

Střední průmyslová škola Třebíč

Studijní obor: Informační technologie

Třída: ITA4

Školní rok: 2023/2024 Lukáš Kurtin

Maturitní práce

Rubikova kostka

Profilová část maturitní zkoušky

Zadání práce

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

ABSTRAKT

Projekt je simulační prostředí Rubikovy kostky. Obsahuje základní funkce, jako jsou otáčení stran a provádění algoritmů podle celosvětové notace tahů. Kostka je vykreslena ve třídimenzionálním prostoru pomocí projekčních matic a lze ji otáčet okolo vertikální osy. Aplikace poskytuje i základní vysvětlení notace a zacházení s programem.

KLÍČOVÁ SLOVA

Maturitní práce, Rubikova kostka, C#, Windows Forms, .NET

ABSTRACT

KEYWORDS

Graduation thesis, Rubik‘s cube, C#, Windows Forms, .NET

PODĚKOVÁNÍ

Děkuji Ing. Ladislavu Havlátovi za odborný pohled při procesu tváření projektu a paní Natálii Pistovčákové, která mi pomohla se správností dokumentace.

V Třebíči dne 27. března 2024 podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne 27. března 2024

podpis autora

Obsah

[Úvod 7](#_Toc162377436)

[1 Teoretická část 8](#_Toc162377437)

[1.1 Rubikova kostka 8](#_Toc162377438)

[1.2 Algoritmizace 8](#_Toc162377439)

[1.2.1 Metodika CFOP 8](#_Toc162377440)

[1.2.2 Značení tahů 9](#_Toc162377441)

[1.3 Programová struktura kostky 9](#_Toc162377442)

[1.3.1 Způsob 1 9](#_Toc162377443)

[1.3.2 Způsob 2 9](#_Toc162377444)

[1.4 Grafické vykreslení 10](#_Toc162377445)

[1.4.1 2D 10](#_Toc162377446)

[1.4.2 3D 10](#_Toc162377447)

[1.4.3 Ortodoxní projekce 10](#_Toc162377448)

[1.5 Použité technologie 11](#_Toc162377449)

[1.5.1 GitHub 11](#_Toc162377450)

[1.5.2 GitHub Desktop 11](#_Toc162377451)

[1.5.3 .NET 11](#_Toc162377452)

[1.5.4 Windows Forms .NET Framework 11](#_Toc162377453)

[1.5.5 C# 11](#_Toc162377454)

[2 Praktická část 12](#_Toc162377455)

[2.1 Třídy 3D prostoru 12](#_Toc162377456)

[2.1.1 Vector3 12](#_Toc162377457)

[2.1.2 Square 12](#_Toc162377458)

[2.1.3 Cube 13](#_Toc162377459)

[2.2 Struktura kostky 13](#_Toc162377460)

[2.2.1 Deklarace kostky 14](#_Toc162377461)

[2.3 Vykreslování 14](#_Toc162377462)

[2.3.1 Projekční matice 14](#_Toc162377463)

[2.3.2 Pořadí vykreslení 14](#_Toc162377464)

[2.4 Tahy 14](#_Toc162377465)

[2.4.1 Vizuální změna polohy 14](#_Toc162377466)

[2.4.2 Programový pohyb 14](#_Toc162377467)

[2.5 Automatické poskládání 15](#_Toc162377468)

[2.5.1 Vrácení všech tahů 15](#_Toc162377469)

[2.5.2 Algoritmická metoda CFOP 15](#_Toc162377470)

[Závěr 17](#_Toc162377471)

[Seznam použitých zdrojů 18](#_Toc162377472)

[Seznam použitých symbolů a zkratek 19](#_Toc162377473)

[Seznam obrázků 20](#_Toc162377474)

[Seznam příloh 21](#_Toc162377475)

Úvod

Jako moji maturitní ročníkovou práci jsem chtěl vypracovat projekt nad moje zkušenosti, u kterého bych strávil více času než na jakémkoliv jiném projektu, který jsem v průběhu školních let dělal. Ideální je proto projekt, který je náročný nejen z hlediska programátorského, ale i grafického.

Vybral jsem si proto simulaci Rubikovy kostky, které jsou mi již léta blízké a vždy mě fascinovala jejich funkčnost. Pro uskutečnění bude potřeba projít několik způsobů, jak naprogramovat základní strukturu a funkce kostky.

Graficky bude kostka zobrazena v krychlové síti a ve 3D prostoru s možností otáčení pohledu. K tomu bude zapotřebí nastudovat projekční matice, které umožňují překlad mezi 3D prostorem a 2D obrazovkou.

Jako vývojové prostředí použiji Microsoft Visual studio a jeho balíček Windows Forms .NET framework, ve kterém se programuje v jazyku C# a který ulehčuje vykreslování na obrazovku a zadávání uživatelského vstupu. Pro řízení projektu a správu verzí využiji platformu GitHub, více popsanou v teoretické části.

Konečný stav projektu si představuji jako intuitivní aplikaci, která bude obsahovat samotné funkce Rubikovy kostky, ale i teoretický návod na její složení pomocí nejpoužívanější metody CFOP (pojmenované podle čtyř kroků, které aplikuje). Také bych chtěl implementovat optimalizovaný algoritmus pro automatické složení kostky, popřípadě ho využít jako doplněk naučné stránky.

# Teoretická část

Rubikova kostka je považována za náročný hlavolam. Jakmile se však člověk snaží přijít na to, jak vlastně funguje její mechanismus, zjistí, že to není tak složité.

Kostek je několik variant: 2x2, 3x3, 4x4, 5x5 a mnohem více. Čísla určují, kolik jedna řada z rohu do rohu obsahuje kostiček. Tato práce se zabývá tou původní a nejznámější verzí: 3x3.

## Rubikova kostka

Velký počet lidí si o Rubikově kostce myslí, že tahy zaměňují barvy, což není plně pravda. Skutečně se totiž neotáčí s barvami, ale s jednotlivými kostičkami. Kostka jich má celkově 22, 8 rohů se třemi barvami a 12 hran se dvěma. Jednobarevné středy se do nich nezapočítávají, neboť jejich poloha se nikdy nemění. Jsou totiž středem osy každého otočení.

Každá kostička má distinktivní kombinaci barev, které za normálních okolností nemohou mít duplikát. Může se měnit pouze jejich poloha a tím i její otočení. Silou otáčet jednotlivé kostičky bez použití algoritmu učiní kostku neskladatelnou, dokud se neotočí zpátky.

## Algoritmizace

Jedním z prvků finální verze projektu by měl být algoritmus pro metodické složení kostky. Pokud se kostkou bude náhodně otáčet, složení by zabralo průměrně 43,252×1015 tahů. Existuje proto několik algoritmických sad, ze kterých se vytváří skládací metody. Nejznámějšími a nejpoužívanějšími z nich jsou CFOP a ROUX. Kvůli svojí znalosti o této metodice bude použita metoda CFOP.

Jako zdroj algoritmů se využije veřejná databáze SpeedCubeDB. [1]

### Metodika CFOP

Název označuje zkratky postupů, podle jakých se Rubikova kostka skládá. Jsou to: Cross (kříž), tedy jako první krok se poskládá kříž na jedné straně (barvě) kostky. Dále se poskládají první 2 vrstvy okolo vytvořeného kříže. Odtud název F2L (first two layers, první dvě vrstvy). Tyto kroky se většinou provádí podle intuitivního sledování a manipulování kostky, jejich algoritmy se téměř nepoužívají, pouze na nejvyšší kompetitivní úrovni. Zbývající dva kroky jsou naopak pouze algoritmické. Pro sadu OLL (Orientating the Last Layer, orientace poslední vrstvy) je celkem 57 algoritmů. Jejich cílem je pootáčet kostičky tak, aby celá vrchní strana byla kompletní, avšak kostka neposkládaná. O to se stará poslední krok, PLL (Permutating the Last Layer, permutace poslední vrstvy), který přemístí všechny zbylé kostičky. Výsledkem je poskládaná kostka. [2]

### Značení tahů

Algoritmy se zapisují podle celosvětové notace, kde každý tah má své písmeno z anglického názvu. Základní jsou R, L, U, D, F, B (pravá, levá, horní, spodní, přední, zadní). Otáčí se po směru hodinových ručiček. Každý tah se také řídí pravidlem, že pokud se za písmenem nachází apostrof nazývaný „prime“, tah se provede v opačném směru a pokud číslice 2, provede se posun 2krát.Rozšiřující jsou pak široké tahy w, rotace pohledu na kostku x, y, z a průřezové M, E, S. [3]

## Programová struktura kostky

Velmi důležitý faktor je, jak vytvořit algoritmus, který bude kostkou otáčet. Na základě toho se vytvoří celý systém a struktura kostky. Po troše hledání jsem došel ke dvěma možným způsobům. Kvůli převažujícím výhodám bude využit způsob 2.

### Způsob 1

Síla tohoto způsobu je nízká náročnost programovatelnosti.

Kostka by se skládala z 6 polí o dvou dimenzích neboli 3x3 matice. Každý prvek by znamenal jedno barevné pole na kostce.

Spočívá v tom, že při zadání příkazu na otočení strany se každá barva na každé ovlivněné kostičce natvrdo posune na její požadovanou pozici. Nevýhodou je neaplikovatelnost v chytrých algoritmech, neboť by se při nich musely samostatně kontrolovat všechny barvy. To by začinilo extrémní časovou náročnost programu  
a zbytečně dlouhý a nepřehledný kód.

### Způsob 2

Další varianta je těžší na programování, ale je kompatibilnější s dynamickým vývojem.

Namísto šesti matic se využijí pouze 3, jedna na každou vrstvu kostky. Jejich objekty by byly třídy kostičky, které by obsahovaly vlastnosti otočení a pevné očíslování základní pozice. Kostičky by měnily svoji aktuální polohu po směru hodinových ručiček relativně k ostatním. Pro získávání stavů kostky pro užití vhodného skládacího algoritmu by se kontrolovala pouze rotace a očíslování kostičky (vztahy mezi větším počtem kostiček).

## Grafické vykreslení

Pro vykreslení kostky se použijí 2 způsoby: síťové zobrazení kostky pro 2D  
a zobrazení ve 3D s možností rotace.

### 2D

Zobrazení celé kostky ve dvou dimenzích je možné pouze oddělením všech jejích stran tak, aby se mohla rozložit na plochu. Takové zobrazení je sice lehké vytvořit, je však velmi nepřehledné.

### 3D

Vykreslení ve třech dimenzích je optimální pro lidské pochopení. V takovém prostředí žijeme, tudíž je to pro nás přírodní. Dosáhnutí toho je však náročnější z programové stránky.

Je nutné vytvořit digitální prostředí pro objekty, které se budou pohybovat po třech světových osách (x, y, z). Obrazovka počítače je dvoudimenzionální, tudíž je nemožné prostor bez úprav zobrazit. Tento problém řeší ortodoxní projekce.

### Ortodoxní projekce

Prostor je zrealizován pomocí třídimenzionálních vektorů. Ty se ale nedají vykreslit na 2D plochu, neměly by hloubku a nic by z nich nešlo poznat. Proto se jednotlivé vektory vynásobí tzv. projekčními maticemi a výsledek je 2D bod na obrazovce s vypočítanými faktory pohledu, jako je vzdálenost od kamery, zorné pole rotace atd.[4]

## Použité technologie

### GitHub

GitHub je online služba, která poskytuje ukládání projektů a jejich přenos mezi zařízeními. Dělá se tak pomocí repositářů neboli prostorů na internetu, kam se ukládají data.[5]

### GitHub Desktop

Aplikace, která ulehčí práci s platformou GitHub. Detekuje všechny změny v souborech daného staženého repositáře na zařízení a umožňuje jejich nahrání nebo stažení pomocí jednoho tlačítka.[6]

### .NET

.NET (nazýván DOTNET) je spolehlivá aplikační platforma, ve které se programuje většinou v jazyku C#. Je dostupná na téměř všech operačních systémech. Poskytuje tzv. Full-stack neboli vývoj na straně klienta i serveru. Runtime, knihovny a jazyky jsou základem .NET struktury.[7]

### Windows Forms .NET Framework

Tento framework používá programovací jazyk C#. Ulehčuje grafickou stránku aplikací přes svoje komponenty UI (uživatelské rozhraní) a formulářový základ. Lehce se implementuje s programem pomocí událostí pro jednotlivé akce na komponentách formuláře. Také poskytuje nástroje pro kreslení vlastních tvarů do aplikace. [8]

Lze kreslit na samotný formulář, ale tato metoda je velice neefektivní pro výkonnostní prostředky počítače. Využije se pro to komponentu PictureBox, která je mnohem lépe optimalizovaná pro vykreslování.

### C#

C# je programovací jazyk na vysoké úrovni, odvozený z jazyka C. Struktura programu je převážně objektově orientovaná, pro většinu akcí se tedy využívají třídy a struktury. Funkční programový způsob znamená, že program operuje ve funkcích, které se dají použít jako jakákoliv jiná entita.[9]

# Praktická část

## Třídy 3D prostoru

Pro správnou funkci kostky je potřeba vytvořit 3D prostor pro její vykreslení. Skládá se třemi třídami, každá z nich posouvá prostor o dimenzi výše.

Při jejich vývoji jsem narazil na jeden zásadní problém, a to je kopírování objektů. Pokud se objekt přiřadí do nového pouze pomocí ‚=‘ jako pro normální proměnnou, zkopíruje se odkaz na daný objekt, takže nyní 2 kostičky měly identickou polohu a všechny ostatní vlastnosti.

Zprvu bylo implementováno řešení přes dědění třídy ICloneEnumerable, neboli přidání klonovací metody, ale objekty stále kopírovaly odkazy. Žádaný výsledek dosáhla nová deklarace tříd, která převzala jako jediný prvek svoji vlastní třídu a kopírovala vlastnost po vlastnosti do nového objektu.

### Vector3

Nejnižším možným prvkem ve všech dimenzionálních prostorách je bod neboli vektor. Tato třída obsahuje tři hodnoty pro posun po osách X,Y a Z. Spojením několika vektorů se dají vytvořit složitější tvary. Většinou se však používají pouze trojúhelníky nebo čtverce, ze kterých se dané tvary skládají.

Dalšími vlastnostmi třídy Vector3 jsou výpomocné proměnné, které určují, jak má vektor měnit svoji pozici při otáčení kostky. Všechny tři vlastnosti jsou vlastní třídy Vector3. Každý vektor kostky má daná pravidla pro každou osu zvlášť.

LengthFrom0 – Vzdálenost vektoru od středu kostky;

Displacement – Udává úhel (v radiánech), který vektor svírá s danou osou;

AnimState – Ukládá změněnou pozici neboli otočení okolo osy

Funkce CalcDisplacement() vypočítává délku a úhel od nuly. Nelze je na pevno nastavit, protože po každém otočení se hodnoty mění a je potřeba je přepočítat.

### Square

Ze čtyř vektorů se skládá jeden čtverec. Jediná další vlastnost je barva čtverce, která se nastavuje manuálně při vytváření kostiček.

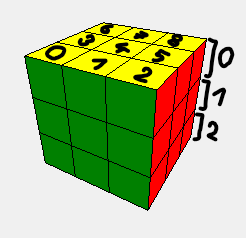
Čtverce mají na rozdíl od vektorů fyzickou podobu, takže se jimi vykresluje kostka.

### Cube

Přidáním třetí dimenze čtverci vzniká kostka – požadovaný tvar. Konkrétněji se jedná o kostičku, ze kterých se sjednocuje celá Rubikova kostka. Obsahuje pole šesti čtverců, které se dají nastavit osmi vektory a šesti barvami při vytváření objektu. Vlastnost cubeIndex udává správnou pozici kostičky. Její hodnoty jsou unikátní čísla 0 až 26, pro každou vrstvu 9. Využívá se především pro algoritmické skládání. Nezbytnou vlastností je rotateAxis neboli osa, po které se mají kostičky otáčet. Nastavuje se manuálně a pouze pro středové kostičky, které mají pouze jednu venkovní (nečernou) stranu. Pokud se s kostkou otočí tak, aby se změnila přední strana, tyto hodnoty se zkopírují společně se zbytkem vlastností a zajistí se tak správná rotace pro každou stranu. Bez tohoto otáčení bylo náročné s Rubikovou kostkou pracovat i přes to, že je na ni člověk zvyklý. Pro laika by uživatelský zážitek byl velice nepříjemný.

## Struktura kostky

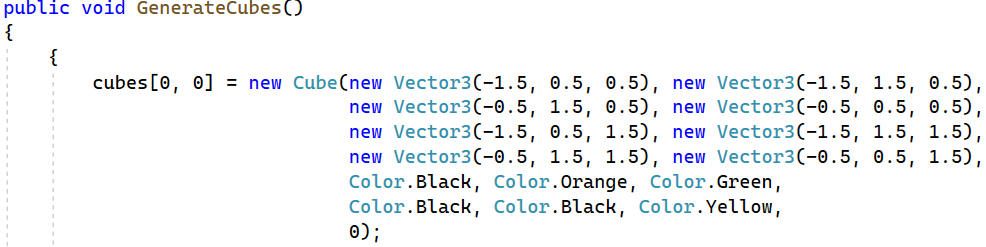
Fyzická Rubikova kostka se skládá z 22 kostiček, které se mohou pohybovat, 12 rohů a 4 hrany. Poloha středů se nikdy nemění. Programově je nutné tyto kostičky zahrnout, takže jich bude celkově 27. Jsou 3 vertikální vrstvy a každá z nich má 9 kostiček. Proto je kostka strukturována jako dvoudimenzionální pole kostiček.



Obrázek 1: Grafické znázornění pole kostky

### Deklarace kostky

Při deklaraci je pro každou z 27 kostiček zadáno 8 vektorů, 6 barev a její číselná pozice. Celá kostka má rozměry -1.5 až 1.5 ve všech směrech. Kdyby bylo potřeba, program se dá lehce přepsat na zadání délky strany a vypočítat jednotlivé kostičky.



Obrázek 2: Deklarace kostičky

## Vykreslování

### Projekční matice

### Pořadí vykreslení

## Tahy

Nezbytnou funkcí programu je otáčení stran kostky. Postup provádění tahů je kvůli struktuře kostiček obrácený od očekávaného postupu. Při příkazu tahu se znaky postupně zadají do fronty, kterou vytváří proměnná List<char>. V události komponenty Timer, která se provede každých 10 ms, se fronta kontroluje a nevhodné znaky vypustí. Pak se spustí animace, na jejiž konci se kostka vrátí do stejného tvaru, tah se vymaže z fronty a provede se programový posun.

### Vizuální změna polohy

Tahy se animují pomocí goniometrických funkcí sinu a cosinu, kterými se mění vlastnosti všech ovlivněných vektorů tak, aby po x krocích dosáhly nové pozice. Počet těchto kroků je ovlivněn rychlostí, která se upravuje v označené číselné komponentě.

### Programový pohyb

Druhý způsob z teoretické části sice pracuje na stejném principu jako ten první, kde se kostičky kopírují na své nové pozice bývalých kostiček, avšak je účinnější a kratší. Také se skrze něj dají snadno a dynamicky hledat pozice kostiček.

Všechny tahy se provádí ve funkci Turn(string), jejíž parametrem je požadovaný pohyb. Může obsahovat pouze písmeno tahu nebo i apostrof či 2 navíc. Podle toho program určí, jestli se má provést tah po či proti směru hodinových ručiček. Pokud obsahuje 2 za písmenem, přidá právě provedený tah na začátek fronty tahů a provede se ihned znovu.

Obsah obrázku text, snímek obrazovky, Písmo, číslo

Popis byl vytvořen automaticky

Obrázek 3: Programový pohyb kostiček při tahu

## Automatické poskládání

### Vrácení všech tahů

### Algoritmická metoda CFOP

Začalo se zezadu, tedy krokem PLL. Pozice horní vrstvy se uspořádaly do jednoho řetězce, podle kterého se následně hledaly algoritmy.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Obrázek 4: Detekce algoritmu

Na obrázku lze vidět formát složeného řetězce. Lze na něm vidět, že 4 různé stavy odkazují na stejný algoritmus, v tomto případě označený „Aa“. Jeden PLL algoritmus může na kostce existovat v 16 různých stavech a těchto algoritmů je 21. Další částí je OLL, kde se namísto pozic kostiček formátuje jejich rotace. Tyto algoritmy už mají pouze 4 stavy, ale existuje jich 57. Napsat detekce všech 78 algoritmů by zabralo značně času. Nejtěžší jsou však kroky kříž a F2L. Ty se totiž provádějí intuitivně, a to počítač bez umělé inteligence nedokáže. Bylo by nutné napsat algoritmus, který detekuje desítky až stovky možných pozic všech kostiček a pak najít vhodný algoritmus pro jejich dosazení na správnou pozici se správnou rotací.

K tomu jsem bohužel z časových nedostatků nevěnoval příliš pozornosti. Algoritmus pro PLL funguje bezchybně, avšak není do aplikace implementován. Vrátil jsem tedy z většiny funkční poskládání pomocí obrácení všech provedených tahů a opravil zbylé chyby.

Závěr

Seznam použitých zdrojů

Seznam použitých symbolů a zkratek

|  |  |  |
| --- | --- | --- |
| Zkratka | Celý název |  |
| CFOP | Kříž F2L OLL PLL |  |
| F2L | První 2 vrstvy |  |
| OLL | Orientace poslední vrstvy |  |
| PLL | Permutace poslední vrstvy |  |
|  |  |  |
|  |  |  |

Seznam obrázků

[Obrázek 1: Grafické znázornění pole kostky 12](#_Toc162374810)

[Obrázek 2: Deklarace kostičky 13](#_Toc162374811)

[Obrázek 3: Programový pohyb kostiček při tahu 14](#_Toc162374812)

[Obrázek 4: Detekce algoritmu 14](#_Toc162374813)

Seznam příloh

Prázdná šablona maturitní práce