Posudek bakalářské práce

Matematicko-fyzikální fakulta Univerzity Karlovy

Autor práce	Monika Bošániová
Název práce	Rubikova kocka
Rok odevzdání	2022
Studijní program	Informatika
Specializace	Programování a softwarové systémy

Autor posudku	Mgr. Pavel Ježek, Ph.D.	**Role**	Oponent
Pracoviště	UK MFF KDSS		

Prosím vyplňte hodnocení křížkem u každého kritéria. Hodnocení *OK* označuje práci, která kritérium vhodným způsobem splňuje. Hodnocení *lepší* a *horší* označují splnění nad a pod rámec obvyklý pro bakalářskou práci, hodnocení *nevyhovuje* označuje práci, která by neměla být obhájena. Hodnocení v případě potřeby doplňte komentářem. Komentář prosím doplňte všude, kde je hodnocení jiné než *OK*.

K celé práci	lepší	OK	horší	nevyhovuje
Obtížnost zadání	☐	☒	☐	☐
Splnění zadání	☐	☐	☐	☒
Rozsah práce *... textová i implementační část, zohlednění náročnosti*	☐	☒	☐	☐
Komentář Zadání práce je velmi stručné, a není vlastně přesně jasné, co si zadavatel od výsledné práce slibuje. Nicméně i tak práce nesplňuje netriviální část zadání, když podporuje právě a pouze kostku 3x3, ale žádné další hlavolamy. Zásadní ale je, že autorka toto v textu práce zcela pomíjí, a jediná zmínka o jiných hlavolamech se „nenápadně“ nachází ve formě odrážky „pridanie viacerých hlavolamov (napr. kocky 2x2, 4x4...)“ bez dalšího vysvětlení v části „Plány do budúcnosti“ na konci práce. Nesplnění této části zadání by bylo ještě pochopitelné, pokud by s tím autorka počítala, a na budoucí rozšíření by práce byla připravena – jak ale uvádím dále v příloze posudku, tak stávajícím kódem aplikace ani zadání splnitelné není, protože autorka má vlastnosti kostky 3x3 velmi natvrdo „zadrátované“ ve většině kódu, a i jen pro podporu hlavolamů velmi podobných 3x3, tedy např. 2x2 nebo 4x4 by musel být celý kód naprogramován zcela od začátku (a to již nezmiňuji komplexnější hlavolamy jako pyramix nebo megamix, které nepoužívají dílky ve tvaru krychliček).				

Textová část práce	lepší	OK	horší	nevyhovuje
Formální úprava *... jazyková úroveň, typografická úroveň, citace*	☐	☐	☐	☒
Struktura textu *... kontext, cíle, analýza, návrh, vyhodnocení, úroveň detailu*	☐	☐	☒	☐
Analýza	☐	☐	☐	☒
Vývojová dokumentace	☐	☐	☐	☒
Uživatelská dokumentace	☐	☐	☒	☐
Komentář Více viz příloha posudku níže.				

Implementační část práce	lepší	OK	horší	nevyhovuje
Kvalita návrhu *... architektura, struktury a algoritmy, použité technologie*	☐	☐	☐	☒
Kvalita zpracování *... jmenné konvence, formátování, komentáře, testování*	☐	☒	☐	☐
Stabilita implementace	☐	☐	☐	☒
Komentář Více viz příloha posudku níže.				

Celkové hodnocení Neprospěl(a)
Práci navrhuji na zvláštní ocenění Ne

Datum 2. září 2022 **Podpis**

Příloha k posudku bakalářské práce:

Autor práce: Monika Bošániová
Název práce: Rubikova kocka

Oponent: Pavel Ježek, UK MFF KDSS

Formální úprava

Práce obsahuje obrovské množství překlepů (typicky několik na každé straně), a to nejen v samotných větách textu (které jsou někdy i různě nedokončené a nesmyslné), ale i ve velkých nadpisech práce.

Citace

V části 2.1.1 jsou **některé věty doslova zkopírované z článku Thistlethwaite's 52-move algorithm** (a doslova přeložené do slovenštiny), např. „To zodpovedá skutočnosti, že táto fáza fixuje orientáciu rohov a umiestňuje hrany strednej vrstvy.“, nebo „správne umiestnené hrany ľavej a pravej steny vo všetkých troch stredových vrstvách (prstencoch), rohy sú v tetrádach a platí pre ne párna permutácia.“ – ale věty nejsou v textu práce nijak označené jako doslovné citace (ani uvozovkami, ani kurzívou), **a článek není ani nijak citován** (pouze v seznamu literatury je generická citace, ale nikoliv u použitých vět). Navíc např. druhá věta ani není v kontextu práce jasná, protože o tetrádách jinde autorka nemluví a nevysvětluje je.

V aplikaci jsou použité obrázky, které jsou k dispozici sice zdarma, ale **vyžadují attribution** („Flaticon License: Free for personal and commercial purpose with attribution. Attribution is required. How to attribute?“), viz obrázek šipky z:

https://www.flaticon.com/free-icon/right-arrow_3248150?term=arrow%20symbol&page=1&position=9&page=1&position=9&related_id=3248150&origin=tag

nebo informační ikonky:

https://www.flaticon.com/free-icon/information_733107?_gl=1*1ptm6u5*test_ga*MTA2MDE3MTAzLjE2NjE5ODU1MTk.*test_ga_523JXC6VL7*MTY2MTk4NTUxOS4xLjEuMTY2MTk4NTgxMC42MC4wLjA.*fp_ga*MTA2MDE3MTAzLjE2NjE5ODU1MTk.*fp_ga_1ZY8468CQB*MTY2MTk4NTUxOS4xLjEuMTY2MTk4NTgxMC42MC4wLjA.&_ga=2.77238869.113399148.1661985519-106017103.1661985519

a další v práci použité obrázky. **O zdroji obrázků se ale autorka v práci nezmiňuje zcela vůbec, natož licencí požadovaným způsobem.**

Stejně tak není v práci označen žádný zdroj použité hudby, ale tu se mi dohledat nepodařilo – je tedy autorka práce i jejím autorem?

Jako další zásadní závadu vnímám, že celá reprezentace 3D modelu kostky, a jejího rozložení do sítě (čili jedna z hlavních částí aplikace), je vyrobena přesně dle tutoriálu na YouTube rozděleného do 6 videí:

https://www.youtube.com/watch?v=JN9vx0veZ-c

a další – kromě toho, že zvolený způsob reprezentace je velmi nevhodný (viz dále v části „Kvalita návrhu a zpracování“), tak hlavně **autorka tento tutoriál v práci zcela vůbec nijak nezmiňuje, ani necituje, a naopak se text „tváří“, jako kdyby s tímto návrhem autorka přišla sama**. Jsem si ale dost jistý, že přesně dle uvedeného rozsáhlého tutoriálu postupovala, jelikož:

1) výsledná implementace kódu a organizace scény v Unity je zcela stejná,

2) při zadání „unity rubik's cube“ nebo podobného hesla do Googlu, je daný tutoriál hned 1. odkaz vrácený z vyhledávání (a to pro různé uživatele),
3) v adresářích odevzdaného projektu a gitové repozitory bakalářské práce jsou screenshoty starších verzí aplikace, které přesně odpovídají konečné fázi tutoriálu (původně měla autorka i anglické popisky tlačítek dle tutoriálu, než je zaměnila za slovenské ve výsledné aplikaci); navíc i výsledná aplikace stále dodržuje relativní uspořádání částí na obrazovce stejně jako v tutoriálu,
4) autorkou nesmyslné první ovládní aplikace odpovídá ovládání, které navrhuje uvedený tutoriál.

Analýza zadání

Jelikož je zadání práce velmi stručné, a jelikož je práce zaměřená na konkrétní cíl zlepšení výuky skládání Rubikovy kostky, tak by pro takovouto práci mělo být zcela zásadní provést kvalitní analýzu zadání, neboli analýzu potřeb a požadavků potenciálních uživatelů a jasné stanovené přesnějších functional i non-functional requirements, které by odpovídalo uzavření kontraktu se zadavatelem. Stejně tak by součástí práce měla být alespoň nějaká základní rešerše podobných aplikací a rozbor jejich výhod a nevýhod a z toho opět vyvození závěrů, jaké vlastnosti a jak do aplikace implementovat. Nicméně nic z toho není v práci vůbec ani náznakem provedeno. Autorka tak průběžně zcela „vaří z vody“, a výsledkem jsou zcela nejasná a nevhodná rozhodnutí.

Již jen samotné ovládání otáčení kostkou jde zcela proti smyslu aplikace. Ovládá se klikáním na tlačítka, který odpovídají jednotlivým stranám kostky a směrům otáčení. Ověřil jsem, že začátečník (pro které je aplikace určena) je tímto ovládáním zcela zahlcen, a jelikož hned od začátku ho aplikace vše učí formálním zápisem algoritmů, tak uživatel zcela ztrácí kontakt s kostkou a vůbec ji nesleduje, a prostě jen „tupě“ kliká na tlačítka, která odpovídají krokům algoritmu, bez toho aby si budoval intuitivní představu o tom, co které tahy s kostkou dělají. To je ještě podpořeno tím, že si uživatel nemůže otočit kostku do libovolné pozice – resp. se mu z ní vždy vrátí do základní. To ale uživatele nutí skládat „zadní“ strany bez toho, aby se na ne díval, a opět se v tom, co se děje na kostce zcela ztrácí, ale opět ho to motivuje jen k bezmyšlenkovitému klikání na tlačítka. Autorka sice toto v práci zmiňuje, že je záměr, že pro rychlé skládání se nemá kostka otáčet, aby uživatel otáčením neztrácel čas – nicméně, zde je jasný rozpor s určením aplikace (opět chybějící analýzou), protože přeci cílem aplikace není naučit se rychle skládat, ale skládat nějak dle konkrétní začátečnické metody, a naopak představu, co tahy dělají na různých stranách kostky by si měl uživatel budovat. Navíc není pravda, že otáčení celou kostkou je vždy špatné – i nejlepší speedcubeři v některých konfiguracích kostku raději otočí, protože jim to umožní v jiném otočení provést jednodušší algoritmus, a tím je pak výsledné skládání stejně rychlejší (i se započítáním času otočení kostkou).

Autorka práce sice uvádí, že zkoušela nejprve ovládáním tažením ze středu strany (což mi přijde rovnou zřejmě málo intuitivní), a to že bylo pro uživatele neintuitivní (což ale nepodporuje žádným konkrétním experimentem, ani popisem počtu a zkušeností uživatelů), proto přešla na uvedený způsob. Nicméně, proč nepoužila ovládání tažením (swipe) ve směru hrany dané vrstvy, které je zcela nejběžnější v drtivé většině „kostkových“ aplikacích … ? – namátkou třeba v aplikacích Magic Cube, nebo CubeX, nebo Cube Rubik, nebo Fridrich CFOP. Zkusil jsem jednu z těchto aplikací předložit 8 učitelům informatiky a/nebo matematiky ve věku 30 až 60 let (což by mohlo rozumně „simulovat“ cílovou skupinu) s tím, že zcela bez jakékoliv nápovědy nebo dokumentace měli provést na kostce 3 tahy dle obrázku, a nikdo z nich neměl žádnou zkušenost s ovládáním nějaké „kostkové“ aplikace. A z toho:

1) 4 rovnou intuitivně zkusilo správný způsob swipe ovládání a bez dalšího zkoumání bez chyb provedli všechny 3 tahy,
2) 1 přišel na správné swipe ovládání po několika nesprávných pokusech do 15 sekund od začátku práce s aplikací, pak pochopil ovládání, a požadované 3 tahy již provedl rychle a bez chyb,
3) 3 nejprve zkoušeli provádět otáčení gestem 2 prsty, ale do 20 sekund přišli všichni na swipe pohyb, a požadované 3 tahy již provedli rychle a bez chyb.

Přestože tento rychlý experiment proběhl na mobilním telefonu, tak podle mě i v prostředí desktopové aplikace dostatečně podporuje hypotézu, že způsob ovládání používaný ostatními aplikacemi je zvolený dobře, a je opravdu pro uživatele opravdu intuitivní. V bakalářské práci by samozřejmě experiment měl být proveden daleko pečlivěji, nicméně je myslím zřejmé, že autorčino rozhodnutí stran ovládání není dobré.

Technická a didaktická analýza

V práci bohužel zcela chybí i jakákoliv technická forma analýzy, autorka vůbec nijak nevede diskusi o možnostech implementace a návrhu aplikace, pouze předkládá popis hotových řešení (a ani to často ne). Autorka nikdy nepředkládá žádné alternativy a nijak nehodnotí výhody a nevýhody zvolených řešení. Stejně tak autorka nijak neřeší didaktickou rovinu své práce a aplikace, nepředkládá žádné alternativy stran svého rozhodnutí, ale vše bez hlubšího vysvětlení prezentuje jako hotový fakt. Problémy jako jak vůbec připravit koncept tutoriálů, jak tutoriály reprezentovat, jaké datové struktury a reprezentace zvolit, jak naimplementovat tahy, atd. – celkově žádné klíčové mechanizmy v práci nejsou popsané ani na úrovni vývojové dokumentace (jak funguje v práci hotové řešení), viz dále, ale ani zmínkou nejsou diskutovány v absentující analýze – u žádného z těchto rozhodnutí se čtenář nedozví odpověď na otázku „proč takhle, a proč ne jinak“.

V části 1.2.2 uvádí, že se rozhodla nepoužít běžnou notaci apostrofu pro inverzní tah, a místo toho používá spíše okrajově používanou notaci malým i. Oficiální materiály WCA (World Cube Association) ale používají apostrofovou notaci, stejně tak je nejběžnější v materiálech pro začátečníky (na které aplikace cílí). Jaký je k této volbě důvod, ale autorka nijak neuvádí a nevysvětluje.

V části 2.1.1 autorka vcelku detailně popisuje Thistlethwaite algoritmus, který se ale pak rozhodne použít. Proč ho ale popisuje, a proč ho nepoužije ale není jasné. Ani v popisu nemá žádnou argumentaci v kontextu práce.

V části 2.1.2 pak zmiňuje algoritmus Kociemba – ten ale nijak důkladně nevysvětluje, a zcela bez kontextu jen zmiňuje, že ho v práci použije. Nicméně jeho použití v práci nedává smysl, protože to je algoritmus pro rychlé vyřešení kostky počítačem, ale negeneruje posloupnost tahů, ze které by se mohl lidský hráč nějak poučit – ale přitom přesně tak je v práci použit, tj. pro zobrazení posloupností tahů uživateli, jak složit kostku. To je ale i pokročilému uživateli zcela k ničemu, natož začátečníkovi, když to je mimo kontext zvolené metody.

Zdá se, že algoritmus Kociemba autorka v práci použila jen proto, že nalezla hotovou knihovnu pro Unity, která ho implementuje (a tu v práci použila).

V části 2.1.3 autorka prezentuje právě 1 začátečnickou metodu skládání Rubikovy kostky a to metodu Rubik’s 2020, a pouze zmiňuje, že existují i další. Bez jakéhokoliv rozboru ale tuto 1 metodu zvolí a v práci implementuje. Proč ale nezvolit třeba CFOP metodu (je příliš náročná?), nebo Roux nebo Petrus metody (ty přeci podporují intuici, a to by mělo začátečníkům pomoci), nebo proč nezačít se „sexy metodou“, která je přeci pro začátečníka nejpřístupnější, protože vyžaduje zásadně méně učení než Rubik’s 2020 (jelikož je založena jen na opakování tahu „sexy move“). Ale přeci existuje i spousta dalších metod, tak proč se nějakým v práci nevěnuje?

Dále v části 2.1.3 autorka metodu Rubik’s 2020 detailně popisuje – opět tak ale činí bez jakéhokoliv náznaku analýzy – proč tam popis je? Autorka nijak neřeší, jak v práci metodu pojmout, jak naimplementovat. Autorka navíc vychází z letáku od Rubik’s (v práci citovaný), ten ale do textu práce nepřepisuje přesně. Některé části mají vynechané podkroky, nebo nějaké případy, nebo jsou prostě špatně – tedy z textu práce nelze metodu správně pochopit – mimo jiné např.:

> Bod 3 – autorka uvádí „předpokládejme, že se nachází pod místem“ – to ale nelze předpokládat, do té pozice je třeba kostku dostat – viz letáček Rubik’s strana 18 dole.
>
> Bod 5 – „otáčíme celým hlavolamem“ – zde ale autorka pomíjí, že otáčením celým hlavolamem v aplikaci zakázala. Navíc není uvedené, že algoritmus musíme provádět opakovaně, a také není uvedené, že se nesmíme koukat na rohy, že mohou být i žluté (obojí ale uvedeno v originálním letáčku).

Bod 6 – v práci má autorka uvedeno „rotaci hranových“ – dle letáčku a správně má být ale „rotaci rohových“. Dříve v textu již píše, že levá strana obrázku je vždy přední a pravá je pravá, a zároveň v textu píše, že „pro tento případ předpokládejme, že oranžová je přední“ – ale odkazuje se tím na obrázek 2.7, kde je přední zelená a oranžová je pravá. V originálním letáčku správně.

Bod 7 – nezmiňuje, že se musí točit horní vrstvou, než budou 2 rohy na správné stěně (v originálním letáčku ale uvedeno)

Bod 8 – nepopisuje vůbec, že může nastat případ, že jsou všechny 4 strany nekompletní (v originálním letáčku ale uvedeno).

Celá tato 5-stránková část je tedy v práci zbytečná – obsahuje tolik chyb, že není pro nic využitelná, a navíc neřeší nijak zasazeni do kontextu implementace bakalářské práce.

Navíc je velmi problematické, že se autorka kvůli snazší implementaci rozhodla upravit metodu, ale své úpravy metody již nijak detailně nepopisuje. Navíc tím uživatelé nemohou použít standardní letáček a další materiály k metodě Rubik's 2020. A dále autorčina úprava směřuje na to, že některé případy musí řešit navíc, a tedy se uživatelé musí učit více algoritmů než u původní metody – a to opět jde proti konceptu aplikace pro začátečníky (kde chceme počet učených algoritmů minimalizovat – ale to autorka v práci neřeší).

V textu jsou navíc i chyby v popisech tahů (které jsou opravdu velmi špatně odhalitelné, zvlášť pro začátečníka), např. v 1.2.2 je uvedeno „Napríklad rotáciu stredného riadka v smere hodinových ručičiek, môžeme substituovať súbežnou rotáciou Ui a D.“ – to je ale špatně – uvedený případ odpovídá posloupnosti U a Di v autorčině notaci.

Vývojová dokumentace

V práci prakticky **chybí vývojová dokumentace**. V části 4.1 je sice uvedený obrázek několika tříd řešiče, ale chybí k tomu libovolný další text, který by fungování tříd, a jejich návrh a API nějak vysvětlil. Pouze je zde nedokončená věta – doslovně citováno „Ďalším pozitívnym aspektom je.“, a to je vše. Stejně tak kompletně chybí dokumentace samotné implementační části v Unity – detailní popis scény, a vztahů jednotlivých objektů, připojené skripty a jejich funkce a vztahy.

Soubor index.html z vygenerované části vývojové dokumentace obsahuje jen placeholder z nějaké šablony, ale vůbec neobsahuje nic souvisejícího s prací a ani z něj nevedou žádné odkazy na nějakou dokumentaci k práci – tedy tato část dokumentace je pro programátora nedostupná. Nicméně jelikož jsou dokumentační komentáře v kódu jen velmi sporadicky, tak by stejně asi pro programátora nebyla vygenerovaná část dokumentace přínosná.

Uživatelská dokumentace

Text práce sice formálně uživatelskou dokumentaci obsahuje, ale většinově popisuje jen zřejmé věci, které by u dané aplikace nebyly třeba (navíc pokročilejší části vyžadují přečtení hlavního textu práce – to ale pro uživatelskou dokumentaci nedává smysl – ta by byla vydána spolu s produktem) – jelikož je navíc celá aplikace koncipována jako tutoriál, tak se domnívám, že nikdo nebude chtít dobrovolně ještě další dokumentaci číst, ale samotné tutoriály by měly být postavené tak, aby byly „samonosné“. Procházení tutoriálů je ale velmi zmatečné, a uživatel často neví, zda se zobrazovaný text týká toho, co právě na obrazovce viděl nebo toho, co ho teprve čeká. Proč např. hned v levelu 1 je nutné před 1. tahem zmáčknout šipku dále, ale před 2. tahem R2 již ne (přitom popis i pozice vypadají na obrazovce v obou případech stejně).

Z uživatelského hlediska je velmi problematické, že se nelze vrátit o krok zpět a znovu dopředu. Lze pouze celý tutoriál ukončit, a znovu ho začít od začátku – čímž ale uživatel přijde o stav kostky. Přitom již v levelu 0 běžně uživatel zapomene ovládání (které mimochodem není vysvětlené intuitivně – význam zkratek pro jednotlivé tahy je jen v textu práce a to ještě nikoliv v uživatelské dokumentaci), ale již si ho nemůže připomenout vrácením do předchozího kroku.

Stejně tak když uživatel v pokročilejších levelech zapomene část postupu, nebo mu není jasný v dané fázi skládání, tak se nemůže k vysvětlení na začátku levelu vrátit.

Navíc stejně jako v textu práce, tak i v samotné aplikaci je velké množství překlepů a nedomyšleností, a popisný text v neříká vždy zcela pravdu. Např. už jen v levelu 1 je napsáno „cíl květinka na horní straně“, ale u „tvoj ciel“ je zobrazená naopak dolní strana a bílý kříž. Dále i zbývající text je zcela nepochopitelný, k pochopení metody je třeba použít původní leták Rubik's – až na to, že v něm uvedený postup je nakonec nepoužitelný, protože autorka v práci upravila sekvenci tahů. Zmatení uživatele je pak dokonáno, když v kroku „zelenobílá“ je uvedeno, že máme provést Ri U Fi Ui, ale ve stejném okamžiku je v nápovědě dole uvedeno, že řešením je provést tahy Li U Bi Ui.
Navíc i celý postup v levelu 1 se liší od postupu v odkazovaném letáčku (jehož metodu práce „implementuje“) – kde v letáčku se do horní vrstvy umisťují dílky bez ohledu na barvu středu, ale v aplikaci se nejprve propojí se středem, ale až pak dávají nahoru – tím ale vznikají zmatečné a neintuitivní situace, že hrana již umístěná na správné pozici v dolní stěně se dle aplikace přenáší do horní stěny, ale později se opět přenáší dolů.

Nebo v levelu 0 se zobrazí text „správně“ i při špatném složení, a pak se bez přestávky pouze rovnou doanimuje kostka pro další pokus. A opět se není ani možné vrátit se zpět, aby uživatel pochopil, co vlastně udělal špatně.

Podobné problémy se tykají i zbytku aplikace a zbylých fungujících levelů (ale většina funkční není vůbec, viz dále). Jelikož k odkazovanému letáčku existuje sada videí (od autorů Rubik's), která metodu vysvětluje daleko lépe didakticky, a jasným rozborem případů, tak se nakonec tato videa zdají daleko přínosnější pro naučení se metody než autorčina aplikace, která je méně univerzální (nevysvětluje všechny případy).

Kvalita návrhu a zpracování

Autorka sice dokázala vyprodukovat obrovské množství kódu, který je ale na tak tragické úrovni, že nedosahuje ani úrovně absolventa 1. ročníku, natož absolventa celého bakalářského studia „Informatiky“ a potažmo zaměření „Programování a softwarové systémy“ na MFF UK. Většina kódu je copy&paste rozkopírována, kde se jeden řádek opakuje několikrát, nebo několik desítek krát, s ručně opraveným jedním nebo několika znaky. Celý kód je prošpikován nekonečnými „switch-like“ kaskádami ifů, a autorka v kódu skoro vše reprezentuje jako textové řetězce, které při volání funkcí neustále generuje a naopak znovu parsuje. Celkově kód tímto způsobem postrádá jakoukoliv univerzalitu nebo zobecnění, absolutně pomíjí jakákoliv alespoň elementární pravidla softwarového inženýrství pro alespoň základní přehlednost a udržovatelnost kódu. Příklad malé části více než 200 řádkové funkce:

```
public override TutorialMove SolveOneMove()
        {
            if (!centersAreCorrect())
                throw new InvalidCubeException("centersAreNotCentered");

            if (!stage1done())
            {
                string stage = "1";
                // right
                if (!cubeMatch("012", "D", "301", "R"))
                {
                    if (cubeMatch("010", "DR", "101", "DR")) return new TutorialMove(stage, "L2", "Modro biela - Otočiť nadol");
                    if (cubeMatch("021", "DR", "201", "DR")) return new TutorialMove(stage, "F2", "Modro biela - Otočiť nadol");
                    if (cubeMatch("001", "DR", "401", "DR")) return new TutorialMove(stage, "B2", "Modro biela - Otočiť nadol");
                    if (cubeMatch("212", "DR", "310", "DR")) return new TutorialMove(stage, "Ri", "Modro biela - Otočiť nadol");
                    if (cubeMatch("312", "DR", "410", "DR")) return new TutorialMove(stage, "R", "Modro biela - Otočiť nadol");
                    if (cubeMatch("112", "DR", "210", "DR")) return new TutorialMove(stage, "L", "Modro biela - Otočiť nadol");
                    if (cubeMatch("412", "DR", "110", "DR")) return new TutorialMove(stage, "Li", "Modro biela - Otočiť nadol");
                    if (cubeMatch("121", "DR", "510", "DR")) return new TutorialMove(stage, "D2", "Modro biela - Otočit na správnu stranu v
spodnej časti");
                    if (cubeMatch("221", "DR", "501", "DR")) return new TutorialMove(stage, "D", "Modro biela - Otočit na správnu stranu v
spodnej časti");
                    if (cubeMatch("421", "DR", "521", "DR")) return new TutorialMove(stage, "Di", "Modro biela - Otočit na správnu stranu v
spodnej časti");
                    if (cubeMatch("321", "DR", "512", "DR")) return new TutorialMove(stage, "R2", "Modro biela - Otočiť nahor");
                    if (cubeMatch("012", "R", "301", "D")) return new TutorialMove(stage, "Ri U Fi Ui", "Modro biela - Otočit diel na
mieste pohybmi Ri U Fi Ui");
                    throw new InvalidCubeException("missing configuration");
                }
                // front
                if (!cubeMatch("021", "D", "201", "F"))
```

```
            {
                if (cubeMatch("010", "DF", "101", "DF")) return new TutorialMove(stage, "L2", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("012", "DF", "301", "DF")) return new TutorialMove(stage, "R2", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("001", "DF", "401", "DF")) return new TutorialMove(stage, "B2", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("212", "DF", "310", "DF")) return new TutorialMove(stage, "F", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("312", "DF", "410", "DF")) return new TutorialMove(stage, "Bi", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("112", "DF", "210", "DF")) return new TutorialMove(stage, "Fi", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("412", "DF", "110", "DF")) return new TutorialMove(stage, "B", "Oranžovo biela - Otočit nadol");
                if (cubeMatch("121", "DF", "510", "DF")) return new TutorialMove(stage, "D", "Oranžovo biela - Otočit na správnu stranu
v spodnej časti");
                if (cubeMatch("321", "DF", "512", "DF")) return new TutorialMove(stage, "Di", "Oranžovo biela - Otočit na správnu
stranu v spodnej časti");
                if (cubeMatch("421", "DF", "521", "DF")) return new TutorialMove(stage, "D2", "Oranžovo biela - Otočit na správnu
stranu v spodnej časti");
                if (cubeMatch("221", "DF", "501", "DF")) return new TutorialMove(stage, "F2", "Oranžovo biela - Otočiť nahor");
                if (cubeMatch("021", "F", "201", "D")) return new TutorialMove(stage, "Fi U Li Ui", "Oranžovo biela - Otočit diel na
mieste pohybmi Ri U Fi Ui");
                throw new InvalidCubeException("missing configuration");
            }
```

nebo malá část jiné funkce:

```
public void DoMoveNow(string move)
        {
            switch (move)
            {
                case "R":
                    side = Colors.Blue;
                    angle = 90;
                    break;
                case "Ri":
                    side = Colors.Blue;
                    angle = -90;
                    break;
                case "R2":
                    side = Colors.Blue;
                    angle = -180;
                    break;
                case "L":
                    side = Colors.Green;
                    angle = 90;
                    break;
                case "Li":
                    side = Colors.Green;
                    angle = -90;
                    break;
                case "L2":
                    side = Colors.Green;
                    angle = -180;
                    break;
                case "F":
                    side = Colors.Orange;
                    angle = -90;
                    break;
                case "Fi":
                    side = Colors.Orange;
                    angle = 90;
                    break;
                case "F2":
                    side = Colors.Orange;
                    angle = -180;
                    break;
```

nebo malá část další funkce:

```
private void DoOppositeMove(string moveToMakeOpposite)
        {
            switch (moveToMakeOpposite)
            {
                case "F":
                    DoMove("Fi");
                    return;
                case "B":
                    DoMove("Bi");
                    return;
                case "R":
                    DoMove("Ri");
                    return;
                case "L":
                    DoMove("Li");
                    return;
                case "U":
                    DoMove("Ui");
```

```
                return;
            case "D":
                DoMove("Di");
                return;
            case "F2":
                DoMove("F2");
                return;
            case "B2":
                DoMove("B2");
                return;
```

nebo část jiné funkce:

```
internal void MakeCubeStateFromeString(string input)
        {
            if (input.Length != 54)
                throw new Exception("incorect layhout length");

            foreach (Transform cube in transform)
            {
                foreach (Transform plane in cube)
                {
                    if (!plane.gameObject.activeSelf)
                        continue;

                    int pieceIndex = -1;
                    Vector3 pos = plane.position;

                    if (Math.Abs(pos.y - 1.5) < .01) // up
                    {
                        pieceIndex = 9 * 0;
                        if (Vector3.Distance(cube.position, new Vector3(1, 1, 1)) < 0.01) pieceIndex += 0;
                        else if (Vector3.Distance(cube.position, new Vector3(1, 1, 0)) < 0.01) pieceIndex += 1;
                        else if (Vector3.Distance(cube.position, new Vector3(1, 1, -1)) < 0.01) pieceIndex += 2;
                        else if (Vector3.Distance(cube.position, new Vector3(0, 1, 1)) < 0.01) pieceIndex += 3;
                        else if (Vector3.Distance(cube.position, new Vector3(0, 1, 0)) < 0.01) pieceIndex += 4;
                        else if (Vector3.Distance(cube.position, new Vector3(0, 1, -1)) < 0.01) pieceIndex += 5;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, 1)) < 0.01) pieceIndex += 6;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, 0)) < 0.01) pieceIndex += 7;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, -1)) < 0.01) pieceIndex += 8;
                    }
                    else if (Math.Abs(pos.z - 1.5) < .01) // left
                    {
                        pieceIndex = 9 * 1;
                        if (Vector3.Distance(cube.position, new Vector3(1, 1, 1)) < 0.01) pieceIndex += 0;
                        else if (Vector3.Distance(cube.position, new Vector3(0, 1, 1)) < 0.01) pieceIndex += 1;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, 1)) < 0.01) pieceIndex += 2;
                        else if (Vector3.Distance(cube.position, new Vector3(1, 0, 1)) < 0.01) pieceIndex += 3;
                        else if (Vector3.Distance(cube.position, new Vector3(0, 0, 1)) < 0.01) pieceIndex += 4;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 0, 1)) < 0.01) pieceIndex += 5;
                        else if (Vector3.Distance(cube.position, new Vector3(1, -1, 1)) < 0.01) pieceIndex += 6;
                        else if (Vector3.Distance(cube.position, new Vector3(0, -1, 1)) < 0.01) pieceIndex += 7;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, -1, 1)) < 0.01) pieceIndex += 8;
                    }
                    else if (Math.Abs(pos.x - -1.5) < .01) // front
                    {
                        pieceIndex = 9 * 2;
                        if (Vector3.Distance(cube.position, new Vector3(-1, 1, 1)) < 0.01) pieceIndex += 0;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, 0)) < 0.01) pieceIndex += 1;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 1, -1)) < 0.01) pieceIndex += 2;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 0, 1)) < 0.01) pieceIndex += 3;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 0, 0)) < 0.01) pieceIndex += 4;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, 0, -1)) < 0.01) pieceIndex += 5;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, -1, 1)) < 0.01) pieceIndex += 6;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, -1, 0)) < 0.01) pieceIndex += 7;
                        else if (Vector3.Distance(cube.position, new Vector3(-1, -1, -1)) < 0.01) pieceIndex += 8;
                    }
```

a mnoho a mnoho dalších.

Navíc je tímto způsobem implementace celá aplikace tak velice úzce svázána právě s kostkou 3x3, že dle mého názoru ani není možné doplnit implementaci jakéhokoliv jiného hlavolamu, jak požaduje zadání práce. I implementace podobných kostek 2x2 nebo 4x4 by vyžadovala kompletní přepracování drtivé většiny kódu, který se v aplikaci nachází.

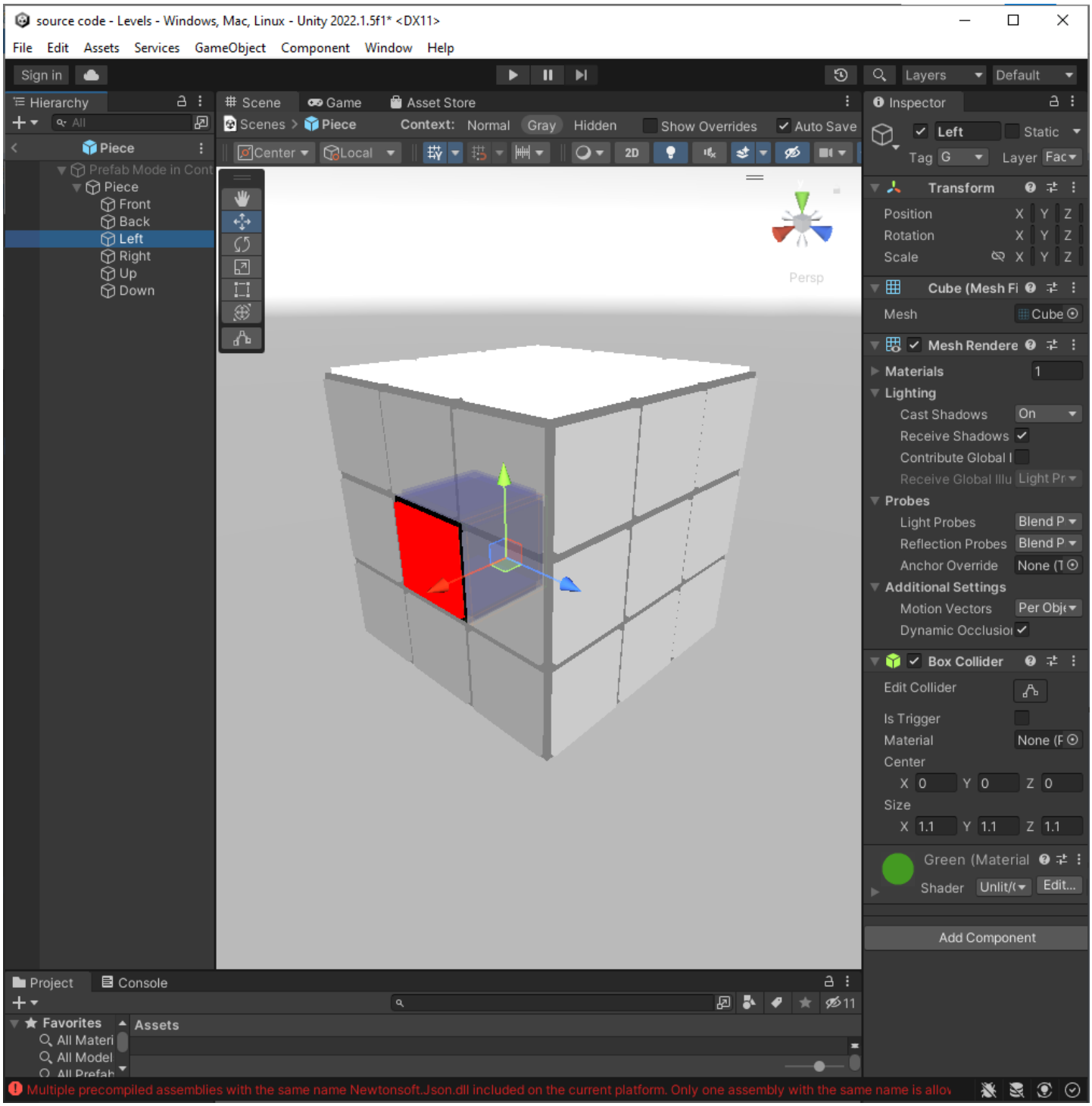
Dalším problémem je, že hlavní informace o rozložení kostky je daná nastavením barev materiálu jednotlivých plošek každého dílku ve 3D modelu ve scéně, a nikoliv nějakou rozumnou interní reprezentací kostky. Tedy 3D model kostky není jen vizualizací jejího stavu, ale samotným nositelem stavu. Tedy přechod na jiný způsob vizualizace by byl nesmyslně komplexní, a navíc v různých částech algoritmu musí autorka neustále provádět neustálé drahé zjišťování

materiálu každé části, a i se komplikuji další funkce programu – např. musí autorka uměle zanášet tag „Centre“, aby si omylem nepřebarvila materiál středových dílků při uživatelském editování rozložení kostky.
Navíc ze jména materiálu neustále znovu vyparsovává různé informace a to opět vede k velice neuniverzálním metodám jako (kde je navíc opět vše copy&paste rozkopírované, závislé na textových řetězcích, apod.):

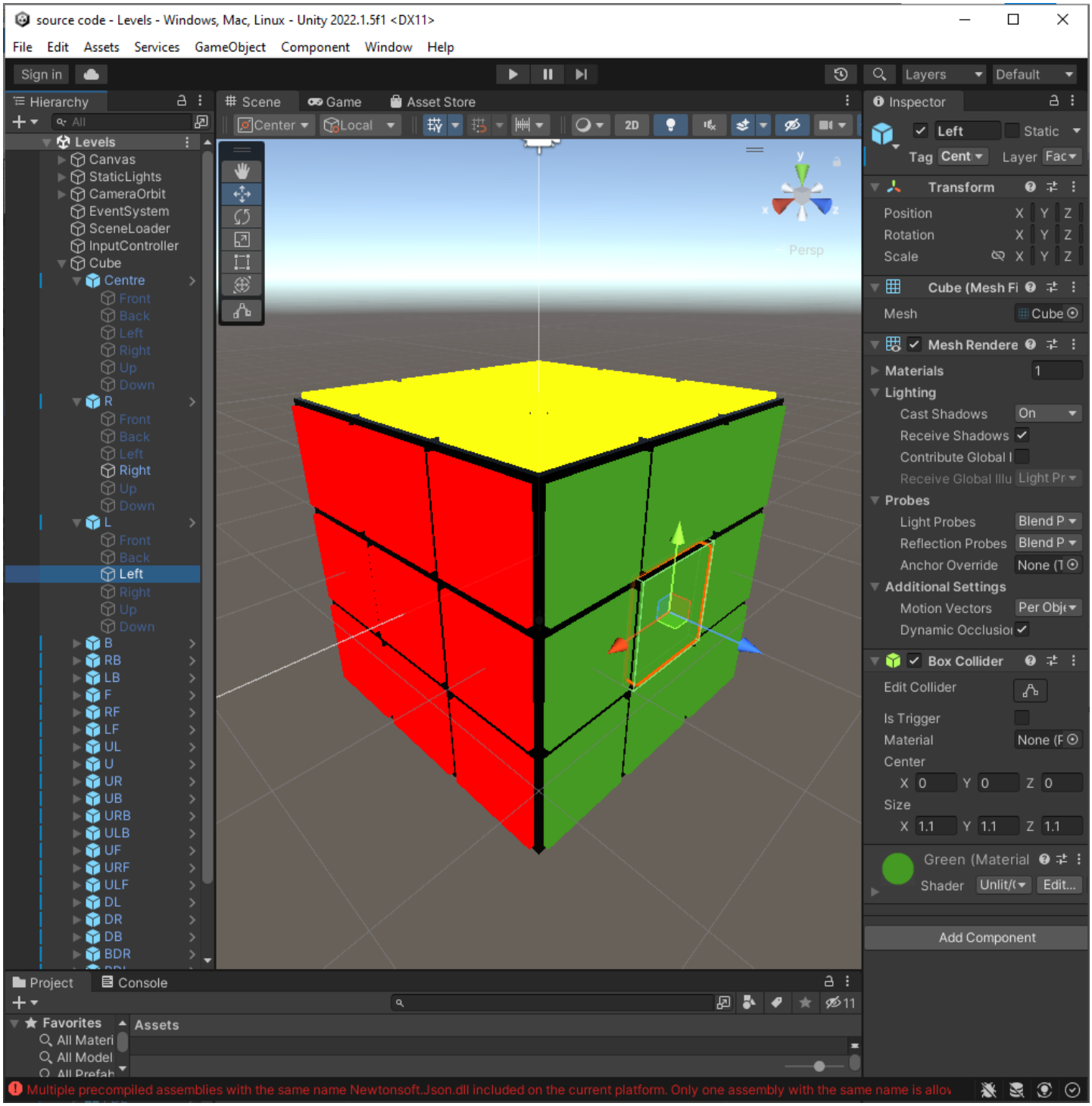
```
public void UnHideSide(string sides)
{
    string unselected = " - Unselected";
    if (sides.Contains("F") && Front.activeSelf &&
        Front.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Front.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Front.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
    if (sides.Contains("B") && Back.activeSelf &&
        Back.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Back.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Back.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
    if (sides.Contains("L") && Left.activeSelf &&
        Left.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Left.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Left.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
    if (sides.Contains("R") && Right.activeSelf &&
        Right.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Right.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Right.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
    if (sides.Contains("U") && Up.activeSelf &&
        Up.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Up.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Up.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
    if (sides.Contains("D") && Down.activeSelf &&
        Down.GetComponent<MeshRenderer>().sharedMaterial.name.Contains(unselected))
        Down.GetComponent<MeshRenderer>().sharedMaterial =
            materials[Down.GetComponent<MeshRenderer>().sharedMaterial.name.Split(' ')[0]];
}
```

Pokračování na další straně

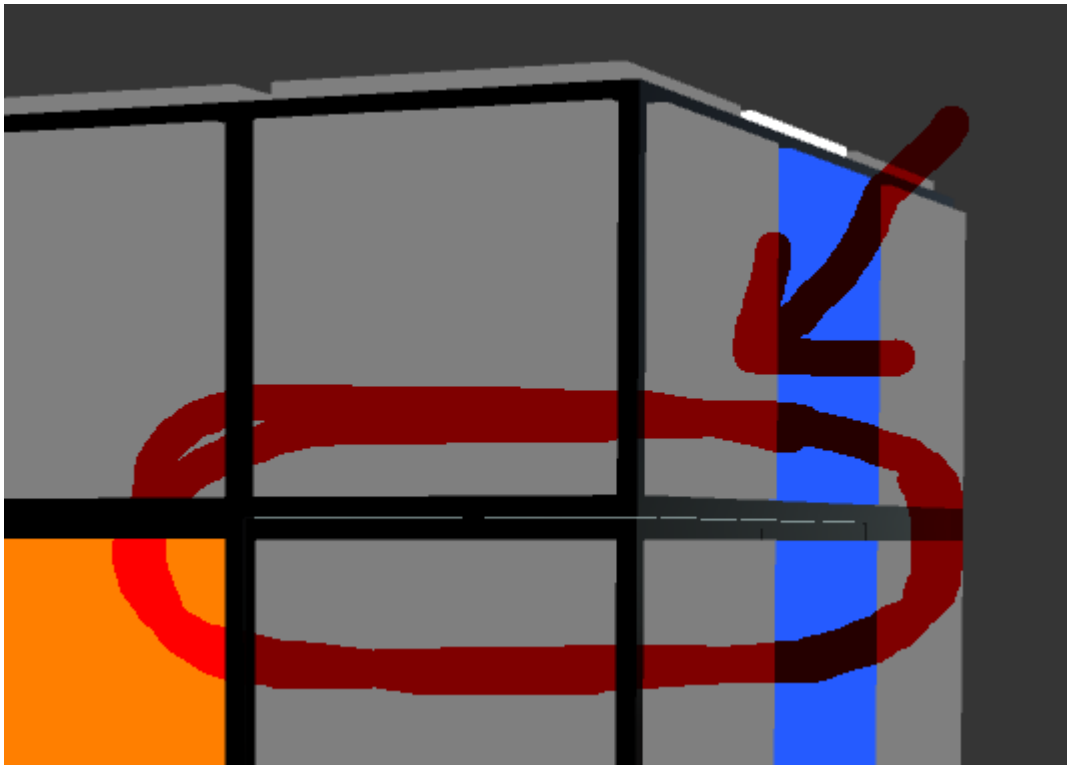
Navíc i samotná reprezentace modelu kostky ve scéně v Unity je nesmyslně komplikovaná, kde každý dílek kostky je reprezentovaný jako ručně vyrobený prefab se všemi 6 stranami různých barev – viz příklad jedné středové kostičky v editoru Unity, kde je vidět přední červená strana, a je zvolena levá strana, který je vidět (vpravo dole), že je plnohodnotný 3D model se zeleným materiálem:



Takovýto prefab je potom ručně 27-krát rozkopírován do scény, kde jsou u každé z těchto 27 kostiček ručně autorkou zobrazené nebo schované ty správné ze 6 stran kostičky, které mají být vidět – viz obsah scény v editoru vpravo, a objekt pro každou kostičku (ručně pojmenovaný dle její počáteční pozice na kostce), a světleji zvýrazněné (povolené zobrazení) vs. tmavěji zobrazené (zakázané zobrazení) podobjekty – pro přehlednost jsem „rozbalil“ jen několik prvních kostiček ve scéně:



Navíc kromě toho, že musela být každá taková kostička ručně pojmenována a oeditována, tak musela být i precizně umístněná – což se asi ne vždy povedlo, protože hodnoty pozice některých nejsou ve stejném vzoru jako ostatní, a i v samotné aplikaci je v určitých polohách kostky vidět nepřesné zarovnání některých kostiček:

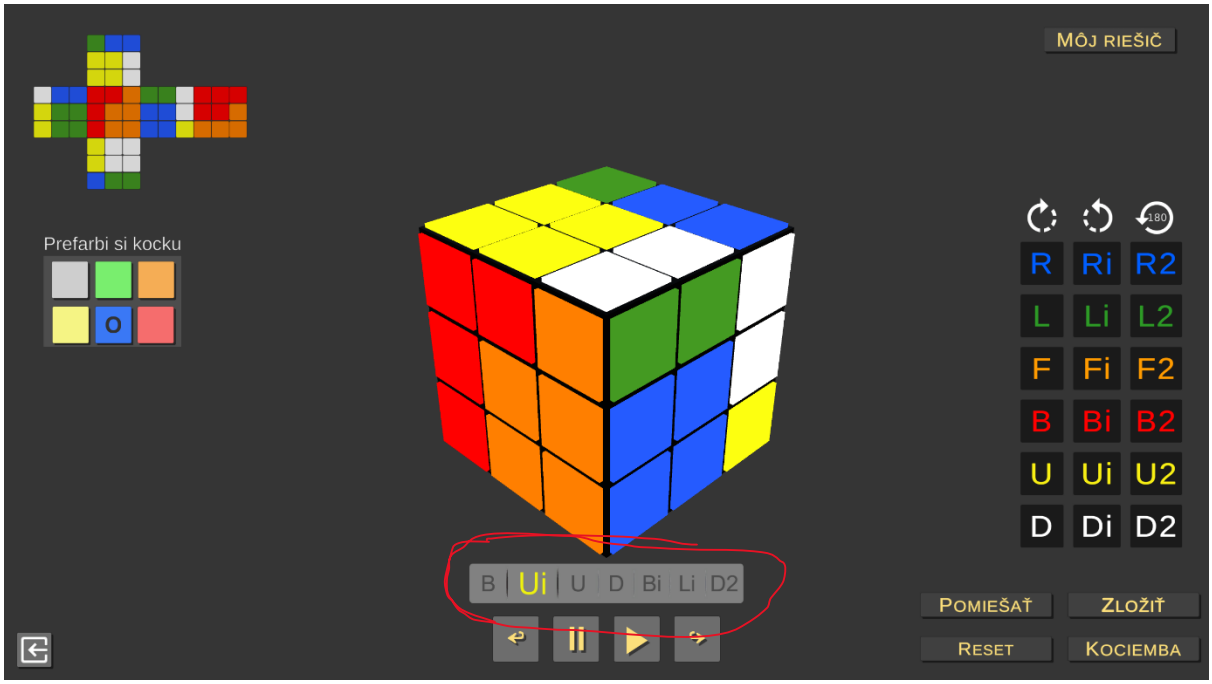


Autorka tedy nijak nevyužívá možnost generování 3D modelu kostky zautomatizovat – prostředí Unity k tomu přitom vyložené vybízí, jelikož v použitém jazyce C# může i jednoduše přidávat rozšíření samotného editoru, pokud by generování z nějakého důvodu nechtěla dělat při spouštění aplikace.

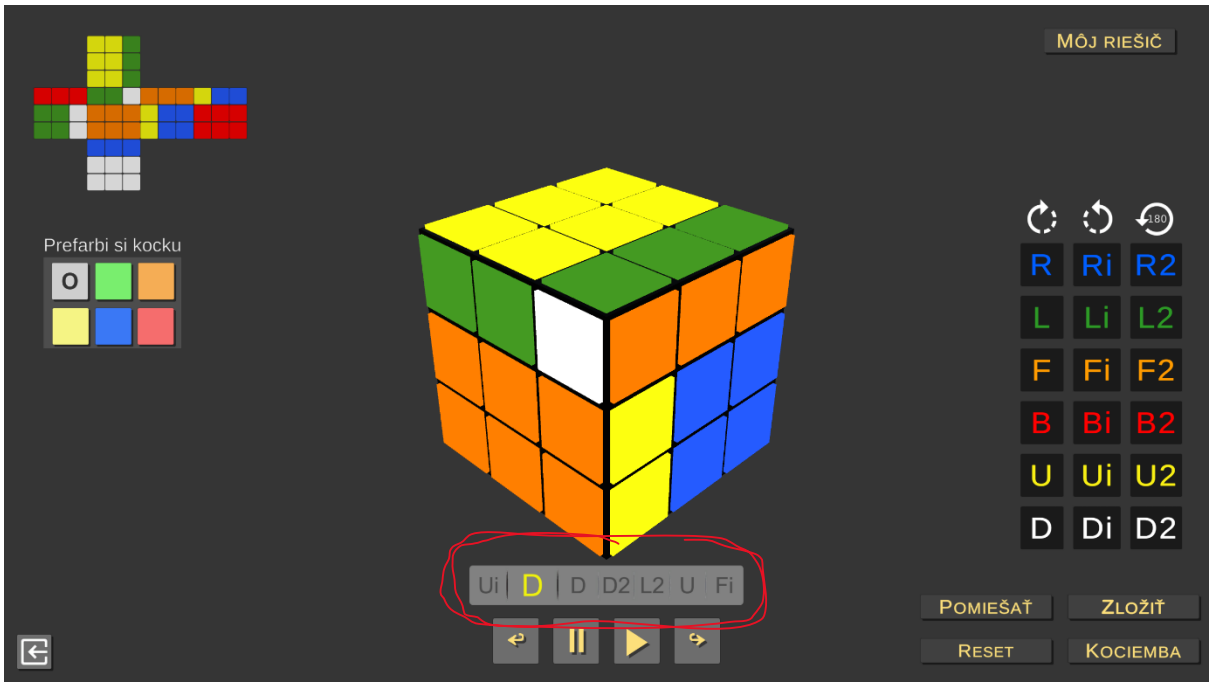
To, že je reprezentace kostky vyrobena dle (necitovaného! tutoriálu, viz výše) nemůže přeci autorka v práci zcela nekriticky využít, ale minimálně v textu musí přijít s nějakou analýzou vhodnosti daného řešení (což v textu zcela schází), ale ještě lépe by měla být schopna přijít s nějakým vlastním rozumnějším řešením (i když třeba v částech inspirovaných daným tutoriálem). Absolvent MFF UK by měl být schopen rozpoznat, který „náhodný návod“ na webu dává smysl, a který vytvořil někdo očividně bez elementární znalosti rozumného návrhu a „správného“ použití enginu Unity.

Dále je zvláštní, že autorka má vytvořený prefab pro slider na hlasitost hudby, který má ale v práci použitý pouze 1 – tedy výhoda prefabu jakožto šablony zde zcela nemá smysl. Ale naopak asi 15 tlačítek menu apod. která jsou zcela totožná, tak prefab nemají, a jsou ručně rozkopírována ve scéně.

Tlačítko „Pomiešať“ negeneruje rozumné scrambly, protože generuje i zcela zbytečné tahy jako Ui U:



nebo D D D2:



Jakým způsobem ale správně generovat scramble ale autorka v práci vůbec neřeší, vůbec ani nezmiňuje, že jsou 2 koncepčně různé běžně používané způsoby „*random sequence*“ (který lze rozumně použít ale s vhodným postprocessingem slučování tahů, nebo zákazem otáčet stejnou stranou dvakrát nebo 2 paralelními stranami třikrát za sebou), vs. „*random state*“ generátorů (který se pak používá např. na WCA soutěžích).

Adresář projektu je bez rozumné organizace. V samotném adresáři projektu Unity má autorka uložené i TeXové zdrojáky textu práce, různé dočasné soubory, vygenerovanou dokumentaci, v adresáři Prefabs je kromě pár prefabů i množství nesouvisejících souborů, apod.

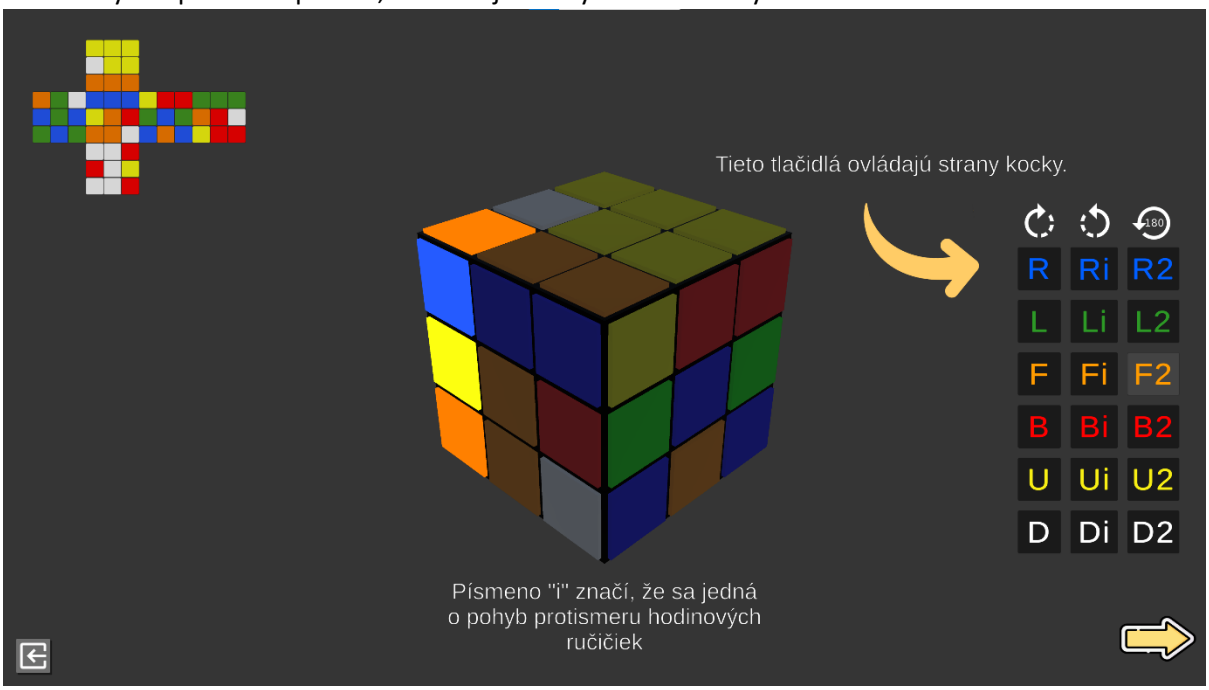
Přestože zvolený návrh a postupy implementace celé práce jsou na bakalářskou práci zcela nedostačující, tak základní věci jako **pojmenovávání tříd, metod, datovým položek, atd. plus formátování kódu jsou zcela v pořádku, a očekávané kvalitě odpovídají.**

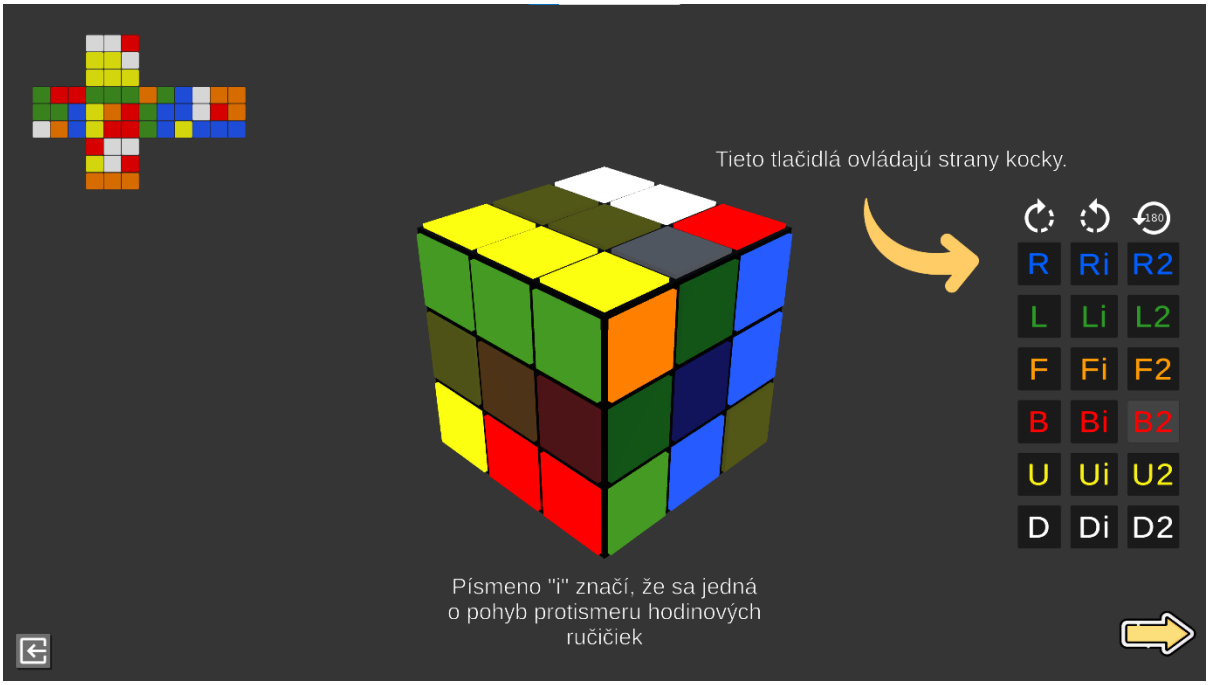
Stabilita implementace

Zcela zásadním problémem je, že v aplikaci jsou funkční pouze level 0, a 1, a pět nečíslovaných levelů – číslovaný **level 2 až level 8 vůbec nefungují** – vše otestování na 2 různých počítačích. U levelu 2 se přehraje animace algoritmu, ale již není možné kostku řešit, jelikož se po stisku šipku „dále“ nezobrazí ovládací tlačítka, a tedy ani se dostat k dalším levelům (které zůstanou zamknuté). Po ručním odemčení všech levelů (dle návodu v práci) ale podobně jako level 2 nefungují ani žádné další levely, jen jiným způsobem – opět se přehraje animace algoritmu, ale rovnou se zobrazí „Gratulujeme“ a uživatel opět nemá možnost kostku skládat (i když je současně zobrazený text „A teraz ty“) – dle logu Unity dochází v aplikaci k různým neošetřeným vyjímkám, např.: „InvalidCubeException: Invalid Cube: *missing configuration*“ v levelu 2, nebo „InvalidCubeException: Invalid Cube: *Cycle detected*“ v levelu 5, nebo „ArgumentOutOfRangeException: Index was out of range. Must be non-negative and less than the size of the collection.“, a další.

V aplikaci jsou ale i různé další chyby a nedodělky, např.:

1) Pokud uživatel skládá kostku příliš rychle, a zvolí další tah před dokončením animace předchozího tahu, tak se sice druhý tah provede správně, ale rozbije se zvýraznění kostky:





2) Level „šachovnice“ – „přehrávací tlačítka“ jsou zobrazená i když nic nedělají (před přehráváním, i při vlastním skládání) – v některých jiných levelech to ale funguje správně, a jsou zobrazená jen když mají.
3) V některých levelech zůstane zobrazený text „Zkus to znova“ i po úspěšném dokončení a současném zobrazení textu „Správně“.
4) Na tlačítko „Moj rešič“ lze kdykoliv klikat, ale nic nedělá (pouze v programu vznikne neošetřená vyjímka). Čekal bych buď zakázání tlačítka nebo upozornění uživatele, že má někam nakopírovat vlastní řešič.

2. září 2022 Pavel Ježek