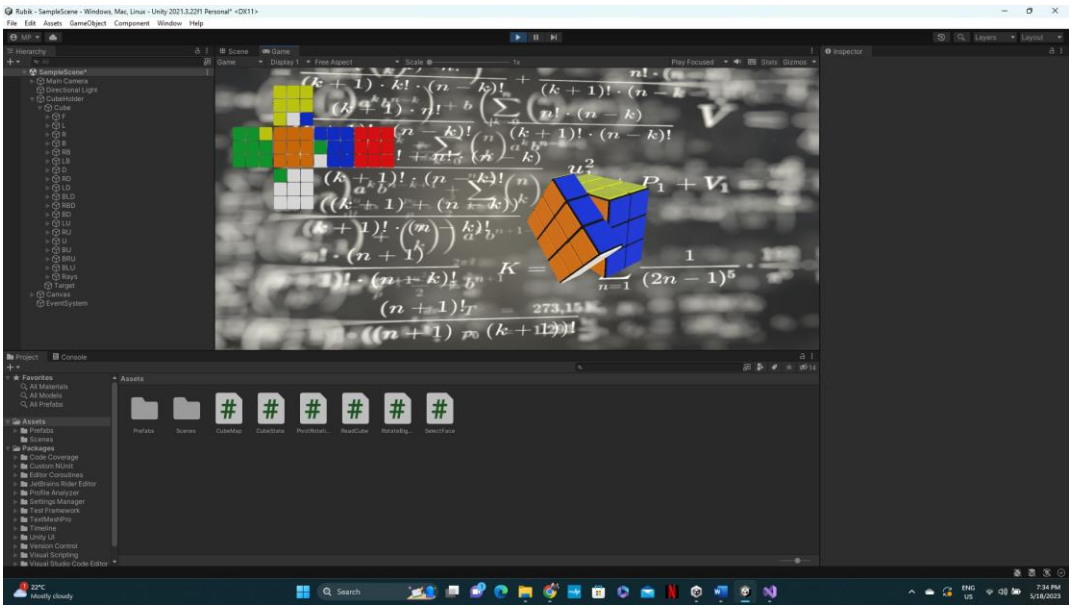
Vom folosi o metodă care este apelată o dată la începutul rotației pentru a seta variabilele necesare rotirii și vom face rotația propriu-zisă în meotda Update.

Am început prin a încercă să rotesc stratul dinspre noi (front layer). Am creat o nouă metodă care analizează stratul pe care vrem să îl rotim și cum trebuie rotit acesta pe baza mișcărilor și coordonatelor mouse-ului. Spre exemplu că să rotim stratul Front, trebuie deplasat centrul acestui strat pe axa OX. Astfel dacă punem mouse-ul pe una din piesele stratului Front și apăsam click stângă, acestea devin copii ai centrului stratului Front și se vor muta toate împreună în funcție de direcția în care jucătorul mișcă mouse-ul.



(7)

Problema acum e că dacă eliberăm mouse-ul la întâmplare până ce stratul efectuează o mișcare completă, stratul va rămâne nealiniat, astfel următorul punct pe care îl urmăresc este ca în momentul în care eliberez mouse-ul în timpul mișcării, cubul să se duca în cea mai apropiată poziție completa, să nu rămână stratul nealiniat precum în imaginea de mai jos



(8)

Astfel am creat o nouă metodă pentru rotirea completă automată. Prima dată folosim metoda pentru a seta variabilele de care am nevoie. M-am folosit de un vector de unghiuri Euler. Unghiurile Euler reprezintă o rotație 3D realizând 3 rotații separate în jurul axelor individuale. Rotațiile sunt efectuate în jurul axelor z,x,y în aceasta ordine. Am luat fiecare componentă, corespunzătoare fiecărei axe și vom aproxima coordonatele lor astfel încât în momentul în care jucătorul eliberează mouse-ul, stratul să efectueze o rotație de 90 sau de -90 de grade (o rotație completă). Ulterior metoda este apelată în Update și se vă aplica de fiecare data când vrem să rotim un strat.

Folosindu-mă de această metodă și de datele ei, implementăm o nouă metodă pentru a alinia stratul cubului și ulterior piesele care mai devreme au fost transformate în copiii centrului vor reveni în ierarhia de obiecte la stadiul inițial. Pentru a pune piesele la loc în ierarhie, am implementat o metodă separată.

Vom reciti cubul pentru a actualiza harta 2D a cubului și vom reseta rotirea automată pentru a fi folosită din nou la următoarea rotire de strat.

Am repetat procedeele descrise mai sus pentru fiecare strat al cubului din cele rămase (cu excepția Front). Fețele Front și Back se rotesc în jurul axei OX, Up și Down în jurul OY, iar right și left în jurul axei OZ. Ca să mișcam un strat în sensul acelor de ceas, ne punem cu mouse-ul deasupra lui, și mișcăm mouse-ul către sensul pozitiv a unei axe imaginare OX. Pentru a roti stratul în sens invers acelor de ceas, mișcăm mouse-ul în sensul negativ al unei axe imaginare OX.

Acum, fiindcă eu am creat cubul prin copierea cubului mic de la pasul 1, se observă că pe interior cubul mare este colorat. Astfel vom elimina culorile din interior care sunt în plus, pentru a avea cubul negru pe interior și colorat doar pe exterior. Pentru a face acest lucru, am selectat fiecare piesa și am șters culorile care nu au nicio utilitate.

Pasul 6:

Am dorit să implementez o modalitate prin care cubul să se amestece singur, apăsând butonul scramble. Pentru acest lucru am folosit un nou script numit Automate. Cubul va efectua mișcări dintr-o listă de mutări. Când facem o mutare trebuie să rotim tot stratul astfel am implementat o metodă care să se ocupe de acest lucru. Metoda are ca parametrii stratul care urmează să fie rotit și unghiul de rotire. Am implementat și o metoda care se ocupă de setarea acestui unghi în funcție de mișcările pe care le facem. Există 6 tipuri de mutări U,D,F,B,R,L și pentru fiecare din ele cubul se mișcă cate 90 de grade în sensul acelor de ceas, pe diferite axe, în funcție de strat.

Am creat o metodă și pentru a seta variabilele necesare mișcării automate. Pentru a roti fiecare strat ne vom folosi de scriptul anterior Pivot Rotation, cu ajutorul căruia roteam straturile manual. Inițial m-am ocupat de stratul de sus (U). Dacă în lista de mutări programul întâlnește U, va muta stratul de sus 90 de grade în sensul acelor de ceas. Ulterior m-am ocupat de celelalte straturi. Spre exemplu daca întâlnește L, va muta stratul din stânga 90 de grade în sensul acelor de ceas și așa mai departe. De asemenea, daca una din mutări este urmata de apostrof, U', L' etc, înseamnă că stratul corespunzător trebuie mutat 90 de grade în sens invers acelor de ceas. Dacă o mutare este urmată de 2, U2,L2,F2, etc, înseamnă că stratul corespunzător trebuie rotit 180 de grade în jurul axei centrale a stratului.

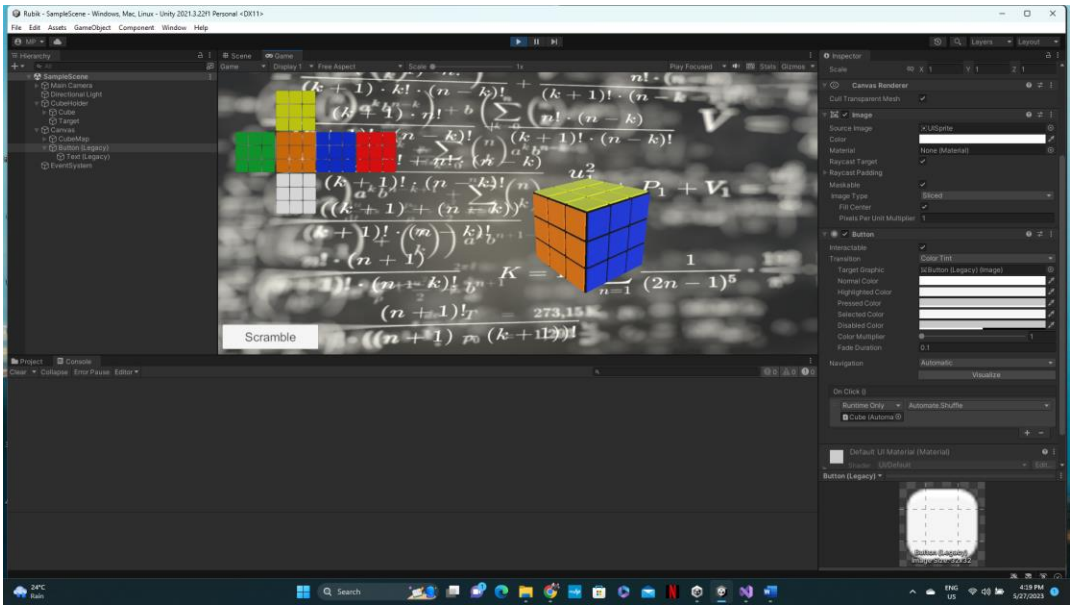
Am creat o listă în care am scris toate mutările posibile care pot fi aplicate cubului,

După aceea, vom folosi o listă de mișcări la întâmplare pentru a genera amestecări diferite ale cubului și pentru a anima aceste mișcări.

Am implementat aceste aspecte creând o nouă metodă numită Shuffle. Metoda va crea liste de diferite amestecări pentru cub. Dimensiunea acestei liste de amestecare va fi mereu aleatoare, fiind ales mereu la întâmplare un număr între 15 și 25. Am creat o buclă for de la 0 la numărul de mutări ales la întâmplare, în care vom selecta aleator câte o mutare, pe rând, din lista cu toate mutările posibile ce pot fi aplicate cubului, și vom stoca aceasta mutare în lista de amestecare. Repetându-se acest procedeu în bucla, ne vom crea o listă de amestecare, ce va fi mereu diferita.

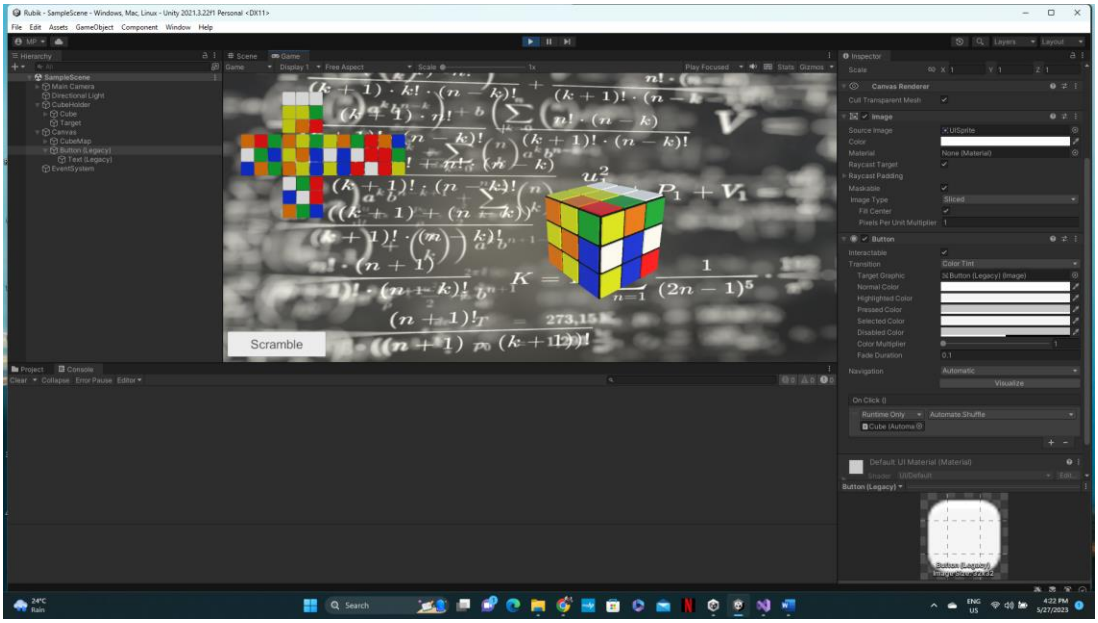
La final, am creat un buton de amestecare, iar mereu când utilizatorul apasă acest buton, cubul se va amesteca.

Înainte de apăsarea butonului scramble:



(9)

După apăsarea butonului:



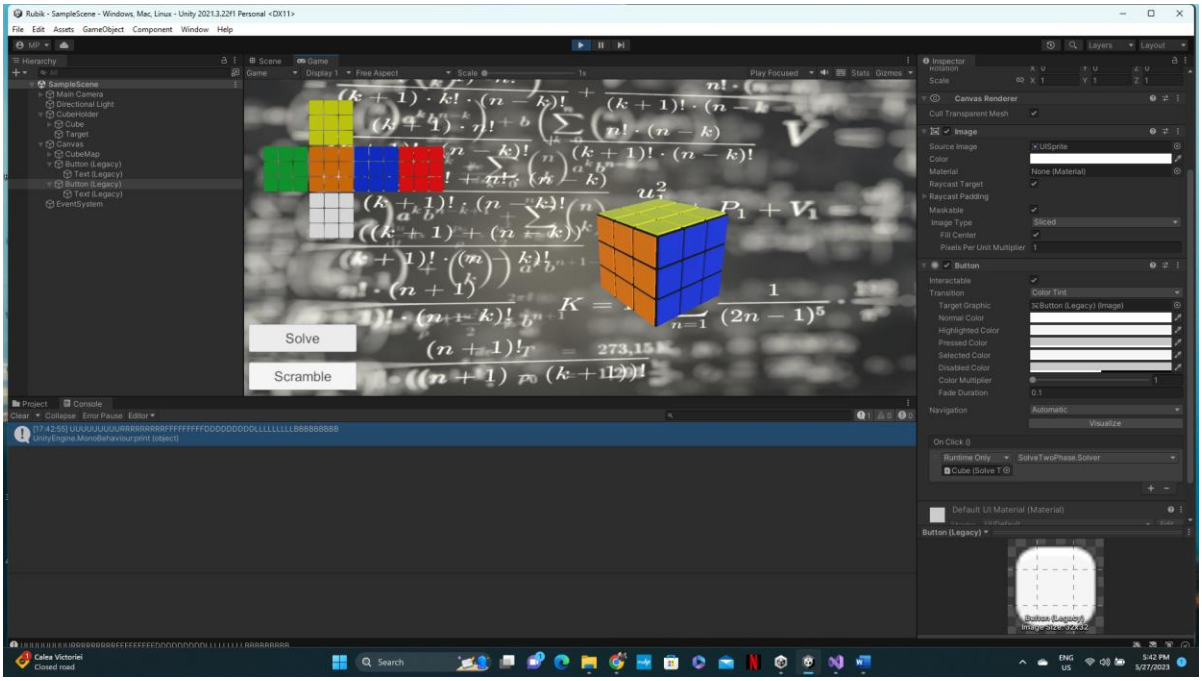
(10)

Pasul 7:

Doresc să implementez o metodă automată de rezolvare a cubului prin apăsarea unui buton numit solve. Metoda de rezolvare va folosi algoritmul lui Kociemba. Am preluat algoritmul de rezolvare și l-am atașat proiectului meu folosind un nou script.

În noul script numit SolveTwoPhase am implementat o metodă pentru a rezolva cubul, numită Solver. Întâi trebuie citit cubul. După aceea pașii pe care i-am urmat au fost: să transpunem starea de amestecare a cubului într-un șir de caractere, să obținem mișcările necesare pentru rezolvare din algoritmul Kociemba, să convertim mișcările din sir de caractere în lista. Să utilizam lista pentru rezolvare. Am avut grija că la construirea șirului de caractere să urmăresc ordinea pieselor așa cum indică algoritmul Kociemba, luând fiecare strat în parte, iar pentru fiecare strat începem cu piesa din stânga-sus și ne oprim la cea din dreapta-jos. Ordinea straturilor este: U R F D L B.

Am verificat ulterior că am introdus piesele în ordinea corecta a straturilor conform algoritmului Kociemba:



(11)

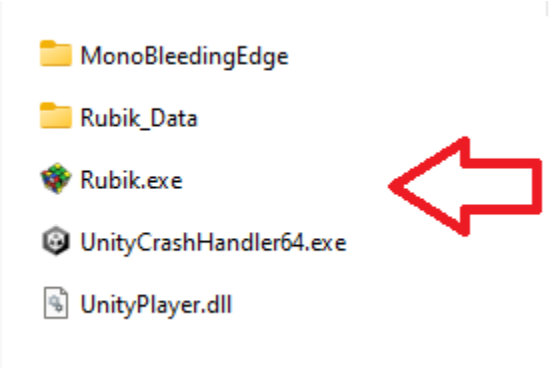
Algoritmul se va executa la apăsarea butonului Solve, cubul fiind rezolvat indiferent de stadiul în care este amestecat.

Pasul 8:

Am adăugat o melodie clasică pe fundal și un buton quit ce poate fi apăsat oricând de utilizator dacă acesta dorește să iasă din aplicație. Am ales o imagine cu un cub Rubik în format png pentru pictograma executabilului.

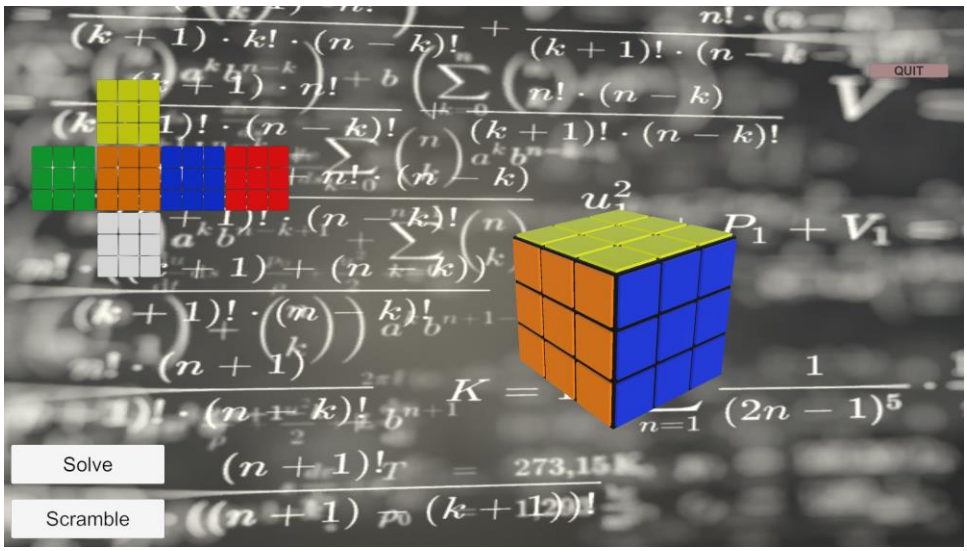
Interacțiune cu utilizatorul

Ca utilizatorul să pornească aplicația, el trebuie să apese dublu click pe pictograma executabilului din folder-ul în care a fost instalată aplicația.



(12)

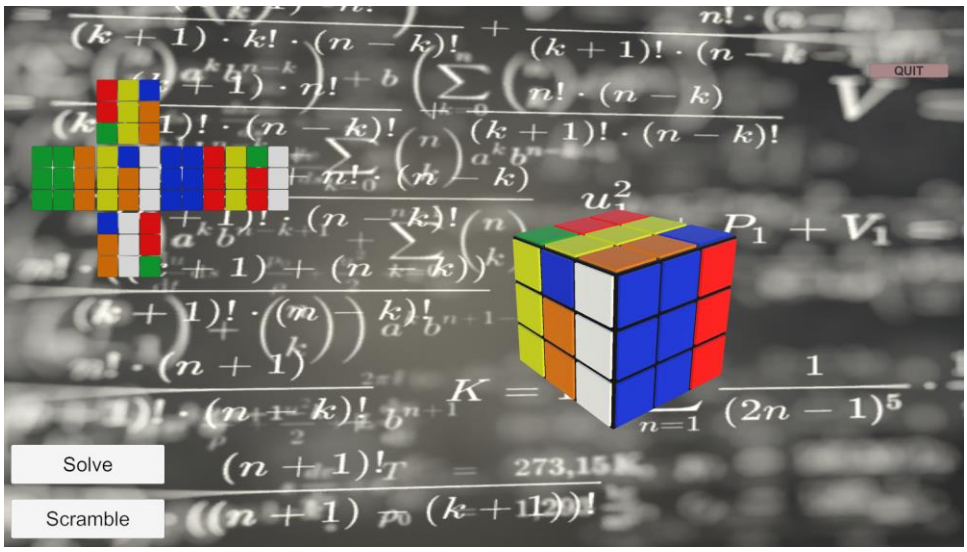
După aceea, aplicația pornește, iar utilizatorul poate începe să utilizeze cubul.



(13)

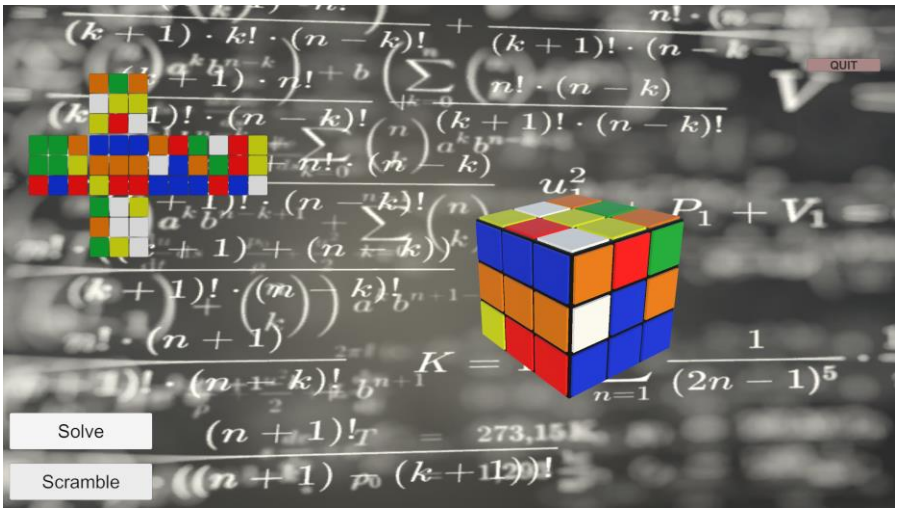
Pentru a roti straturile cubului, utilizatorul trebuie să își pună cursorul de la mouse deasupra stratului pe care vrea să îl miște. Ținând click dreapta pe acest strat, dacă jucătorul mișcă mouse-ul la dreapta, stratul se va roti în sensul acelor de ceas, dacă mută mouse-ul la stânga, stratul se va roti în sens invers acelor de ceas. Pentru a roti întreg cubul dintr-o parte în alta, trebuie ținut click stânga pe mouse și rotit în funcție de cum dorește utilizatorul să deplaseze cubul.

Spre exemplu dacă dorește să facă la întâmplare 3 mutări, U, R, L, cubul va arăta astfel:



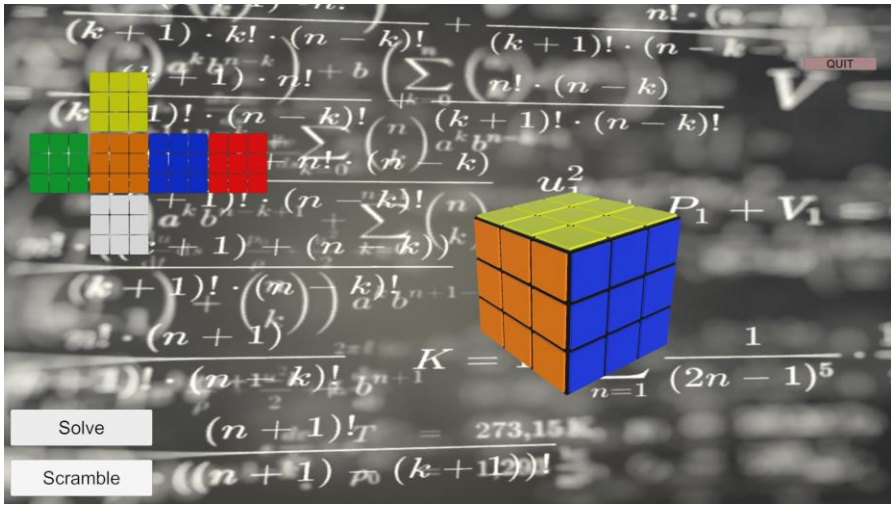
(14)

Dacă utilizatorul dorește ca aplicația să îi amestece cubul la întâmplare, el trebuie să apese butonul scramble.



(15)

Așa arată cubul cu o amestecare aleatoare realizată de aplicație. Dacă utilizatorul vrea ca puzzle-ul să fie rezolvat automat, trebuie să apese butonul Solve.



(16)

Pentru a părăsi aplicația, jucătorul poate apăsa butonul quit din partea dreaptă-sus a ecranului.

Concluzii

În cele din urmă, urmând pașii prezentați mai sus, am reușit prin utilizarea unor concepte multimedia precum culori RGB, imagini, sunete, animații și concepte de algoritmică să realizez o aplicație care să simuleze interacțiunea cu un cub Rubik într-un mediu virtual. Toate obiectivele propuse au fost atinse, astfel jucătorul poate folosi cubul rotind oricare din cele 6 straturi după bunul plac, poate să îl amestece folosind butonul scramble, fiindu-i generată o amestecare aleatoare și poate rezolva cubul fie singur, fie apăsând butonul solve. Acest ultim buton realizează rezolvarea automată a cubului oricare ar fi stadiul lui, în cât mai puține mutări, algoritmul Kociemba fiind una din soluțiile optime de rezolvare a cubului. Pe tot parcursul interacțiunii dintre utilizator și aplicație, acesta poate urmări stadiul în care se află piesele pe cub, analizându-le în harta 2D de culori a cubului, ce reprezintă puzzle-ul în plan.

Referințe

[1] https://en.wikipedia.org/wiki/Unity_(game_engine), 2005

[2] https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/, Thomas, 2009

[3] https://www.megalomobile.com/

[4] https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube

[5] https://github.com/Megalomatt/Kociemba/tree/Unity

[6] http://kociemba.org/cube.htm Hembert Kociemba, 1992