

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK**



**SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM**

Rubik-kocka kirakó mobilalkalmazás

DIPLOMADOLGOZAT

Témavezető:
Dr. Iclánzan Dávid,
Egyetemi docens

Végzős hallgató:
Kolumbán Attila

2023

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ**



**UNIVERSITATEA
SAPIENTIA**

Aplicație mobilă pentru rezolvarea cubului Rubik

LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Iclănanzán Dávid,
Conferențiar universitar

Absolvent:
Kolumbán Attila

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA**
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

Rubik's cube solver for phones

BACHELOR THESIS

Scientific advisor:
Dr. Iclănanzán Dávid,
Associate professor

Student:
Kolumbán Attila

2023

Declarație

Subsemnatul/a KOLUMBÁN ATTILA, absolvent(ă) al/a specializării INFORMATICA, promoția 2018, cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, TÂRGU-MUREŞ
Data: 13.06.2023

Absolvent

Semnătura. Kolumbán

LUCRARE DE DIPLOMĂ

Coordonator științific:
dr. Iclanzan David

Candidat: **Kolumbán Attila**
Anul absolvirii: 2023

a) Tema lucrării de licență: Dezvoltarea unei aplicații multiplatformă de citire (scanare) și rezolvare a cubului Rubik

b) Problemele principale tratate:

- Studiu bibliografic asupra algoritmilor de rezolvare a cubului Rubik.
- Studierea metodelor existente pentru scanarea și recunoașterea cubului Rubik utilizând imagini statice sau camere (streamuri) video.
- Analizarea tehniciilor și metodelor de recunoaștere a culorilor și pozițiilor pieselor cubului Rubik în imagini.
- Proiectarea, implementarea și testarea algoritmului de scanare a cubului Rubik folosind camera dispozitivelor mobile și/sau webcam-ului pe PC.
- Dezvoltarea unui algoritm eficient și robust de recunoaștere a culorilor și identificare a pozițiilor pieselor cubului.
- Examinarea platformelor și tehnologiilor multiplatformă pentru dezvoltarea aplicației, cum ar fi Unity, React Native sau Flutter. Cercetarea și compararea instrumentelor și bibliotecilor disponibile pentru dezvoltarea aplicațiilor multiplatformă.
- Proiectarea unei interfețe utilizator prietenoase pentru capturarea imaginilor și afișarea rezultatelor scanării în 3D.
- Implementarea sau adaptarea din librării a algoritmilor de rezolvare a cubului Rubik, care să genereze o soluție optimă într-un timp rezonabil.
- Compararea teoretică și empirică a algoritmilor de rezolvare pentru cubul Rubik.
- Examinarea aspectelor de utilizabilitate și interfață utilizator pentru a crea o experiență prietenoasă și intuitivă.
- Documentarea adekvată a stadiilor de proiectare, implementare și testare a aplicației.

c) Desene obligatorii:

- Schema bloc a sistemului
- Diagrame de proiectare, implementare și testare pentru aplicația software realizată.

d) Softuri obligatorii:

- Aplicație multiplatformă de citire (scanare) și rezolvare a cubului Rubik.

e) Bibliografia recomandată:

Demaine, E. D., Demaine, M. L., Eisenstat, S., Lubiw, A., & Winslow, A. (2011). Algorithms for solving Rubik's cubes. In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5–9, 2011. Proceedings* 19(pp. 689–700). Springer Berlin Heidelberg.

El-Sourani, N., Hauke, S., & Borschbach, M. (2010). An evolutionary approach for solving the Rubik's cube incorporating exact methods. In *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7–9, 2010, Proceedings, Part I* (pp. 80–89). Springer Berlin Heidelberg.

McAleer, S., Agostinelli, F., Shmakov, A. K., & Baldi, P. (2019, January). Solving the rubik's cube with approximate policy iteration. In *International Conference on Learning Representations*.

Agostinelli, F., McAleer, S., Shmakov, A., & Baldi, P. (2019). Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8), 356-363.

Hocking, J. (2022). *Unity in action: multiplatform game development* in C. Simon and Schuster.

Blackman, S. (2011). *Beginning 3D Game Development with Unity: All-in-one, multi-platform game development*. Apress.

f) Termene obligatorii de consultații: săptămânal, preponderent online
g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 413
Primit tema la data de: 25.04.2022
Termen de predare: 06.07.2023

Semnătura Director Departament

Semnătura responsabilului
programului de studiu

Semnătura coordonatorului

Semnătura candidatului

Kivonat

A Rubik-kocka a világ egyik legikonikusabb és legérdekesebb puzzle játéka. 1974 volt az év, amikor Rubik Ernő Magyar építész, tervező, illetve játékfejlesztő elkészítette az első Rubik-kockát. Feltalálása óta világszerte népszerűvé vállt, több millió embert szórakoztat naponta. A Rubik-kocka megoldása odafigyelést, türelmet és stratégikus gondolkodást igényel, de köszönhetően a technológia fejlődésének ez napjainkban sokkal könnyebben elérhető mint régen.

Okos eszközeink fejlődésének és automatizált robotjainknak köszönhetően a Rubik-kocka kirákása új szinteket ért el. Rubik-kocka megoldó alkalmazások manapság elérhetőek úgy okostelefonokon, mint asztali számítógépeken, nagyban megkönnyítve a játékosok dolgát az algoritmusok és kirakási módszerek elsajátításában. Egy ilyen alkalmazás általában egy 3D modellt mutat be a Rubik-kockáról a felhasználó képernyőjén, amelyet könnyedén érintéssel lehet mozgatni. Az alkalmazások elemzik a Rubik-kocka adott állapotát és megoldást adnak a felhasználónak lépésenként. A felhasználó lemoshatja ezeket a lépéseket a saját fizikai kockáján és fokozatosan fejlesztheti ilyen módon saját képességeit is. Álltalában egy hasonló alkalmazásnak előny, ha nem csak egy, a kirakáshoz vezető lépést mutat a felhasználónak, hanem lehetőséget ad specifikus algoritmusok betanulására is, amely nagyban elősegíti a felhasználó számára a Kocka működésének titkainak elsajátítását.

Az álltalam továbbgondolt szoftver képes egy okos eszköz kameráját használva beolvasni egy fizikai kocka állását, ezzel nagyban meggyorsítva a fizikai világból a virtuálisba való adatátvitelt. Beolvasás után a felhasználónak lehetősége van egy optimális megoldást lekérni, amelynek lépésein egyenként is végighaladhat, vagy bármikor ha úgy érzi, megpróbálhatja ő maga is befejezni a megoldást a virtuális térben. Ezen kívül fontosnak találtam, hogy lehetőséget adjunk a felhasználó számára egy egy adott megoldó algoritmus elsajátítására is. Erre a célra az applikációban létezik egy külön menüpont, ahol kiválasztható egy megoldó algoritmus és megtekinthetők ennek lépései a kockán.

Az alkalmazást Unityben valósítottam meg, C# nyelv segítségével, a Kocka megoldásához pedig Kociemba két fázisos algoritmusát használtam. Ez az algoritmus köz kedvelt gyorsasága és alacsony lépésszáma miatt.

Rezumat

Cubului Rubik este unul dintre cele mai iconice și interesante jocuri de puzzle din lume. Primul Cub Rubik a fost creat în anul 1974 de către Rubik Ernő, cine a fost un arhitect, sculptor și designer de jocuri. De la înființare, a devenit popular în întreaga lume și aduce distracție milioanelor de oameni în fiecare zi. Rezolvarea Cubului Rubik necesită atenție, răbdare și gândire strategică din partea utilizatorului, dar datorită îmbunătățirilor tehnologiei moderne, modul de gândire necesar pentru a rezolva cubului este mult mai ușor de învățat.

Datorită îmbunătățirilor dispozitivelor noastre inteligente și a roboților automatizați, rezolvarea Cubului Rubik a atins noi niveluri. În prezent, există aplicații pentru rezolvarea Cubului Rubik disponibile pe computerele personale și pe telefoanele inteligente, ceea ce face învățarea noilor algoritmi de rezolvare mult mai ușoară. De obicei, o aplicație de acest gen afișează o reprezentare 3D a Cubului Rubik pe ecranul dispozitivului, care poate fi rotită de către utilizator folosind degetele sau un mouse. Aplicațiile calculează apoi o soluție optimă și o afișează utilizatorului. Utilizatorul poate copia acești pași pe propriul său cub, astfel își poate dezvolta abilitățile proprii. De obicei, este un avantaj dacă aplicația nu numai poate să arăte o soluție optimă, ci poate afișa și algoritmi de rezolvare specifici, ajutând astfel utilizatorul să înțeleagă mai bine cubul.

Software-ul meu poate citi un cub real, folosind camera unui dispozitiv intelligent, accelerând astfel procesul de introducere a stării cubului în aplicație. După ce scanarea cubului este terminată, utilizatorul poate solicita o soluție optimă, prin care poate trece pas cu pas, sau, dacă dorește, poate încerca să rezolve cubul singur. Pe lângă aceasta, am dorit să implementez o funcție prin care utilizatorul poate învăța algoritmi de rezolvare specifici. Din acest motiv, există un meniu separat în care utilizatorul poate alege un algoritm de rezolvare și poate vedea pașii pe cub.

Aplicația a fost realizată în Unity, folosind C#, iar pentru găsirea unei soluții optime am folosit algoritmul de rezolvare în două etape al lui Kociemba. Acest algoritm este apreciat pentru viteza și numărul redus de pași necesari pentru soluție.

Abstract

The Rubik's Cube is one of the worlds most iconic and most interesting puzzle game. The first Rubik's Cube was made in the year 1974 by Rubik Ernő, who was a Hungarian architect, sculptor and game designer. Since its inception, it became popular worldwide, it entertains millions of people every day. Solving the Rubik's Cube demands attention, patients and strategic thinking of its user, but thanks to improvements of modern technology, the way of thinking needed to solve the cube is a lot easier to learn.

Thanks to the improvements of our smart devices and automated robots, solving the Rubik's Cube reached new levels. These days Rubik's Cube solver application are available on personal computers, and also on smart phones, making the learning of new solving algorithms a lot easier. Usually an application like this show a 3D representation of a Rubik's Cube on the screen of the device, which can be rotated by the user using fingers or a mouse. The applications then calculate an optimal solution, and shows it to the user. The user can copy these steps on his own cube, this way he can develop his own skills. Usually its a plus, if the application is not only able to show an optimal solution, but also can show given solving algorithms, this way helping the user understand the cube more.

My software is able to read in a real cube, using the camera of a smart device, this way speeding up the process of inputting the state of the cube into the application. After the scanning of the cube is done, the user can ask for an optimal solution, of which he can go through step by step, or if he feels like it, he can try to solve the cube himself. Aside from this, i wanted to implement a feature, where the user can learn given solving algorithms. For this reason exists a separate menu, where the user can choose a solving algorithm, and can see its steps on the cube.

The application was made in Unity, using C#, and for finding an optimal solution i used Kociemba's two step solving algorithm. This algorithm is well liked for its speed and low solution step count.

Tartalomjegyzék

1. Bevezető	1
1.1. Általános leírás	1
1.2. Hasonló alkalmazások	2
1.2.1. Rubik's Cube Solver	2
1.2.2. AZ Rubik's cube solver	3
1.3. Következtetések	4
2. Felhasznált technológiák és programok bemutatása	5
2.1. Unity https://unity.com/	5
2.2. C#	5
2.3. Kociemba algoritmusa	6
2.4. Visual Studio https://visualstudio.microsoft.com/	7
2.5. GitHub Desktop https://desktop.github.com/	7
3. Funkcionális és nem funkcionális követelmények	8
3.1. Funkcionális követelmények	8
3.2. Nem funkcionális követelmények	10
4. Alkalmazás működésének leírása	11
4.1. Létező program, mint alap	11
4.2. Rubik-kocka megjelenítése 3D térben	12
4.3. Rubik-kocka 2D reprezentációjának megjelenítése	14
4.4. Rubik-kocka beolvasása az eszköz kamerájának segítségével	14
5. Színtelismerő algoritmus	16
5.1. Sikertelen színtelismerési próbálkozások	16
5.1.1. Távolság 2 RGB szín között	16
5.2. RGB vs HSV	16
5.3. Általam készített színtelismerő algoritmus	17
6. Az alkalmazás felhasználói szempontból	20
6.1. A kezdőoldal	20
6.1.1. A kocka forgatása	20
6.1.2. A kocka összekeverése és megoldása	21
6.1.3. Betárazott lépések listája	21
6.1.4. Automatikus forgatás ki/be kapcsolása	21
6.1.5. Algoritmusok nézet	21

6.1.6. Kocka kameráról beolvasott állásra, vagy kezdő állásra állítása	21
6.1.7. Kocka beolvasása kamerával	22
7. Mérések	23
7.1. Megoldás megtalálásának időtartama	23
7.2. Fejlesztéshez felhasznált rendszer specifikációi	23
8. Továbbfejlesztési lehetőségek	25
Összefoglaló	26
Köszönetnyilvánítás	27
Ábrák jegyzéke	28
Táblázatok jegyzéke	29
Irodalomjegyzék	30
Függelék	31
F.1. A Unity játékmotor felülete	31

1. fejezet

Bevezető

1.1. Általános leírás

A Rubik-kocka a világ egyik legikonikusabb és legérdekesebb puzzle játéka. 1974 volt az év, amikor Rubik Ernő Magyar építész, tervező, illetve játékfejlesztő elkészítette az első Rubik-kockát. Feltalálása óta világszerte népszerűvé vállt, több millió embert szórakoztat naponta. A Rubik-kocka megoldása odafigyelést, türelmet és stratégikus gondolkodást igényel, de köszönhetően a technológia fejlődésének ez napjainkban sokkal könnyebben elérhető mint régen.

Okos eszközeink fejlődésének és automatizált robotjainknak köszönhetően a Rubik-kocka kirakása új szinteket ért el. Rubik-kocka megoldó alkalmazások manapság elérhetőek úgy okostelefonokon, mint asztali számítógépeken, nagyban megkönnyítve a játékosok dolgát az algoritmusok és kirakási módszerek elsajátításában. Egy ilyen alkalmazás általában egy 3D modellt mutat be a Rubik-kockáról a felhasználó képernyőjén, amelyet könnyedén érintéssel lehet mozgatni. Az alkalmazások elemzik a Rubik-kocka adott állapotát és megoldást adnak a felhasználónak lépésenként. A felhasználó lemásolhatja ezeket a lépéseket a saját fizikai kockáján és fokozatosan fejlesztheti ilyen módon saját képességeit is. Általában egy hasonló alkalmazásnak előny, ha nem csak egy, a kirakáshoz vezető lépést mutat a felhasználónak, hanem lehetőséget ad specifikus algoritmusok betanulására is, amely nagyban elősegíti a felhasználó számára a Kocka működésének titkainak elsajátítását.

Az általam továbbgondolt szoftver képes egy okos eszköz kameráját használva beolvasni egy fizikai kocka állását, ezzel nagyban meggyorsítva a fizikai világból a virtuálisba való adatátvitelt. Beolvasás után a felhasználónak lehetősége van egy optimális megoldást lekérni, amelynek lépésein egyenként is végighaladhat, vagy bármikor ha úgy érzi, megpróbálhatja ő maga is befejezni a megoldást a virtuális térben. Ezen kívül fontosnak találtam, hogy lehetőséget adjunk a felhasználó számára egy egy adott megoldó algoritmus elsajátítására is. Erre a célra az applikációban létezik egy külön menüpont, ahol kiválasztható egy megoldó algoritmus és megtekinthetők ennek lépései a kockán.

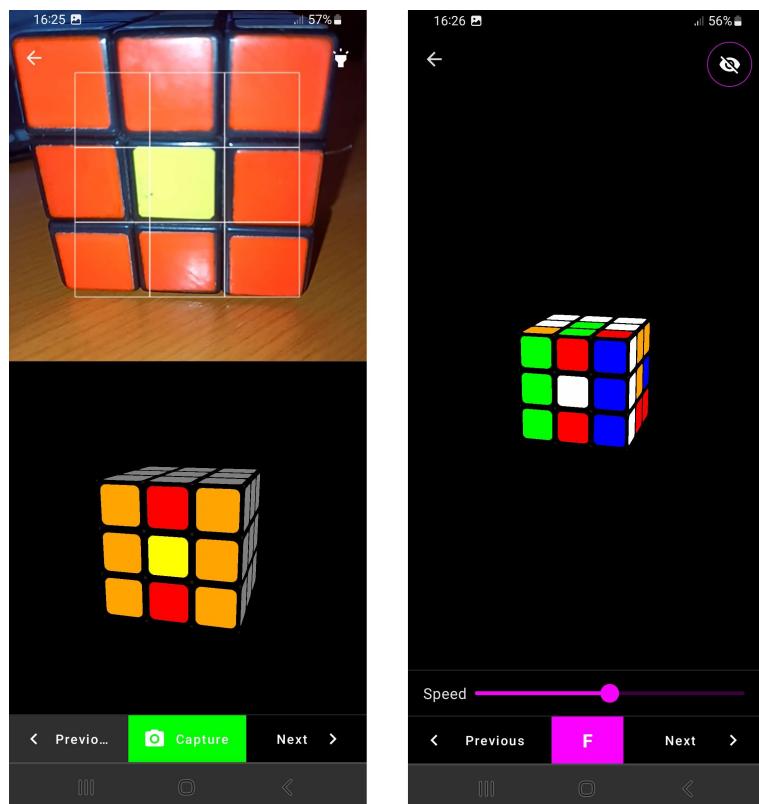
Az alkalmazást Unityben valósítottam meg, C# nyelv segítségével, a Kocka megoldásához pedig Kociemba két fázisos algoritmusát használtam. Ez az algoritmus köz kedvelt gyorsasága és alacsony lépésszáma miatt.

1.2. Hasonló alkalmazások

Nagyon sok Rubik-kocka megoldó alkalmazás létezik, ezek közül nagyon sok tartalmaz az én alkalmazásomban is megtalálható funkciókat, de általában nem az összeset. Számtalan példát találtam, ahol az alkalmazás lehetőséget ad a felhasználónak a kocka beolvasására, megoldására, de nem tudjuk kilistázni a lépéseket, vagy ennek a fordítottja, ahol nagyon szép a felhasználói felület, minden lépést kilistázhatunk, algoritmusokat tanulhatunk, de nem tudjuk kamera segítségével beolvasni a kockát.

1.2.1. Rubik's Cube Solver

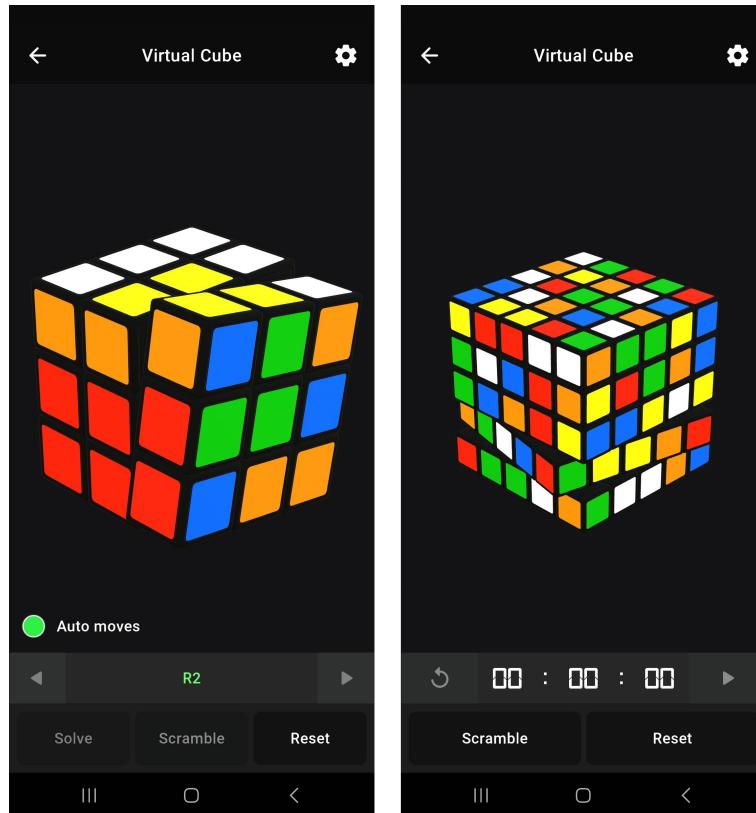
Play áruházban megtalálható alkalmazás, 4.7 csillagú értékeléssel és több mint egy millió letöltéssel. Bár az alkalmazás felhasználói felülete letisztult, de az ingyenes verzió csak a kocka beolvasására ad lehetőséget kamerával, vagy manuálisan és a megoldás lépéseként megtekintését. Itt egy általam észrevett tervezési hiba, hogy a kamerával való beolvasás szörnyen pontatlan, részben azért, mert a telefonon található fényt beolvasáskor bekapcsolja az alkalmazás, ez rontja az oldalak felismerhetőségét. Ezen kívül egy negatívum a túlzott mennyiséggű hirdetés megjelenítés.



1.1. ábra. Rubik's Cube Solver kinézete.

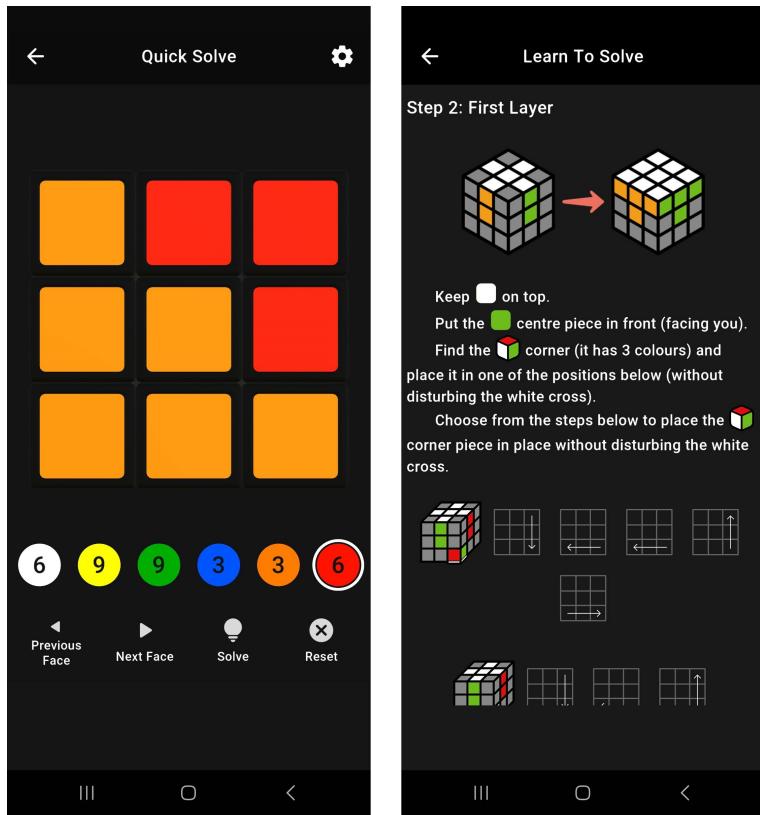
1.2.2. AZ Rubik's cube solver

Ez az alkalmazás is elérhető a Play áruházból, 4.2 csillagú értékeléssel, és több mint egy millió letöltéssel. Általam kipróbált alkalmazások közül ez volt az egyik kedvencem. Funkciói közül a legszimpatikusabb, hogy lehet több méretű Rubik-kockát is választani, ahogy ezt az alábbi ábrán be is mutatom.



1.2. ábra. AZ Rubik's cube solver 3x3 illetve 5x5 kocka módban.

Ezen kívül az alkalmazás tartalmaz kirakáshoz használható stopperórát, egy teljes szekciót, ahol megoldó algoritmusokat tanulhatunk, illetve megtekinthetjük a legjobb időinket, ami alatt meg tudtuk oldani a kockát. Egyetlen negatívuma az alkalmazásnak, hogy nem ad módot kamerával való beolvasásra.



1.3. ábra. AZ Rubik's cube solver 3x3 illetve 5x5 kocka módban.

1.3. Következtetések

Bár a manuális beolvasás is eredményes módszer, mégis sokkal kényelmesebb a felhasználónak, ha nem kell minden oldalt manuálisan kitöltenie, hanem egyszerűen néhány fotót készít el, amit majd az alkalmazás értelmez. Ezen kívül pozitívum, ha az alkalmazás rendelkezik tanulást elősegítő funkciókkal is, illetve nem tartalmaz túl sok hirdetést.

2. fejezet

Felhasznált technológiák és programok bemutatása

2.1. Unity <https://unity.com/>

A Unity egy videojáték-motor, amelyet a Unity Technologies fejleszt és 2005-ben történt meg az első kiadása. [Tec] Unity segítségével 2D vagy 3D interaktív jellegű tartalmakat, vagy játékokat tudunk fejleszteni.

Megjelenése óta a folyamatos fejlesztéseknek köszönhetően szinte minden jelentős platformon megtalálható, köztük Android, iOS, Windows és Linux. Különösen népszerű Android és iOS alkalmazások fejlesztésére könnyed használhatósága miatt. 2018-ban Androidra kiadott játékok megközelítőleg felét Unityvel készítették.

2.2. C#

A C# egy rendkívül erőteljes és sokoldalú programozási nyelv és a Unity játékfejlesztő platformon belüli használata számos előnnyel rendelkezik.

Először is, a C# az alapvető programozási nyelv, amit a Unity támogat. Erős és intuitív programozási környezetet biztosít, amely tökéletesen integrálódik a Unity funkcióival és munkamenetével. Ez azt jelenti, hogy C# használatával a fejlesztők teljes mértékben kihasználhatják a Unity széles körű eszközeit, ami hatékony és gördülékeny játékfejlesztést tesz lehetővé.

A C# számos olyan funkciót és lehetőséget kínál, amelyek ideálisak a játékfejlesztéshez. Objektumorientált nyelv, amely elősegíti a kód szervezését és modularitását. Ez lehetővé teszi a tiszta, karbantartható és újrahasznosítható kód létrehozását, ami javítja a termelékenységet és a kód minőségét.

Egy másik jelentős előnye a C# használatának a Unity-ben a teljesítmény. A C# egy fordított nyelv, ami azt jelenti, hogy a kód végrehajtása előtt gép által olvasható utasításokra alakul át. Ez a fordítási folyamat optimalizált teljesítményt eredményez, ami ideális választássá teszi a erőforrásszintű játékfejlesztési feladatokhoz.

A C#-nak továbbá egy nagy és aktív fejlesztői közössége van. Ez azt jelenti, hogy a fejlesztők számos oktatóanyaghöz, dokumentációhoz és fórumokhoz férhetnek hozzá, amelyek a Unityre és a C# programozási nyelvre fókusznak. [Nor13] A közösségi tá-

mogatás elérhetősége megkönnyíti a hibajavítást és lehetőséget nyújt a tudásmegosztásra illetve más fejlesztőkkel való kollaborációra.

Ezenkívül a C# kiterjedt könyvtárakkal és keretrendszerrel rendelkezik, amelyek kihasználhatóak Unity-n belül. Legyen szó bevitel kezeléséről, mesterséges intelligencia megvalósításáról vagy összetett játékrendszerek létrehozásáról, a C#-könyvtárak lehetőségek széles skáláját kínálják a fejlesztőknek a fejlesztés felgyorsítására és a játékok funkcionálisának javítására.

Összefoglalva, a C# használata a Unity-ben számos előnyt kínál, ideértve a zökkenőmentes integrációt a Unity funkcióival, erős teljesítményt, kód szervezést és modularitást, aktív fejlesztői közösséget és hozzáférést a széles körű könyvtárakhoz és keretrendszerhez. Ezekkel az előnyökkel a C# lehetővé teszi a fejlesztők számára, hogy hatékonyan és eredményesen készítsenek lenyűgöző, magas minőségű szoftvereket.

2.3. Kociemba algoritmusa

Kociemba Rubik-kocka megoldó algoritmusa egy rendkívül hatékony és elismert módszere a kocka kirakására. [Koc] Herbert Kociemba, egy német számítógép tudós és matematikus által fejlesztett algoritmus, amely a kivételes sebességről és optimális megoldások gyors megtalálásáról híres. Kociemba algoritmusának alapja a csoportelmélet és különböző kifinomult technikákat alkalmaz a feladvány bonyolultságának kezelhető lépésekre bontásához.

A 43 kvintillió lehetséges kombinációval rendelkező Rubik-kocka optimális kirakása látszólag meghaladta az emberi képességeket, egészen addig, amíg fejlett algoritmusok, mint a Kociemba algoritmus, nem jelentek meg. Az algoritmus rövid idő alatt megtalálja a legkevesebb lépést igénylő optimális megoldást bármilyen összekevert Rubik-kocka állapotra.

Kociemba algoritmusának kétlépéses megközelítése van a feladvány megoldására. Az első lépésben a cél az első két réteg kirakása, miközben minimalizálja a lépések számát. Ezt a lépést heurisztikus módszerekkel és hatékony metszötáblákkal éri el, amelyek jelentősen csökkentik a keresési teret. Miután az első két réteg megoldódott, az algoritmus a második lépésnek kezd, ahol a hátralévő réteget oldja meg, ezzel a teljes kocka kirakását befejezve.

Kociemba algoritmusának egyik jelentős jellemzője az optimális megoldások megtalálása. Ezzel garantálja a lehető legrövidebb megoldás megtalálását, a mely a legkevesebb lépést igényli. Ez a tulajdonsága világszerte szeretetté és keresetté teszi úgy a játék hétköznapi felhasználói között, mint a sebességgkirakók körében.

Kociemba algoritmusának nemcsak a számítástechnika és a matematika területén van jelentősége, hanem jelentős hatással volt a versenyszintű kocka megoldás világában is. Hatékonysága és pontossága forradalmasította a módját, ahogy az emberek megközelítették és megoldották a kocka kirakásának problémáját.

A munkámban Kociemba 2 lépéses algoritmusának egy C#-ba portolt nyílt forráskódú változatát használtam (<https://github.com/Megalomatt/Kociemba>).

2.4. Visual Studio <https://visualstudio.microsoft.com/>

Nagyon népszerű több programozási nyelvet is tartalmazó fejlesztőkörnyezet [Mic]. Kiemelkedő előnyei:

- átlátható, letisztult felület
- fejlett kódkiegészítő lehetőségek
- személyreszabhatóság
- számtalan használati utasítás

2.5. GitHub Desktop <https://desktop.github.com/>

A GitHub vizuális felüettel is ellátott alkalmazása [Git], nagyban megkönnyíti a verziókövetést és kiválóan használható Unityben fejlesztendő projektekkel.

3. fejezet

Funkcionális és nem funkcionális követelmények

3.1. Funkcionális követelmények

- Kocka szabad módon történő forgatása
 - Teljes kocka 3D térben történő forgatása
 - Kocka oldalainak egyenként való forgatása
- Teljes kocka állásának egy 2D reprezentációról való leolvasása
- Kocka kamerával való beolvasása
- Kocka véletlenszerű összekeverése
- Kocka optimális megoldása
- Megoldó képletek kockára való alkalmazása
 - Különböző megoldó képletek között való váltás
- Kocka alaphelyzetbe való állítása

Az alábbi use-case diagram bemutatja a felhasználó lehetőségeit: [3.1](#)



3.1. ábra. Alkalmazás eset diagramja

3.2. Nem funkcionális követelmények

- Egyszerű használhatóság
- Optimális kocka megoldó sebesség
- Kényelmes kezelés

4. fejezet

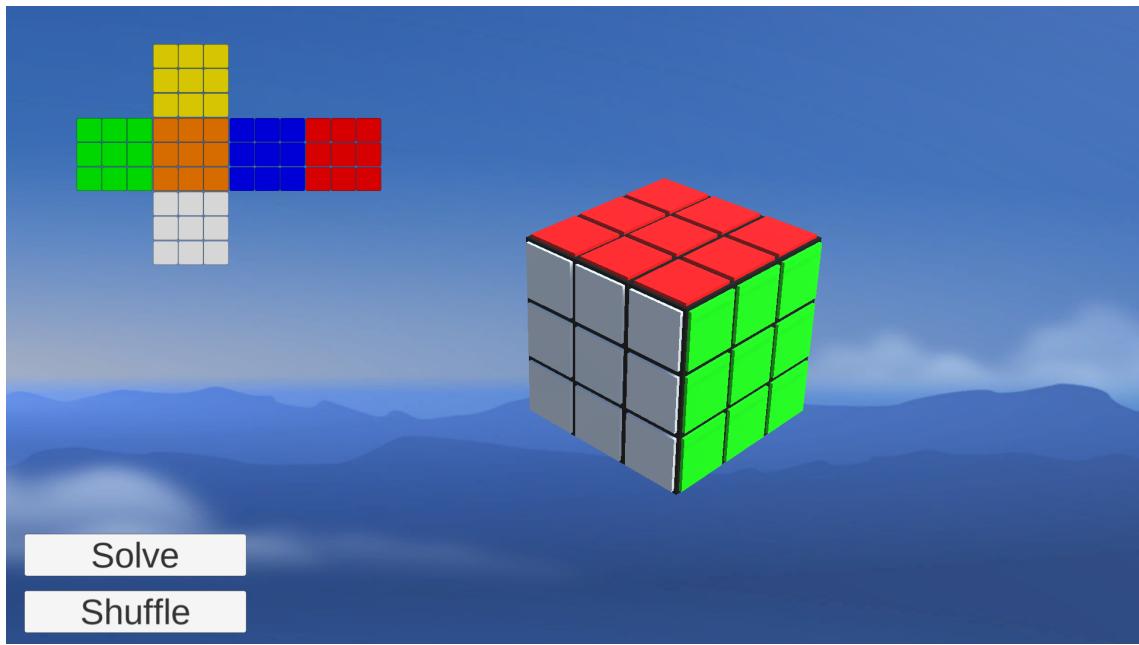
Alkalmazás működésének leírása

4.1. Létező program, mint alap

Az általam megírt államvizsgaprojekt egy már korábban megírt alkalmazást vesz alapul, amelyet a Megalomobile YouTube csatorna készített [Meg]. Ezt az alapot fejlesztettem tovább különböző funkcionálisokkal, javítottam ki meglévő hibáit illetve áthelyeztem Androidos eszközökre. Általam hozzáadott funkcionálisok:

- Kocka beolvasása az eszköz kamerájának segítségével
- Kamerával történő beolvasás esetén lehetőség nyújtás adott oldalak újraolvasására
- Felhasználói felület bővítése
- Előre eltárolt megoldó algoritmusok alkalmazása a kockára
- Lehetőség automatikus és manuális lépésvégrehajtási módszer között változtatni
- Megoldások megjelenítése text formátumban
- Kocka alaphelyzetbe állítása

Az alapul vett szoftver kinézete az alábbi képen megtekinthető: [4.1](#). Az általam kibővített applikáció kinézete az alábbi képeken megtekinthető: [6.2](#), [6.3](#).

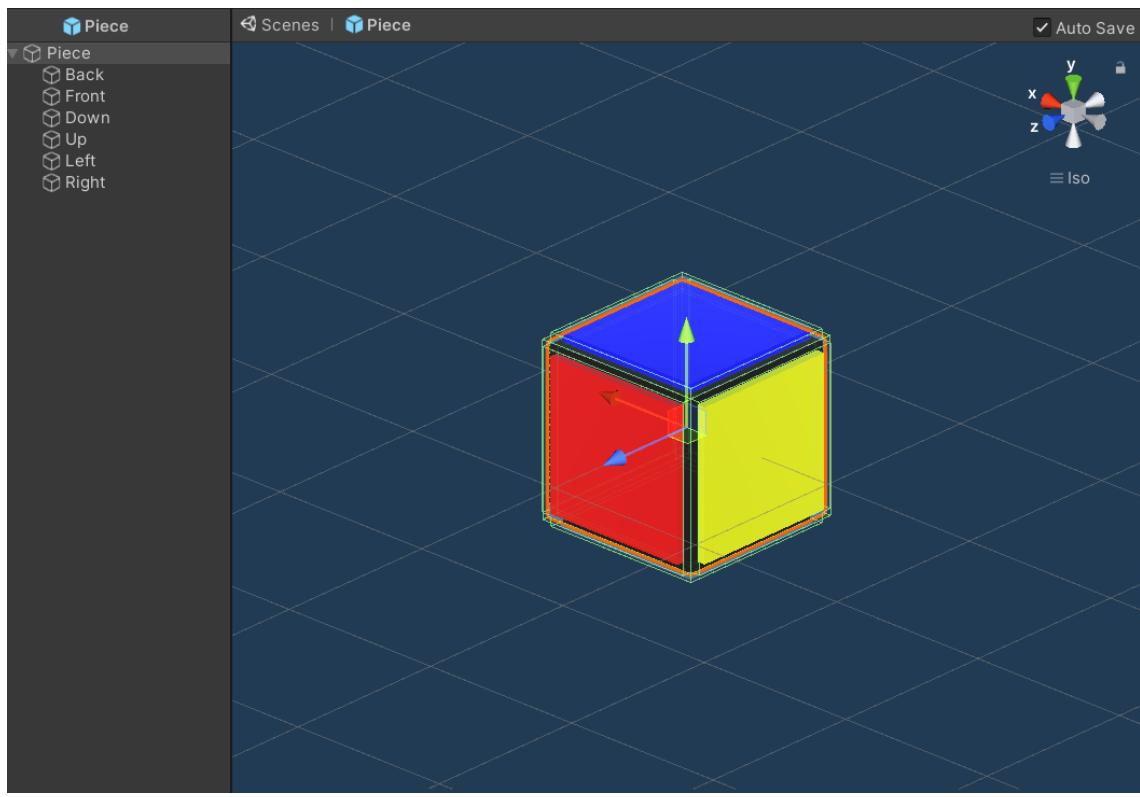


4.1. ábra. Alapul vett program kinézete.

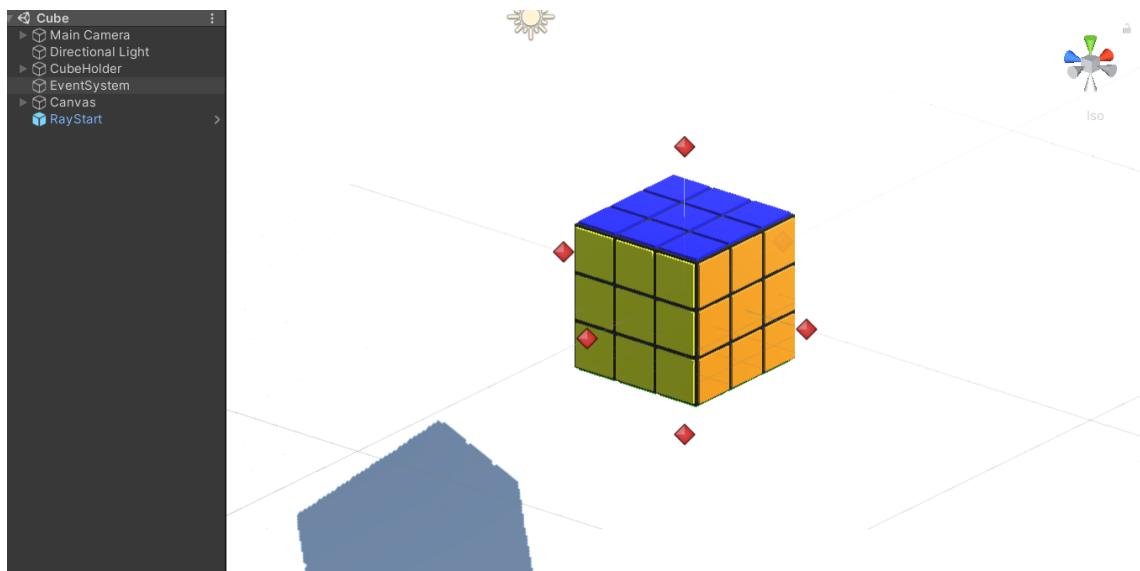
4.2. Rubik-kocka megjelenítése 3D térben

A kocka meggyőző megjelenítése az egyik legfontosabb feladat, mivel ez a leglátványosabb része az alkalmazásnak és ez az a dolog, amivel a felhasználó a leggyakrabban fog interaktálni.

A teljes Rubik-kocka 26 kisebb darabból áll. Unity lehetőséget ad úgynevezett Prefabok létrehozására ami nagyban megkönnyíti azonos objektumok létrehozását. A Prefab a Unity játékmotorban egy előre elkészített objektum/objektumcsoport, amelyet többször is felhasználhatunk. Lényegében egy sablon. Ez a Prefab tartalmazza a darabot reprezentáló objektumot, ami egy fekete 1x1x1-es négyzet és a 6 gyermekobjektumát, amik az oldalain találhatóak, ezek adják a darab oldalainak a színét. Ez a prefab megtekinthető az alábbi ábrán [4.2](#) A 26 darab prefab lehelyezése után, már megjelenik előttünk egy felismerhető Rubik-kocka [4.3](#).



4.2. ábra. Prefab.



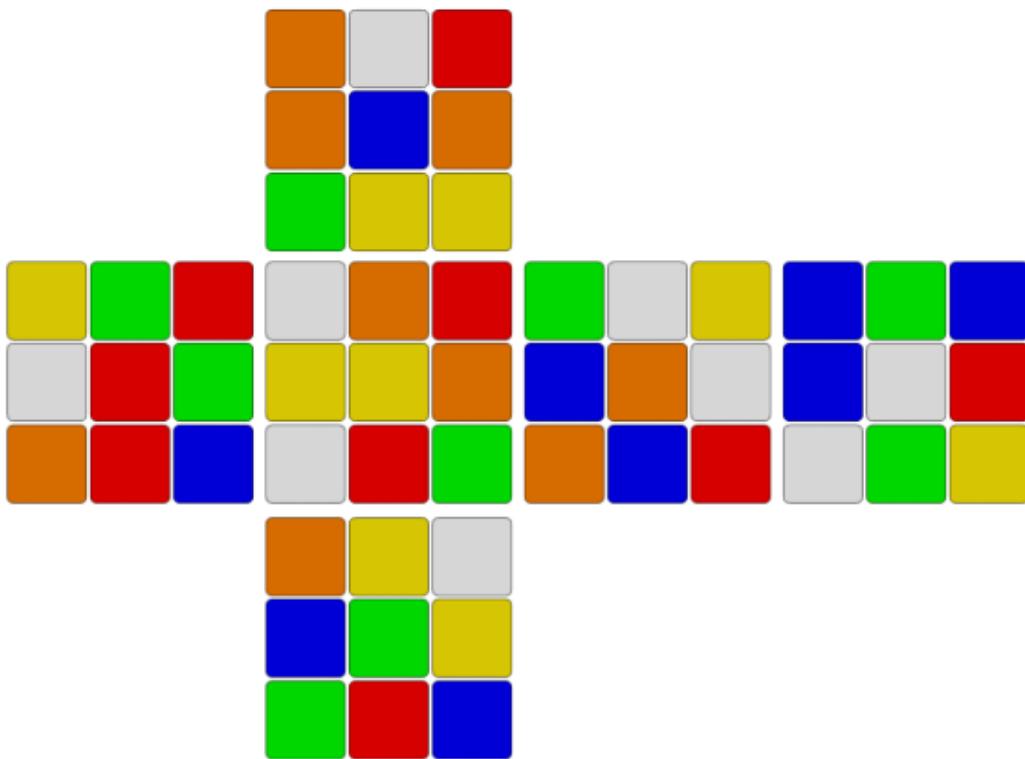
4.3. ábra. A prefabokból összeállított kocka.

4.3. Rubik-kocka 2D reprezentációjának megjelenítése

Annak érdekében, hogy a felhasználó minden oldal helyzetét láthassa egyidőben, szükségünk van a kocka kivetítésére 2D-be. Ennek a kinézetét megtekinthetjük az alábbi ábrán: [4.4](#) Az adott oldal állásának leolvasása Unity Raycastok segítségével működik.

Unityben a Raycast egy fizikai alapú módszer, amelyet objektumok detektálására használhatunk egy adott irányban a 3D térben. A módszer lényege, hogy egy képzeletbeli sugarat vetítünk ki egy pontból a térben, majd megvizsgáljuk, hogy metszi-e bármelyik objektumot.

Minden oldalra 9 sugarat vetítünk a kocka irányába, minden darabra egyet, amelyek ütközni fognak minden darab oldalán található színnel, amit így be tudunk állítani a 2D reprezentáció megfelelő részén.



4.4. ábra. A kocka 2D térbe való kivetítése.

4.4. Rubik-kocka beolvasása az eszköz kamerájának segítségével

Ha az applikációt olyan eszközön futtatjuk, amelynek van elérhető kamerája, a felhasználó ezt használva egyszerűen és gyorsan beolvashatja a saját Rubik-kockájának állását. Annyit kell tennie, hogy a képernyőjén megjelenő rácsot ráhelyezze a saját kockájára és kiválassza melyik oldalt szeretné éppen beolvasni.

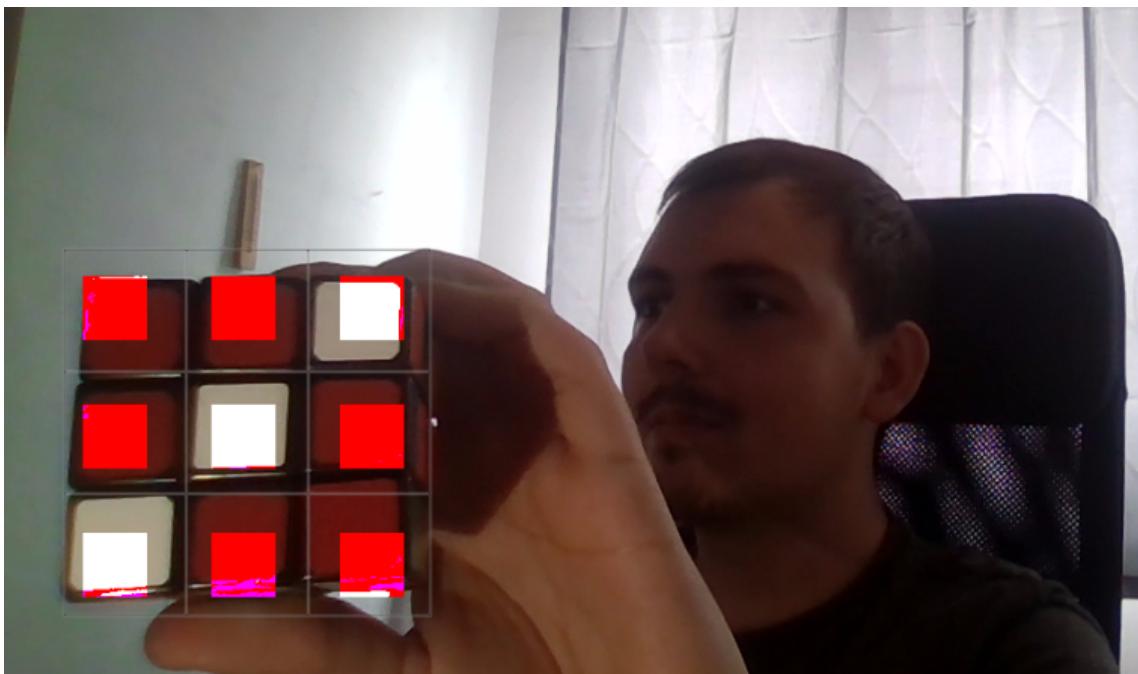
Az applikáció végez egy fotót, a rácsot 9 szekcióra osztja, és mindenik szekcióból adott mennyiségű pixel színét átlagolja.

A pontos eredmény érdekében szükséges a jó képminőség, de mivel ez nem garantált ki kell szűrnünk a hibák nagy százalékát. Ilyen eseteket okozhat szennyeződés a Rubik-kocka felületén, kamera minőségi hibája vagy a fény kedvezőtlen csillanása a kocka oldalán.

Egy szekció színének átlagát ezután össze kell hasonlítani a Rubik-kocka előre meghatározott 6 színével. Ezen művelet elvégzése többféle módon is elvégezhető, több iteráción átesett az applikáció, amíg kellő megbízhatósággal kezdett működni.

A színpelismerő algoritmus részletesebb leírása megtalálható a következő fejezetben: [5.3.](#)

Az alábbi képen demonstrálom a színbeolvasás működését debug módban [4.5](#). Itt jól látszik, hogy bár a kocka nem illeszkedett tökéletesen a sablonra, de mégis el tudjuk már ennyi adat alapján is dönten a helyes színeit a daraboknak. Mivel nem várhatjuk el, hogy a felhasználó minden tökéletesen rátávolítsa a megadott rácsra a kockáját, ezért csak a közepét dolgozom fel minden négyzetnek.



4.5. ábra. A kocka beolvasása kamerával.

5. fejezet

Színterminálás algoritmus

5.1. Sikertelen színterminálási próbálkozások

5.1.1. Távolság 2 RGB szín között

A legegyszerűbb megoldás az, ha a kocka 6 darab színét eltároljuk egy listában, majd a beolvasott kép adott pixeljeit egyenként hasonlítsuk ezekhez matematikai távolság alapján. Amelyik a legközelebb volt azt eltároljuk, majd végigmegyünk az eltárolt listában lévő színeken és amelyik a leggyakrabban szerepel az találat.

$$\text{távolság} = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

Ez a megoldás jónak tűnik első látásra, bár mégsem működik. Ennek az oka az, hogy az RGB színtároló módszer nem "érzékelési szempontból egységes". Emiatt az euklideszi RGB távolság nem fog megegyezni az ember által érzékelt színek közötti távolsággal. Bár az esetek nagy százalékában ez a módszer is működhet, de a narancssárga, piros és sárga színek között szinte lehetetlen magabiztos döntést hozni.

Számos kísérlet történt az RGB-értékek mérlegelésére, hogy jobban illeszkedjenek az emberi szem érzékeléséhez, ahol az összetevőket általában súlyozzák (piros 30%, zöld 59%, kék 11%), azonban még így sem tudunk egy magabiztos meghatározást végezni, mivel gyakori a hiba sarokesetekben.

Az egyik hasonló megközelítés, amit "redmean"-nek is neveznek hasonló elven működik.

$$\bar{r} = \frac{1}{2}(R_1 + R_2)$$

$$\Delta C = \sqrt{\left(2 + \frac{\bar{r}}{256}\right) \Delta R^2 + 4\Delta G^2 + \left(2 + \frac{255 - \bar{r}}{256}\right) \Delta B^2}$$

5.2. RGB vs HSV

Korábban elvétett hibáimból tanulva elvettem a színek RGB színtérben való kezelését. Az új algoritmusom is hasonló módon működik, de egy fontos részletben tér el, RGB színtér helyett HSV színtérrel dolgozok.

A HSV (vagy HSB) színtér egy színtároló modell, amely az RGB színtérhez hasonlóan színinformációkat tárol, de a színek leírását eltérő módon végzi. Az HSV színtér három fő komponensből áll: az árnyalat (hue), a telítettség (saturation) és világosság (value/brightness).

Az árnyalat komponens az alapszín vagy színtónus meghatározásáért felelős, és a 0-360 fok közötti tartományba esik. A telítettség komponens a szín tisztaságát vagy intenzitását jelöli, és értéke 0 és 100% között változhat. Végül a világosság komponens az adott szín fényességének szintjét határozza meg, és szintén értéke 0 és 100% között mozog.

Az HSV színtér előnye, hogy könnyen érthető és könnyen kezelhető színtároló rendszer, amely lehetővé teszi a színárnyalatok intuitív kezelését és módosítását. [SCB87] Az RGB színtérhez képest jobban megfelel az emberi szem színérzékelésének, és könnyebben lehet vele színváltoztatást, árnyalatmódosítást kezelní, valamint színeket összehasonlítni. Ezért a HSV színtér széles körben használatos számítógépes grafikában, képfeldolgozásban és színtervezésben.

5.3. Általam készített színterhelmerő algoritmus

Az algoritmus első lépésként készít egy képernyőképet a kamera felvételéről, majd az ezen elhelyezett keret koordinátái alapján felosztja a képet 9 szekcióra, azaz a kocka oldalának 9 darabjára. Ezután minden szekcióból kiválasztunk négyzethálósan 2500 darab pixelt. Ezeket RGB színtérben kapjuk meg, de Unity segítségével tudjuk konvertálni HSV színtérbe.

Mivel az eszköz kamerája, amivel készül a kép nem mindig eredményez egy magas minőségű képet és a fényviszonyok sem minden helyzetben optimálisak, valószínű, hogy lesznek hibásan megítélt pixelek. A minél pontosabb becslés érdekében ezért a beolvadt 2500 darab színértékből számolok egy átlagot és azokat az értékeket nagy mértékben eltérnek ettől eldobom, így biztosítva, hogy legalább a nagyobb képhibákat ki tudjuk küszöbölni.

Ezután végigmegyünk a megmaradt színeinken, mindenket hasonlítjuk a kocka 6 színéhez és eltároljuk a hozzá legközelebb állót. Miután megtettük ezt mindenik beolvadt pixelre a leggyakrabban megjelent színt visszatérítjük.

Egy szín meghatározását határértékek segítségével oldottam meg. Mind a 6 színnek meghatároztam 6 darab határértéket olyan módon, hogy ne ütközzenek ezek egymással. Ennek segítségével egy találat megállapítása néhány egyszerű összehasonlítási művelet. Ehhez a lépéshez felhasznált kód az alábbi részletben megtékinthető: 5.3

```

1 public short[,] InRange(HSVColor[,] input, HSVColor upperTreshold, HSVColor lowerTreshold)
2 {
3     short[,] output = new short[50, 50];
4
5     for (int i = 0; i < 50; ++i)
6     {
7         for (int j = 0; j < 50; j++)
8         {
9             if (input[i, j].H > 1 || input[i, j].S > 1 ||
10                 input[i, j].V > 1 || input[i, j].H < 0 ||
11                 input[i, j].S < 0 || input[i, j].V < 0)
12             {
13                 Debug.Log("HSV InRange ERROR!");
14                 Debug.Assert(true);
15             }
16
17             output[i, j] = 0;
18
19             if (input[i, j].H <= upperTreshold.H && input[i, j].H >= lowerTreshold.H &&
20                 input[i, j].S <= upperTreshold.S && input[i, j].S >= lowerTreshold.S &&
21                 input[i, j].V <= upperTreshold.V && input[i, j].V >= lowerTreshold.V)
22             {
23                 output[i, j] = 1;
24             }
25         }
26     }
27
28     return output;
29 }
```

A fenti függvényben 5.3 az input paraméter egy 50*50 méretű mátrixot vár el, melynek elemei HSV színtároló módszerben kódolt színek, illetve az upperTreshold és lowerTreshold paraméter a két határérték. Az input paraméter a beolvasott rács egy négyzetéről kiolvasott pixelek színeit tartalmazza. A rács minden kis négyzetére külön hívódik meg ez a függvény, illetve minden szín különböző határértékére szintén. Ez azt jelenti, hogy összesen egy oldal beolvasásához ez a függvény 63-szor hívódik (9 négyzet és 7 különböző szín határérték pár. A piros színnek két határérték pár szükséges, mivel a skála eleje és vége is piros).

Az 5. és 7. sorban lévő ciklusokkal iterálunk a mátrixon. A 9. sorban található feltétel a mátrix adott pozicióján lévő szín HSV paramétereinek helyességét vizsgálja, mivel ezek százalékokként vannak eltárolva, nem haladhatják meg az 1-et és nem lehetnek negatívak. A 19. sorban lévő feltétel vizsgálja meg egyszerű összehasonlításokkal, hogy a mátrix adott poziciójában lévő szín a két határértékként megadott szín között fekszik-e, ha igen, eltárol az eredményként visszatérített mátrix adott pozícióján egy 1-est, másképp egy 0-át.

A határértékként felhasznált színek meghatározása nagyon sok munkát igényelt. Különböző eszközökkel készítettem fotókat Rubik-kockákról, majd minden színnek egyenként megpróbáltam egy minél tágabb intervallumot kiválasztani, olyan módon, hogy ne ütközzenek. Az alábbi képen láthatunk néhány eltárolt színhatárértéket: 5.1.

```

// green treshold
HSVColor upperThresholdGreen;
upperThresholdGreen.H = 185;
upperThresholdGreen.S = 100;
upperThresholdGreen.V = 100;
upperThresholdGreen = HSVtoPercentileHSV(upperThresholdGreen);

HSVColor lowerThresholdGreen;
lowerThresholdGreen.H = 90;
lowerThresholdGreen.S = 50;
lowerThresholdGreen.V = 15;
lowerThresholdGreen = HSVtoPercentileHSV(lowerThresholdGreen);

// orange treshold
HSVColor upperThresholdOrange;
upperThresholdOrange.H = 40;
upperThresholdOrange.S = 100;
upperThresholdOrange.V = 100;
upperThresholdOrange = HSVtoPercentileHSV(upperThresholdOrange);

HSVColor lowerThresholdOrange;
lowerThresholdOrange.H = 10;
lowerThresholdOrange.S = 50;
lowerThresholdOrange.V = 15;
lowerThresholdOrange = HSVtoPercentileHSV(lowerThresholdOrange);

// red treshold
HSVColor upperThresholdRed;
upperThresholdRed.H = 360;
upperThresholdRed.S = 100;
upperThresholdRed.V = 100;
upperThresholdRed = HSVtoPercentileHSV(upperThresholdRed);

HSVColor lowerThresholdRed;
lowerThresholdRed.H = 345;
lowerThresholdRed.S = 50;
lowerThresholdRed.V = 10;
lowerThresholdRed = HSVtoPercentileHSV(lowerThresholdRed);

```

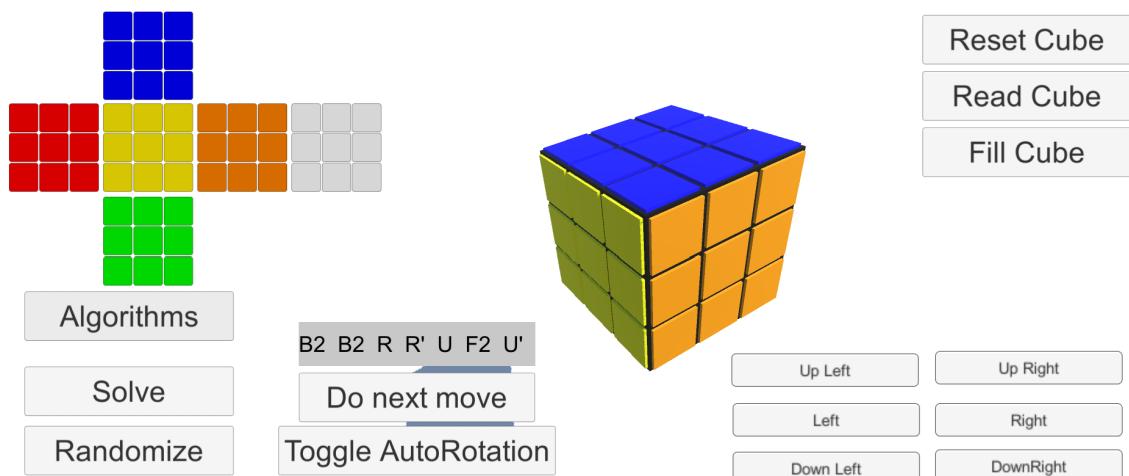
5.1. ábra. Zöld, narancs és piros színek határérték pájai

6. fejezet

Az alkalmazás felhasználói szempontból

6.1. A kezdőoldal

Amikor a felhasználó megnyitja az alkalmazást, ez az oldal fogadja 6.1. Itt megtalálhatja a végrehajtható lehetőségek nagy százalékát. A felhasználói felület egyszerű, gyorsan megérthető, középen található a kocka, amit forgathatunk, és körülötte a különböző funkciókat alkalmazó gombok.



6.1. ábra. Az alkalmazás kezdőoldala.

6.1.1. A kocka forgatása

A felhasználó attól függően, hogy melyik oldalát érinti meg a kockának, azt forgathatja az újja mozdításával. A bal felső sarokban található 2D reprezentációja a kockának ilyenkor minden frissül, hogy pontosan tükrözze a kocka jelenlegi állását. Azért van erre szükség, mivel a felhasználók nagy százalékának segít, ha nyomon tudja követni minden oldal állását, nem csak az általa látható szemben lévő hármat.

Abban az esetben, ha a kockát a térben akarja forgatni a felhasználó, a jobb alsó sarokban található 6 gomb közül kell használnia a neki megfelelőt. Ezek a gombok 90 fokban forgatják a teljes kockát adott irányban, így lehetőséget adnak a felhasználónak a kocka jobb átlátására.

6.1.2. A kocka összekeverése és megoldása

A bal alsó sarokban található két gomb segítségével a felhasználó összekeverheti a kockát véletlenszerűen, vagy kikérhet egy optimális megoldást, melyet az alkalmazás eltárol egy lépéskövető listában, amelynek lépéseiit a felhasználó egyszerre, vagy lépésenként is végrehajthatja.

6.1.3. Betárazott lépések listája

A következő lépést végrehajtó gomb fölött található egy lista, amelyben a következő 7 betárazott lépést láthatjuk. A következő lépés gomb megnyomása esetén, végrehajtódik egy lépés a listából és a lista állapota frissül.

6.1.4. Automatikus forgatás ki/be kapcsolása

A képernyő alján középen található gombbal tudja a felhasználó kiválasztani a forgatás működésének módját. Abban az esetben, ha automatikus forgatásra van állítva, az alkalmazás gyors egymásutánban végzi el a lépéseket. Ez alkalmas akár véletlenszerű összekeverés gyors végrehajtására, vagy a kocka gyors alaphelyzetbe való állítására. Abban az esetben viszont, ha kikapcsoljuk az automatikus forgatást, csak akkor fog végrehajtani egy lépést az alkalmazás, ha a felhasználó gombnyomással adja ki a parancsot.

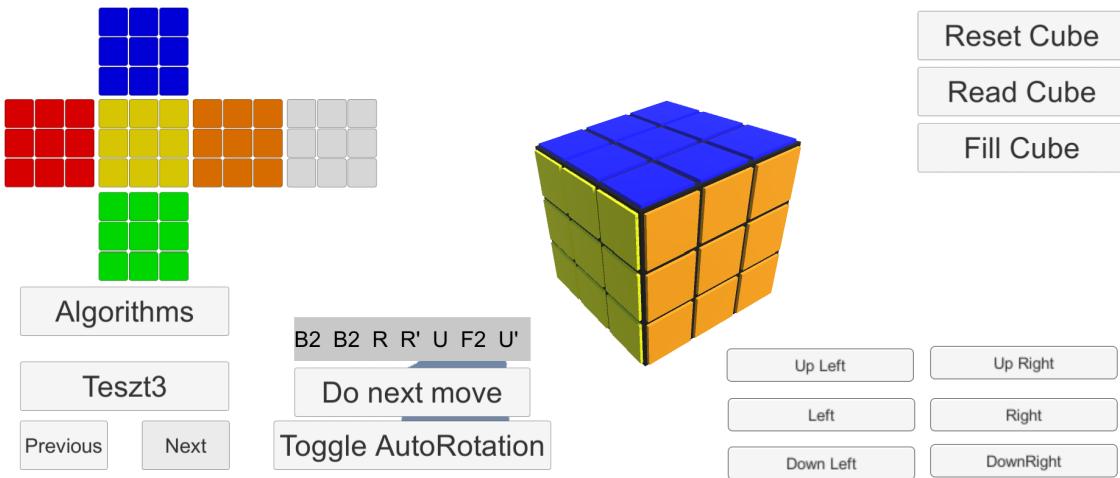
6.1.5. Algoritmusok nézet

A képernyő bal oldalán található algoritmusok gombbal tud átváltani a felhasználó az algoritmusok nézetre. A következő képen látható ennek kinézetele [6.2](#).

A felhasználó a bal alsó sarokban lévő két gombbal navigálhat a létező képletek között. A képlet nevét hordozó gombra (jelen esetben Teszt3) kattintva betározza a lépés listába a képletet. Ilyen módon felhasználhatja a kocka megoldására, vagy akár lépésenként le is másolhatja, így segítve ennek memorizálásában.

6.1.6. Kocka kameráról beolvasott állásra, vagy kezdő állásra állítása

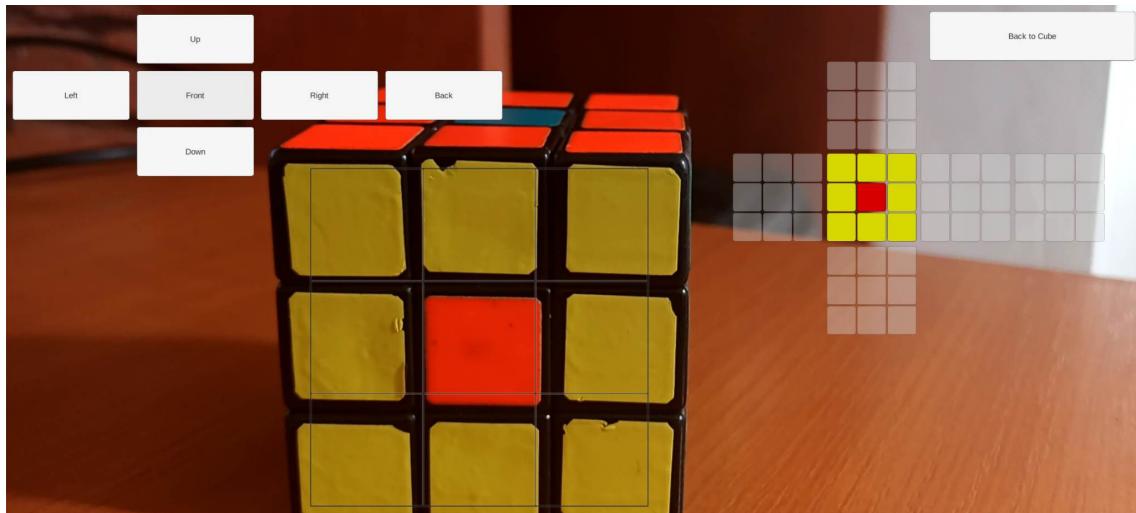
A jobb felső sarokban található gombok segítségével a felhasználó azonnal alaphelyzetbe állíthatja a kockát, vagy a kamerával beolvasott és eltárolt állást alkalmazhatja. Ez a funkcionalitás egy későbbi alfejezetben bemutatásra kerül bővebben.



6.2. ábra. Az alkalmazás kezdőoldala algoritmusok mód bekapcsolása esetén.

6.1.7. Kocka beolvasása kamerával

A kezdőoldal jobb felső részén található gomb segítségével átválthatunk a kamerával való beolvasásra. Ennek kinézete az alábbi ábrán megtekinthető: [6.3.](#)



6.3. ábra. Az alkalmazás kezdőoldala algoritmusok mód bekapcsolása esetén.

Ezen az oldalon a felhasználó a bal felső sarokban található gombokkal adhatja meg, éppen melyik oldalt szeretné beolvasni és a beolvasás eredményét a jobb oldali reprezentáción ellenőrizheti. Ha befejezte a beolvasást, a jobb felső sarokban található gombbal navigálhat vissza a kezdőoldalra, ahol a beolvasott eredményt alkalmazhatja.

Abban az esetben, ha valamelyen okból kifolyólag nem pontos a beolvasás egy adott oldalon, a felhasználó a kocka 2D reprezentáció egy adott négyzetére kattintva megváltoztathatja annak színét, így javítva a hibás beolvasást.

7. fejezet

Mérések

7.1. Megoldás megtalálásának időtartama

Az alkalmazás sebességét legfőképp befolyásoló tényező a megoldás lépéseinek meghatározása. Szerencsére Kociemba algoritmusa nagyon gyorsan meg tudja találni a megoldást. A méréseket a C# Stopwatch osztályal végeztem, ez a beépített osztály lehetőséget ad futási idő pontos mérésére.

A méréseket egy kellően összekevert kockán végeztem el, a mért időtartam tartalmazza a kocka állapotának leolvasását és a helyes megoldás megkeresését. A mérés eredményei az alábbi táblázatban megtékinthetők: [7.1](#)

A táblázat alapján észlelhető, hogy átlagosan 48.5 milliszekundum időbe kerül a megoldáshoz szükséges lépések megtalálása és eltárolása az alkalmazásban. A szórás 31.3 és a legtöbb időt igénybe vett megoldás megtalálása 158 milliszekundumot igényelt. A kocka véletlenszerű keverései közül léteznek állások, amik kedveznek a megoldó algoritmus számára, így nagyon gyorsan, akár kevesebb mint 10 milliszekundum alatt is megtalálhatja a megoldást, de igaz ennek az ellenkezője is, ez magyarázza a magas szórást.

7.2. Fejlesztéshez felhasznált rendszer specifikációi

Mivel a mért eredményeket nagyban befolyásolja a használt rendszer specifikációja, fontostak tartom leírni az általam használt rendszer tulajdonságait.

Az alkalmazás végleges verzióját kipróbáltam különböző Android telefonokon, nem észleltem bármiféle lassulás.

A lemért időkre leginkább kiható specifikációk listája:

- **CPU:** AMD Ryzen 5 5600H
- **GPU:** NVidia RTX 3050 4 Gb
- **RAM:** 16 GB DDR4
- **OS:** Windows 10 Pro 64 bit

7.1. táblázat. Megoldó lépések megtalálása

Megoldó lépések megtalálása	
Idő(milliszekundum)	Lépések száma
4	20
93	22
65	20
18	19
70	21
13	21
82	21
6	20
3	20
13	21
62	20
32	19
50	20
54	21
72	22
94	20
47	21
158	22
8	19
30	20

8. fejezet

Továbbfejlesztési lehetőségek

Az alkalmazást nagyon sok irányban lehetne tovább fejleszteni, kinézetén finomítani.

- Különböző méretű kockákat is választhatóvá tenni
- Automatizálni az oldalak beolvasását
- Szebbé tenni a kezelőfelületet
- Beolvasásnál keresendő színek fényviszonyok szerinti kalibrálása
- Személyre szabható kocka színek
- Lehetővé tenni a felhasználó számára saját megoldóképletek hozzáadását a meglévőkhöz, illetve ezek törlését

Jelenlegi formájában az alkalmazás kitűnő alapot képez, amelyre a már létező függvények/metódusok segítségével könnyedén lehet építkezni.

Összefoglaló

Dolgozatomban egy Rubik-kocka megoldó alkalmazással foglalkoztam, amelyet Unity játékmotorban valósítottam meg. Számomra az egyik legnagyobb feladatot a versenyszerű Rubik kocka megoldás világának megismerése jelentette, főképpen az algoritmusokkal foglalkozó része, illetve a pontos színpelismerő módszerek megértése. Ugyan már használtam Unity játékmotort illetve C# programozási nyelvet, de az eddigi projektjeim sokkal kisebb méretűek voltak.

Kifejezetten érdekes volt megismerni a versenyszerű Kocka megoldás világát, ennek fejlődését az évek során. Ahogyan egyre jobban magával ragadt ez világ, egyre könnyebben tudtam új funkcionálisokat implementálni az alkalmazásba illetve a meglévőket fejleszteni. Alkalmazásom főbb funkciói a kamerával való beolvasása egy kockának, optimális megoldás kikérése illetve algoritmusok alkalmazása a kockára.

Az alkalmazásom forráskódja elérhető a következő linken:
<https://github.com/KolumbanA/RubiksCubeSolver>

Köszönetnyilvánítás

Köszönöm témavezetőmnek Iclánzan Dávidnak a segítségét és hogy lehetőséget biztosított számomra a szakdolgozat megírásához.

Ábrák jegyzéke

1.1.	Rubik's Cube Solver kinézete.	2
1.2.	AZ Rubik's cube solver 3x3 illetve 5x5 kocka módban.	3
1.3.	AZ Rubik's cube solver 3x3 illetve 5x5 kocka módban.	4
3.1.	Alkalmazás eset diagramja	9
4.1.	Alapul vett program kinézete.	12
4.2.	Prefab.	13
4.3.	A prefabokból összeállított kocka.	13
4.4.	A kocka 2D térbe való kivetítése.	14
4.5.	A kocka beolvasása kamerával.	15
5.1.	Zöld, narancs és piros színek határérték párajai	19
6.1.	Az alkalmazás kezdőoldala.	20
6.2.	Az alkalmazás kezdőoldala algoritmusok mód bekapcsolása esetén.	22
6.3.	Az alkalmazás kezdőoldala algoritmusok mód bekapesolása esetén.	22
F.1.1A	Unity játékmotor felülete	31

Táblázatok jegyzéke

7.1. Megoldó lépések megtalálása	24
--	----

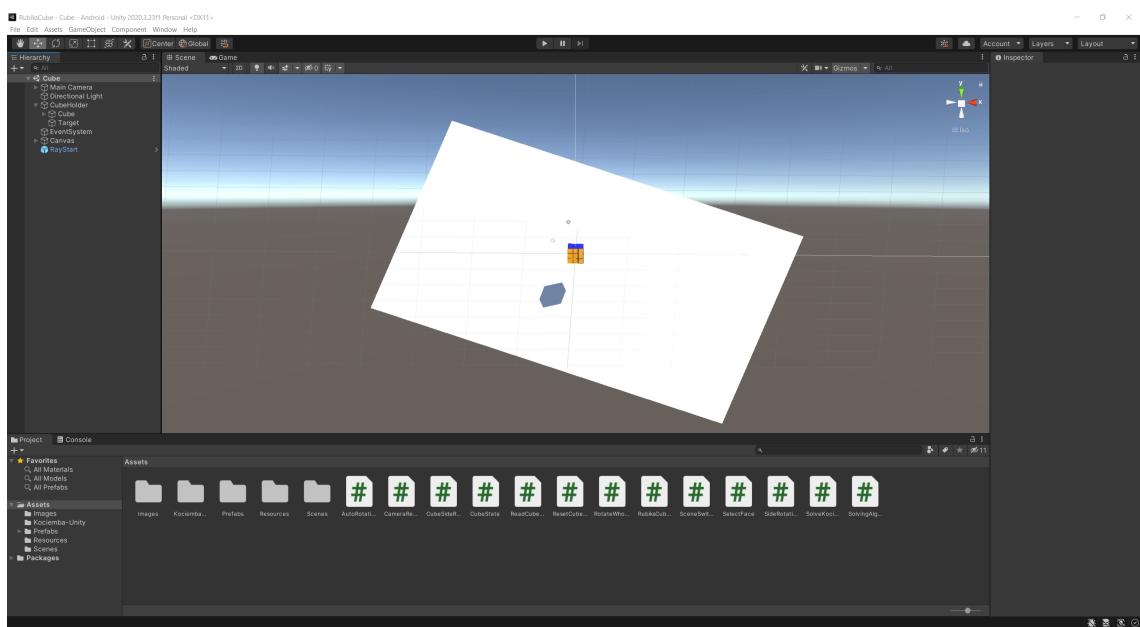
Irodalomjegyzék

- [Git] Inc GitHub. Github desktop.
- [Koc] Herbert Kociemba. Herbert kociemba's homepage.
- [Meg] Megalomobile. Megalomobile youtube channel.
- [Mic] Microsoft. Visual studio community edition.
- [Nor13] Terry Norton. *Learning C# by developing games with unity 3D*. Packt Publishing Ltd, 2013.
- [SCB87] Michael W Schwarz, William B Cowan, and John C Beatty. An experimental comparison of rgb, yiq, lab, hsv, and opponent color models. *Acm Transactions on Graphics (tog)*, 6(2):123–158, 1987.
- [Tec] Unity Technologies. Unity game engine.

Függelék

F.1. A Unity játékmotor felülete

<https://unity.com/> A Unity játékmotor, amely nagyon sok hasznos funkciót tartalmaz. F.1.1



F.1.1. ábra. A Unity játékmotor felülete