UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Zdeněk Sýkora: Struktury

Bc. Jiří Hora

Diplomová práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří Hora**
Osobní číslo:	**I15208**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Zdeněk Sýkora: Struktury**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je popsat principy, které využíval Zdeněk Sýkora ve svých tzv. strukturálních obrazech.

Tyto obrazy jsou vytvářeny opakovaným použitím několika základních elementů, které mají specifické vnitřní členění a mají čtvercový nebo obdélníkový tvar. Vzájemnou polohu elementů určuje zvolené pravidlo: navazují barvami a tvary, navazují barvami a nenavazují tvary, nenavazují barvami a navazují tvary, nenavazují barvami ani tvary. Při vlastní tvorbě Zdeněk Sýkora používal tzv. partituru s polohou výchozích elementů a znamének plus a mínus. Znaménka určují, kde se přičítá nebo odečítá koeficient, který má funkci urychlení nebo zpomalení přechodů od světlých elementů k tmavým nebo naopak. Přesný popis koeficientu nejspíše není známý. Roli tohoto koeficientu je třeba analyzovat statistickými metodami na některém obraze, u kterého je k dispozici partitura.

Hlavním úkolem je vytvořit aplikaci, která umí obrazy na bázi Sýkorových pravidel vytvořit. Způsob implementace je ponechán na volbě diplomanta. U strukturálních obrazů bude třeba zvolit vhodnou reprezentaci struktur v počítači a naprogramovat volbu z množiny přípustných struktur. U některých obrazů vznikajících rotací a výřezem ze základního rastru je třeba aplikovat vhodné transformace souřadnic.

Součástí závěrečné práce bude uživatelská příručka popisující aplikaci.

Rozsah grafických prací: **15**

Rozsah pracovní zprávy: **35**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

***KAPPEL, Pavel. Zdeněk Sýkora 90. Praha: Verzone, 2010. 106 s. ISBN 978-80-904546-2-0.**
***MAREK, Jaroslav a Marie NEDVĚDOVÁ. Historie digitálního umění: náhoda, počítač a linie Zdeňka Sýkory. In: Bečvář, Jindřich a Martina Bečvářová (eds.). 37. mezinárodní konference Poděbrady 2016. Praha: MATFYZPRESS, nakladatelství Matematicko-fyzikální fakulty Univerzity Karlovy, 2016, s. 137-146. Dostupný též z WWW: <https://www.fd.cvut.cz/personal/becvamar/konference/konference%20HM%2037%20-%20text%20na%20web.pdf>.**
***SÝKORA, Zdeněk a Jaroslav BLAŽEK. Computer aided Multi element Geometrical Abstract Paintings. Leonardo. 1970, roč. 3, s. 409.**

Vedoucí diplomové práce: **Mgr. Jaroslav Marek, Ph.D.**
Katedra matematiky a fyziky

Datum zadání diplomové práce: **31. října 2016**

Termín odevzdání diplomové práce: **17. května 2017**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. sprna 2017

Bc. Jiří Hora

Poděkování

Děkuji svému vedoucímu práce Mgr. Jaroslavu Markovi, Ph.D. za pomoc při vypracování této práce.

Děkuji také všem, kteří mě podporovali po celou dobu studia.

Anotace

Tato diplomová práce se zaměřuje na tvorbu českého malíře Zdeňka Sýkory – zejména na jeho Struktury. Jsou popsány principy tvorby strukturálních obrazů a Černobílá Struktura. V první části se autor věnuje životu a uměleckému dílu Zdeňka Sýkory. Ve druhé části popisuje systém pro tvorbu struktrálních obrazů. Dále popisuje implementaci aplikace, která tvoří obrazy na základě principů tohoto malíře. V závěru se věnuje generování Struktur na základě vzorového zadání.

Klíčová slova

Zdeněk Sýkora, digitální umění, obrazy, struktury, aplikace, C#, Windows Forms

Annotation

This diploma thesis focuses on the work of Czech painter Zdeněk Sýkora – especially on his Structures. The principles of structural images and the Black and White Structure are described. In the first part the author focuses on the life and work of Zdeněk Sýkora. The second part describes a system for creating structural images. It also describes the implementation of an application that creates images based on the principles of this painter. At the end, it focuses on generating Structures based on sample assignments.

Keywords

Zdeněk Sýkora, digital art, paintings, structure, application, C#, Windows forms

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

PNG	Portable Network Graphics
PDF	Portable Document Format
JSON	JavaScript Object Notation

Úvod

THE 'Earth' WITHOUT 'art' IS JUST 'eh'

Zdeněk Sýkora (1920–2011) je považován za jednoho z průkopníků užití počítače při tvorbě výtvarného díla.V roce 1964 vytvořil ve spolupráci s matematikem Jaroslavem Blažkem propracovaný systém, který napomáhal tvořit obrazy (Struktury) za pomoci počítače. Smyslem tohoto systému bylo kombinatorické vyčerpání všech možných vzájemných poloh elementů dle zvolených pravidel. Princip tohoto systému popsali Sýkora s Blažkem v roce 1970 článku Computer-aided multielement geometrical abstract paintings v časopise Leonardo. Od strukturálních obrazů se Sýkora ve své tvorbě posunul k liniím, které objevil právě ve strukturálních obrazech.

Tato diplomová práce a aplikace, která je její součástí, se věnuje strukturám. Cílem je nastudovat systém z článku v časopise Leonardo a vytvořit aplikaci, která bude vytvářet strukturální obrazy na základě definovaných pravidel.

Aplikace je vytvořena v programovacím jazyce C# s využitím vývojového prostředí Microsoft Visual Studio 2017 Enterprise. Pro grafické rozhranní aplikace byl zvolen framework Windows Forms, který je součástí frameworku .NET. Aplikace umožňuje tvorbu a načítání tzv. partitur, které jsou textovou definicí strukturálních obrazů. Na základě partitur pak aplikace „kreslí“ výsledné obrazy. Vzniklé obrazy ukládá ve formátu PNG. Zároveň aplikace umožňuje načítání struktur.

První kapitola se věnuje životu a dílu Zdeňka Sýkory. Druhá kapitola popisuje „Černobílou strukturu“ a systém z článku v časopise Leonardo. Třetí kapitola se věnuje implementaci aplikace. Je zde popsána struktura tříd v aplikaci a podrobněji rozebrány stěžejní části aplikace. Čtvrtá kapitola popisuje použití aplikace. Poslední kapitola se věnuje generování Struktur na základě partitury z článku v časopise Leonardo.

1 Zdeněk Sýkora

„Několikrát v životě jsem měl pocit, že jsem ve své práci zabloudil. Do každé nové situace jsem byl pomalu vtahován silami, kterým jsem se nemohl a ani nechtěl bránit. Nové výsledky na mě vždy působily cize, přestože vznikaly spontánně a nebyly výsledkem nějakého rozhodnutí nebo úmyslu. Vždy až po delší době jsem jim začal rozumět a uvědomil jsem si, že předešly mé myšlení.“

Z. Sýkora v rozhovoru s V. Čapkem [2]

1.1 Život Zdeňka Sýkory

Zdeněk Sýkora byl český výtvarník, který je považován za průkopníka využití počítače v přípravě uměleckého díla. Narodil se 3. února 1920 v Lounech a zemřel 12. července 2011 tamtéž.

V roce 1938 maturoval na lounském reálném gymnáziu a poté nastoupil na Vysokou školu báňskou. Po uzavření vysokých škol v roce 1939 pracoval na dráze v Lounech a na konci války byl výpravčím v Českém Brodě. V tomto období se věnoval fotografování a experimentoval s technikou koláže.

V letech 1945–1947 studoval výtvarnou výchovu a deskriptivní geometrii a modelování na Univerzitě Karlově v Praze. Po ukončení studia začal vyučovat jako asistent na katedře výtvarné výchovy Pedagogické fakulty. [3]

Na podzim roku 1963 byla v Praze založena tvůrčí skupina „Křižovatka“, jíž byl Sýkora členem. Další členové byli: Vladimír Burda, Richard Fremund, Josef Hlaváček, Jiří Kolář, Běla Kolářová, Jan Kubíček, Karel Malich, Pavla Mautnerová, Vladislav Mirvald, Jiří Padrta, Otakar Slavík. Skupina vyhlašovala, že chce „čelit převaze romantického sentimentu a přemíře subjektivity, jako obraz k objektivním tendencím opřeným o důsledné principy konstruktivního pořádku, proporce a čísla.“ [5]

Zdeněk Sýkora, jako jeden z mála Čechů, uspěl i v zahraničí. O jeho úspěchu a uznávanosti svědčí řada ocenění, které získal. Jedním z nich je titul rytíře *Řádu umění a literatury*, který mu v roce 2003 udělil francouzský ministr kultury. V roce 2005 mu v rakouském Salcburku byla udělena *Herbert-Boeckl-Preis* za celoživotní dílo. V roce 2009 se stal držitelem

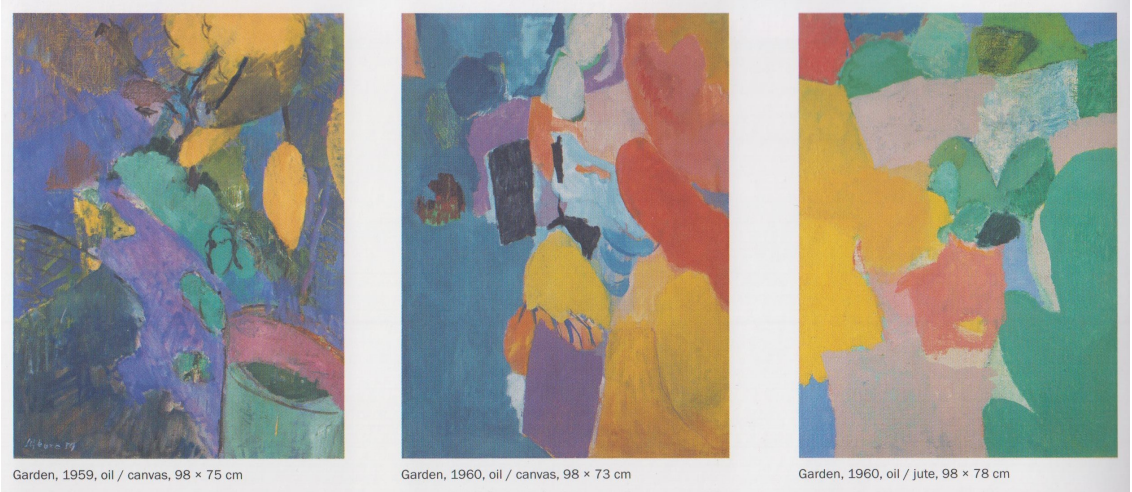
Ceny Vladimíra Boudníka za rok 2008.

Sýkorovu úspěšnost potvrzuje i fakt, že jeho díla jsou zastoupena ve světových sbírkách (například ve vídeňském Muzeu moderního a současného umění či ve sbírkách Centre Pompidou v Paříži).

1.2 Dílo Zdeňka Sýkory

Tvorba Zdeňka Sýkory je inspirována přírodou a krajinou. Po surrealistických a kubistických obrazech ze 40. let se nejprve věnoval realistické krajinomalbě a malbám zátiší. Maloval okolí svého rodného města, především řeku Ohři, často společně s přítelem a bývalým spolužákem Vladislavem Mirvaldem.

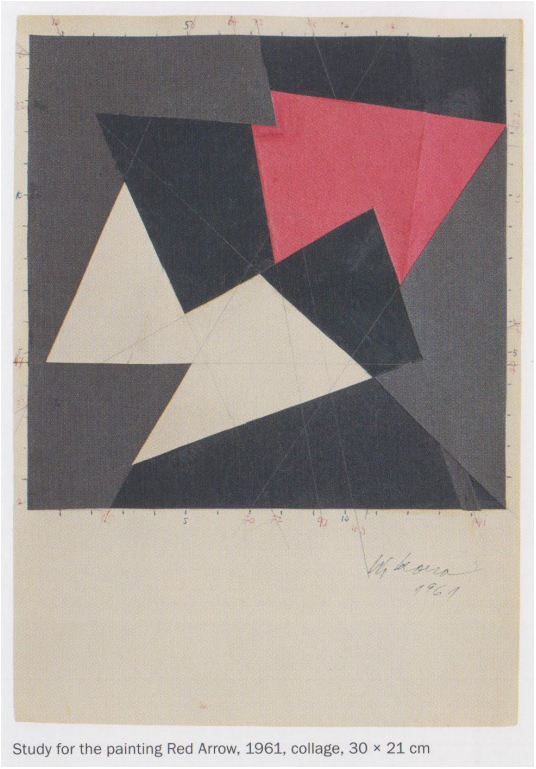
V průběhu 50. let jeho tvorba zaznamenala proměny malířských stylů od impresionismu, přes fauismus, až ke kubismu a zjednodušovala se. Opouští budování iluzivního prostoru a řeší především vztah mezi barevnými plochami. Vyvrcholením jeho snahy v tomto období je cyklus *Zahrady* (1958–1960), kde krajina byla impulzem k volnému pojednání výrazových kvalit osvobozené barevné skvrny.



Obrázek 1: Ukázka z cyklu Zahrady. Zdroj:[1].

Na přelomu 50. a 60. let se Sýkorova tvorba začala přiklánět k abstrakci a vyvrcholením tohoto období byly jeho Struktury.

Cestu ke strukturám, které tvoří velmi rozsáhlou a důležitou část jeho díla, vytyčil obraz *Červená šipka* (obrázek 2). Tento obraz byl inspirován Sýkorovou návštěvou Ermitáže



(a) Konstrukce

(b) Výsledný obraz

Obrázek 2: Červená šipka. Zdroj:[1].

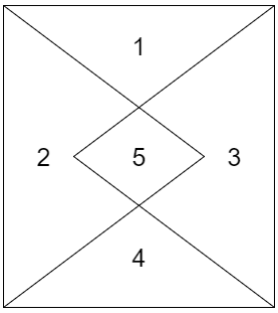
v dnešním Petrohradě, kde se osobně střetl s obrazy Henriho Matisse. Obraz je čtvercového formátu a vznikl pomocí řídících bodů na krajích čtverce, kterými Sýkora vedl linky. Ty vytvořili určité plochy, které Sýkora dle svého vnímání různě vybarvil.

1.2.1 Kombinatorické struktury

Sýkorovy struktury jsou obrazy tvořené z jednoduchých geometrických elementů. Tyto elementy mají jedinečný tvar (z počátku obdélník, později čtverec) a specifický vnitřní vzor.

Právě pro tvorbu kombinatorických struktur začal Sýkora používat počítač. Ten však nebyl hlavním prvkem v jeho tvorbě, ale sloužil pouze jako pomocník. Díky němu měl Sýkora možnost prozkoumat kombinatorické možnosti vzájemných poloh elementů. Sýkora si stanovil jistá pravidla, která předal počítači a výsledek pak respektoval. Mohla totiž nastat situace, kdy by se jeden element několikrát za sebou opakoval. Takovou situaci by sám autor umělec asi nepřipustil.

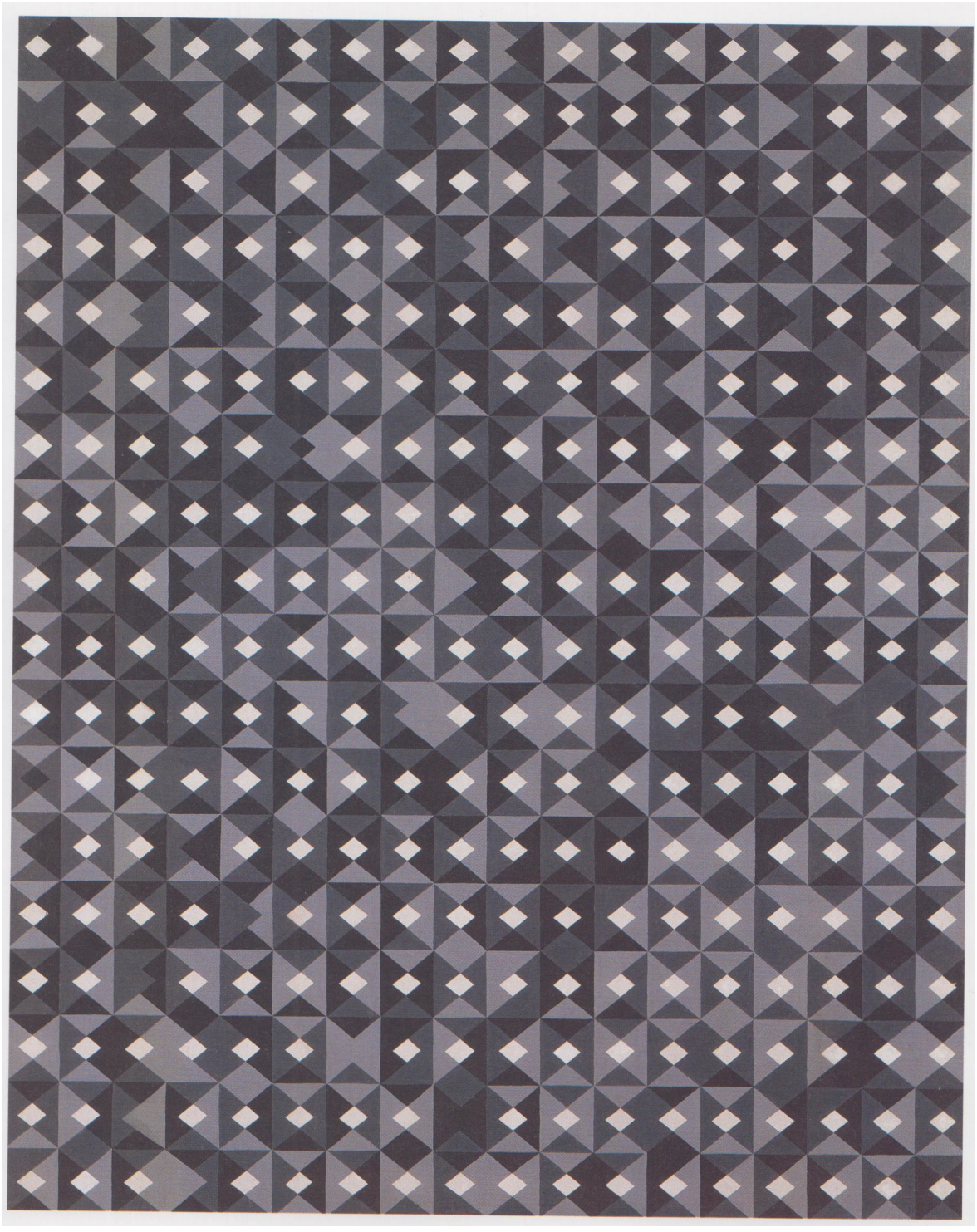
První strukturální obraz vznikl v roce 1963 a Sýkora jej nazval *Šedá struktura*. Obraz se skládá z 324 (18 x 18) elementů obdélníkového tvaru (obrázek 3). Každý element v obraze se skládá ze dvou rovnoramenných trojúhelníků směřujících proti sobě, jejichž průnikem je kosočtverec. Tímto spojením vznikne 5 dílčích částí, které jsou obarveny jednou ze 4 barev – černá, bílá nebo 2 odstíny šedé (světle šedá a tmavě šedá). Barvy se mohou opakovat. Z toho vychází $4^5 = 1024$ variant různého vybarvení jediného elementu. Sýkora se snažil o to, aby se barvy co nejméně opakovaly.



Obrázek 3: Základ pro element Šedé struktury. Zdroj:vlastní.

V roce 1964 Sýkora společně s Jaroslavem Blažkem vytvořili na počítači LPG-30 řadu programů, které vedly k vyčerpání všech možných kombinatorických možností základních elementů v obraze. V roce 1966 společně vytvořili *Černobílou strukturu* z více tvarově odlišných elementů [6]. Černobílé struktuře se podrobně věnuje kapitola 2.

V roce 1973 se Sýkora po několika výrazových obměnách (barevnost, velikost, polohy elementů) přestal věnovat geometrickým tvarům a začal se věnovat tzv. *makrostrukturám*. Ty vznikali zvětšováním elementů, prakticky jako detaily již existujících struktur. Vzniklé linie, vytyčující hranici mezi bílou a černou plochou, byly Sýkorovi podnětem ke vzniku *liniových obrazů*.



Obrázek 4: Šedá struktura. Zdroj:[1].

(a) Makrostruktura 1972

(b) Makrostruktura 1973

Obrázek 5: Makrostruktury. Zdroj:[1].

1.2.2 Liniové obrazy

Makrostruktury navíc zdůraznily elementový základ. Byla to vlastně náhoda, že jsem si tím sám sobě dal před oči výrazovou a významovou sílu linií, které vznikaly spojováním kruhových částí jednotlivých elementů jako hranice mezi jejich černou a bílou plochou. Linie mne stále více přitahovaly.

Z. Sýkora v rozhovoru s V. Čapkem [2]

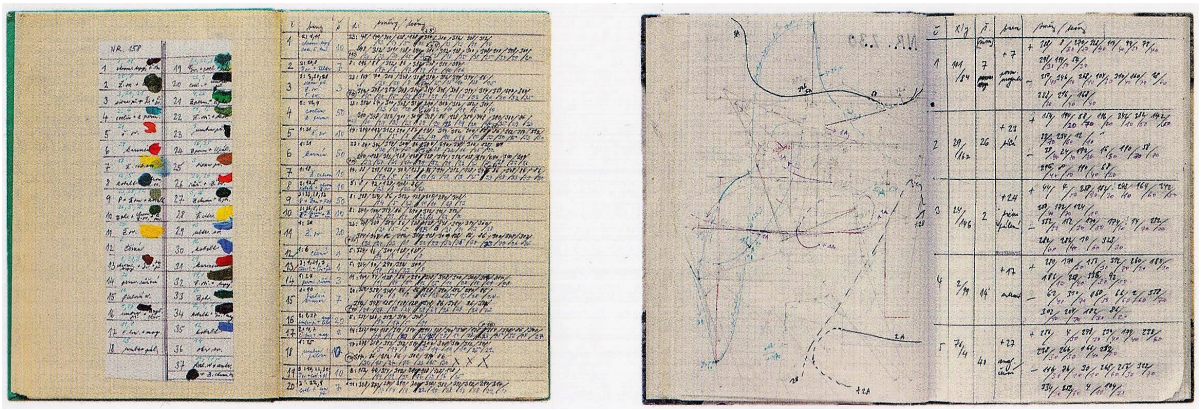
Základem liniových obrazů, jak již název napovídá, se staly křivky a linie. Liniím se Sýkora věnoval od 70. let až do svého skonu. Zpočátku jsou Liniové obrazy, stejně jako Struktury, umístěny do rastru, ale později od toho Sýkora upustil. Obraz *První linie* vnikl v roce 1973 a skládal se ze 3 linií, přičemž všechny měly stejnou tloušťku.

Linie jsou určeny barvou, tloušťkou a průběhem. Ve výsledném obrazu je důležitý i počet linií. Všechny tyto aspekty určuje počítač generováním náhodných čísel. Toto je nejpodstatnější rozdíl oproti strukturám, u nichž se Sýkora snažil náhodu minimalizovat.

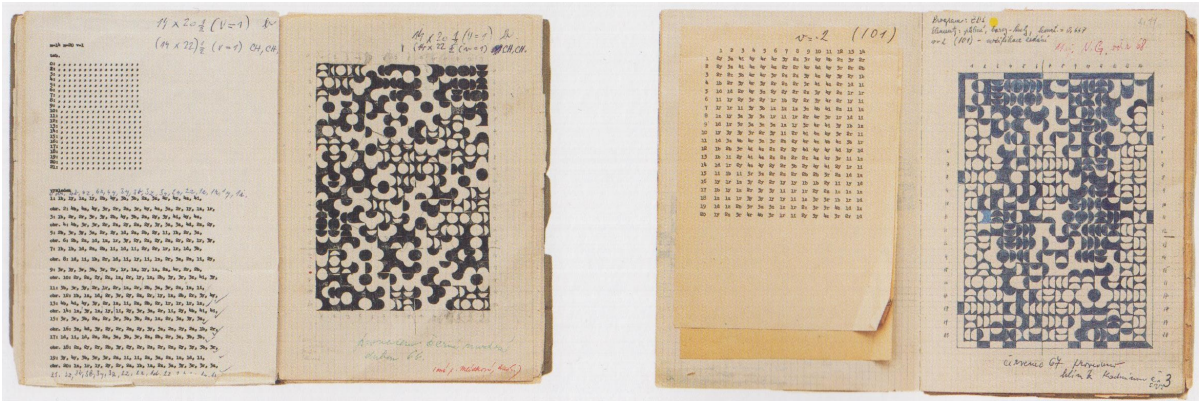
Linové obrazy tvoří křivky, které se skládají z oblouků a úseček nebo pouze z jediného bodu. Oblouky mají tvar části kružnice a vznikají pomocí geometrické konstrukce. Vstupními parametry jsou počáteční bod, barva, tloušťka pera a směry a délky tečen. Tyto

hodnoty, které generoval počítačem, Sýkora zapisoval do tzv. *partitury*.

Partitura neboli „skóre“ je uspořádaný zápis hodnot, podle kterých vznikne obraz. U linií partitura obsahuje počet linií, a pro každou z nich poté její počáteční bod, dobu života, směr, tečnu, barvu a šířku. U struktur pak partitura obsahuje pouze zápis použitých elementů.



Obrázek 6: Ukázka partitury pro linie. Zdroj:[1].



Obrázek 7: Ukázka partitury pro struktury. Zdroj:[1].

Při konstrukci řešil také vzájemné křížení linií. Generovaly se čísla 1 a 2. Číslo 1 znamenalo, že linie bude namalována přes druhou, v případě čísla 2 byla linie namalována pod jinou. Délku kroků vnímal jako „dobu života“, což metaforicky představovalo lidské osudy. Pro každou linii se snažil vybrat unikátní barvu.

První výstava liniových obrazů proběhla v nizozemském Gorinchemu v roce 1979. Od roku 1983 Sýkora pracoval na jeho nejvýznamějším liniovém díle. Později ho nazval

„Poslední soud“. Obraz obsahoval 227 linií, měl rozměr 3 x 3 metry a Sýkorovy zabrala tvorba jeden rok.

Základní ideje pro tvorbu Linií Sýkora uváděl v rozhovorech s Vítkem Čapkem [2]. V některých svých svých knihách publikoval i partitury pro některé Linie. K dispozici jsou v knize (viz [1]) partitury pro Linie č. 145, špatně čitelná partitura pro Linie č. 158, v knize (viz [10]) pak partitura pro Linie č. 96 (serigrafie 7 linií pro Uměleckou besedu) a část partitury pro serigrafii 12 linií pro Heinze Teufla [9]. Nicméně konkrétní algoritmus je publikován patrně pouze v článku *Construction and statistical analysis of Zdeněk Sýkora's lines* [8].

Konstrukce linie:

1. Linie začíná v bodě P_1.
2. Nakreslí se tečna t_1 ve směru α_1; $P_1 \in t_1$.
3. Určí se bod O; $|OP| = d$; (vzdálenost d Sýkora nazýval „délka tečny“).
4. Nakreslí se přímka t_2 v daném směru α_2; $O \in t_2$.
5. Určí se osa o přímek t_1 a t_2, o je dané směrem $\varphi = \frac{\alpha_1+\alpha_2}{2}$.
6. Zkonstruuje se kolmice n_1 k tečně t_1, pata této kolmice je bod P_1.
7. Určí se střed kružnice S; $S \in n_1 \cap o$.
8. Zkonstruuje se kolmice n_2 k tečně t_2, $S \in n_2$.
9. Určí se bod P_2; $P_2 \in n_2 \cap t_2$.
10. Nakreslí se oblouk na kružnici se středem S a poloměrem $r = |SP_1| = |SP_2|$; oblouk je určený předchozími tečnami a body. Poloměr kružnice $r = d \tan(\alpha_2 - \alpha_1)$.

Obrázek 8: Ukázka z liniové tvorby. Zdroj:[1].

Když Sýkora mluví o svém umění, říká: *„Chvílemi jsem byl zděšen, kam jsem se to vlastně dostal. Brzy jsem si ale uvědomil, že jsem opět na počátku. Na topolech se chvělo listí a třpytila se řeka“*. [4]

Poslední věta poukazuje na fakt, že i přes technologický pokrok a matematiku, se opět vrací k přírodě. Z té celá Sýkorova tvorba vychází.

1.2.3 Architektura

Sýkora je také autorem realizací v architektuře. Například mozaikového obkladu větracích komínů Letenského tunelu v Praze (dokončeno 1969), návrhu dlažby pro náměstí v nizozemském Gorinchen (1974, rekonstrukce 2005), mozaiky do kulturního domu v Litvínově (1975, rekonstrukce 2016), chodníku a stěny rovněž v Litvínově (1977), řešení vstupní haly firmy Selmoni AG v Basileji (1993) či vstupní haly budovy Národního integrovaného střediska Řízení letového provozu České republiky v Jenči u Prahy (spoluautorkou Lenka Sýkorová, 2005). [4]

Obrázek 9: Chodník a zeď v Litvínově, větrací komíny Letenského tunelu. Zdroj:[1].

1.3 Generování náhodných čísel

K přípravě svých děl využíval Sýkora náhodných čísel, která získával různými způsoby. Nejprve používal hrací kostku nebo z pytlíčku tahal žetony popsané čísly. Jaroslav Blažek,

matematik, s nímž Sýkora pracoval během období struturálních obrazů, mu později dodával kartičky s řadou náhodných čísel v požadovaných intervalech. (1-7, 1-12, 1-24, později 1-30, 0-360, 0-150, 0-200, 1-2, 1-3). Na počátku 80. let získával pseudo-náhodná čísla z počítačového generátoru od Jindřicha Plocha a Pavla Raimana, z něhož se stal dlouholetý blízký spolupracovník a poradce, zvláště v oborech matematiky a geometrie [1].

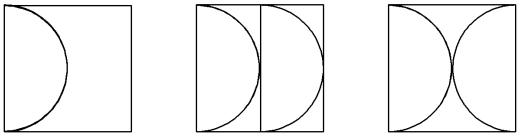
Obrázek 10: Pomůcky pro získávání náhodných čísel. Zdroj:[1].

2 Černobílá struktura

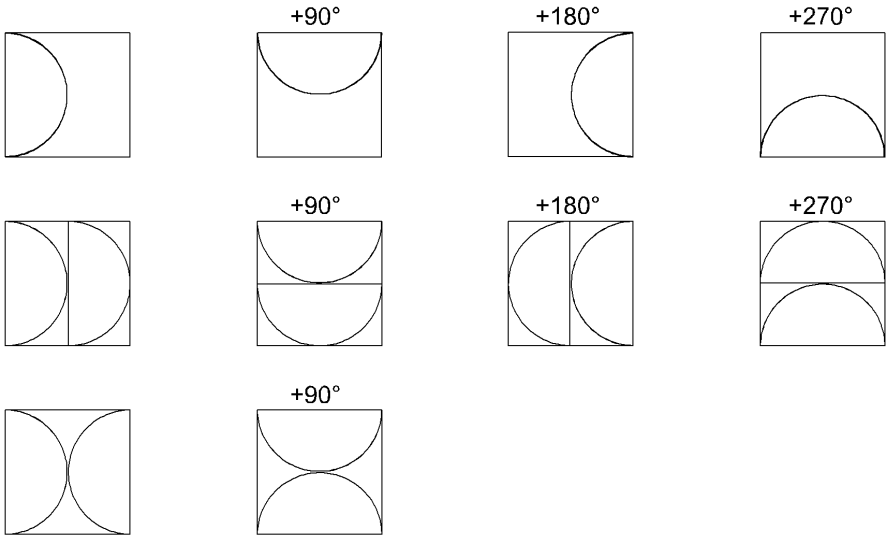
Černobílou strukturu vytvořil Sýkora společně s Jaroslavem Blažkem v roce 1966. Popsali ji společně v článku *Computer-aided multielement geometrical abstract paintings* v časopise *Leonardo* [6].

2.1 Základní elementy

Černobílá struktura je složena z více tvarově odlišných elementů. Základní elementy jsou 3 – obrázek 11. Jedná se o čtverec a v něm umístený jeden nebo dva půlkruhy. Půlkruhy mohou být umístené za sebou nebo proti sobě. Nikdy ne tak, aby společně tvořili celý kruh. Postupným otáčením základních tvarů o 90° získal Sýkora 10 elementů – obrázek 12.

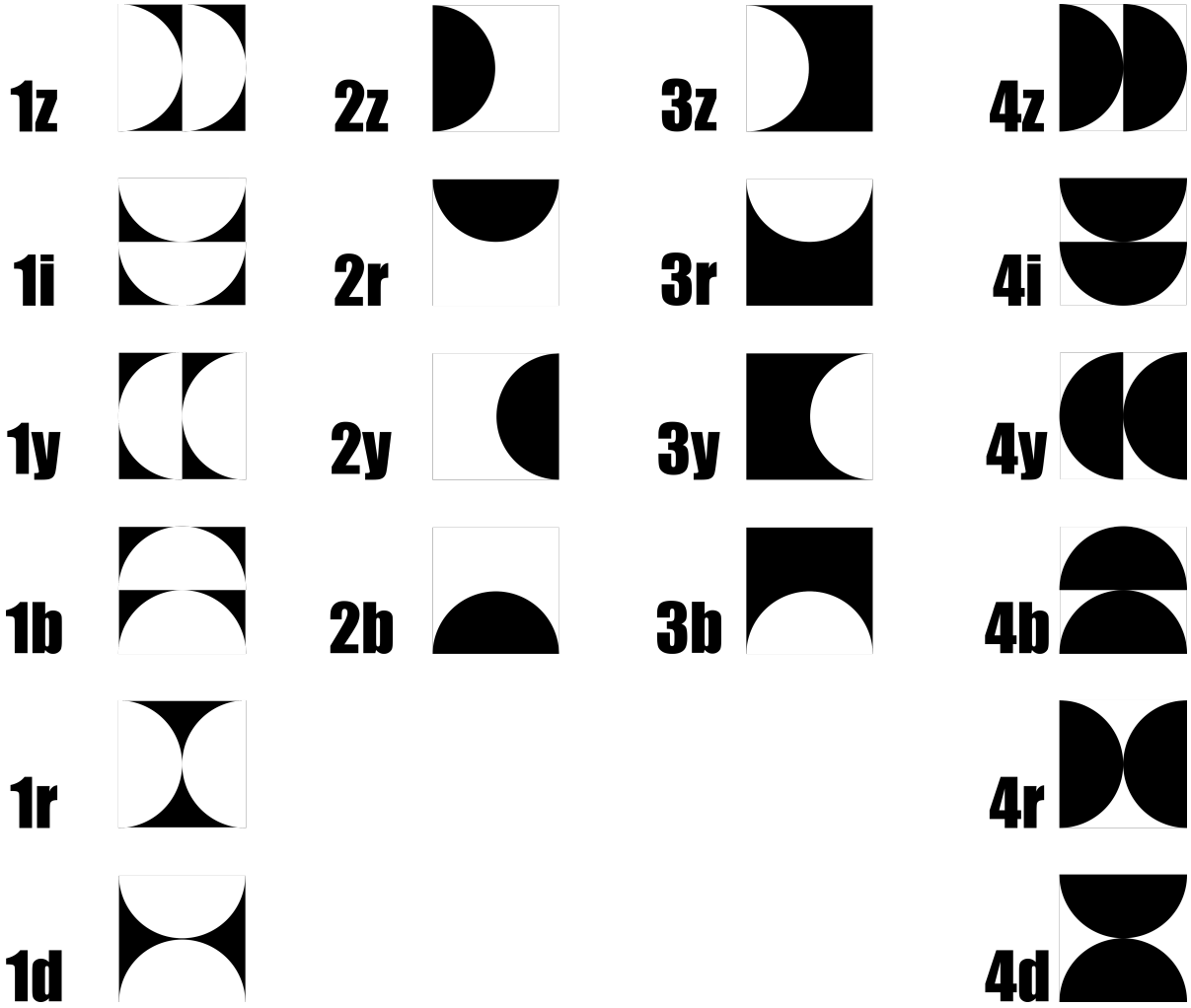


Obrázek 11: Tři základní tvary pro elementy Černobílé struktury. Zdroj: vlastní.



Obrázek 12: Základní tvary pro elementy Černobílé struktury s rotacemi. Zdroj: vlastní.

Sýkora ještě přidal černou a bílou barvu a tím získal finálních 20 elementů – obrázek 13. Tyto elementy rozdělil do 4 skupin podle světelnosti a označil je písmeny *z, i, y, b, r* a *d*. První skupina obsahuje *nejsvětlejší prvky*, druhá a třetí *tmavší* a čtvrtá skupina obsahuje prvky *nejtmavší*. První a čtvrtá skupina obsahuje totožné elementy, pouze se liší barevným provedením. Stejně tak tomu je u druhé a třetí skupiny.



Obrázek 13: Dvacet elementů pro Černobílou strukturu. Zdroj: vlastní.

2.2 Pravidla

Pravidla jsou důležitou součástí Černobílé struktury, tak jak ji Sýkora definoval. Sýkora definoval 4 pravidla, která se o opírají o pokračování či nepokračování barvou a tvarem. Pravidla jsou 4:

- V = 0 – pokračuje barva; pokračuje tvar,
- V = 1 – pokračuje barva; nepokračuje tvar,
- V = 2 – nepokračuje barva; pokračuje tvar,
- V = 3 – nepokračuje barva, nepokračuje tvar.

Říkáme, že *barva pokračuje*, jestliže je stejná jako barva na podélné straně sousedního elementu.

Říkáme, že *tvar pokračuje* na straně elementu každého půlkruhu otevřeného k jedné straně, která se připojuje k půlkruhu nebo pokud se spojují dva vzory, z nichž ani jeden není půlkruh otevřený k jedné straně. Volně přeloženo z [6].

Sousedním elementem se myslí element, který s dalším elementem sdílí stranu. Pokračování barvy má přednost před pokračovám tvaru. Znamená to, že v pravidle 0 se vybírají nejprve elementy pokračující barvou a až po té se ze zbylých vybírají elementy pokračující tvarem.

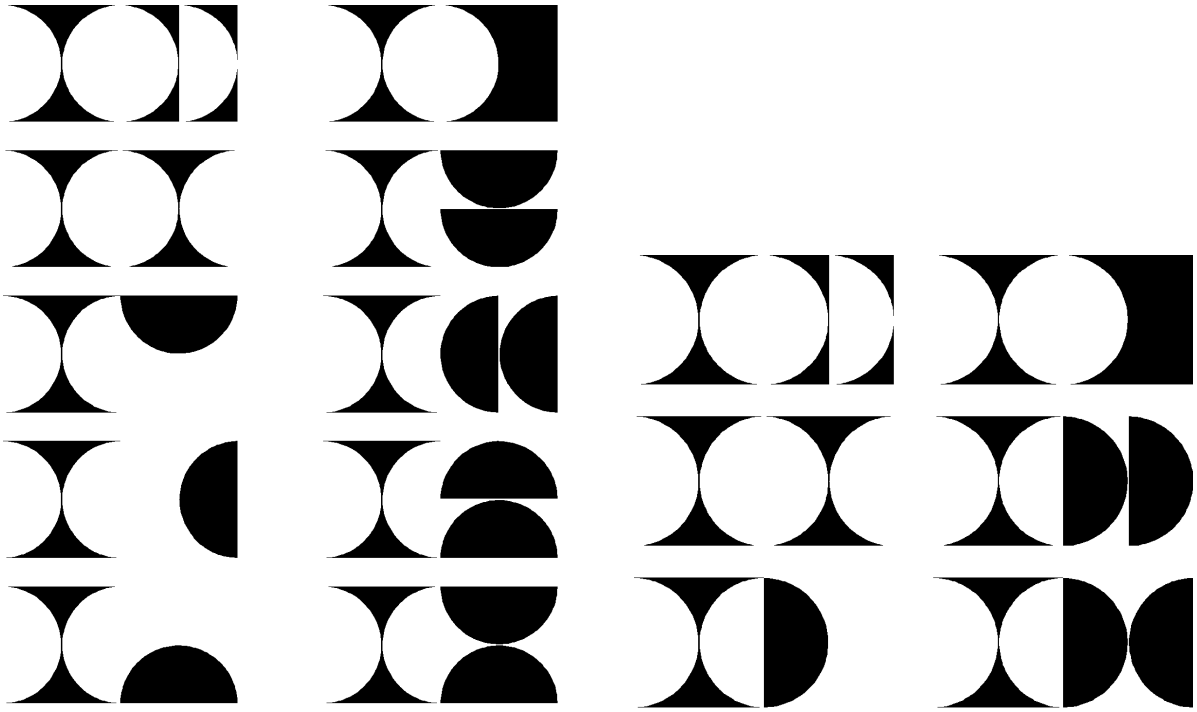
Příklad: element 1r. Navazující elementy jsou uvedeny v tabulce 1. Návaznosti v případě konkrétních pravidel jsou uvedeny v tabulce 2. Obrázek 14 ilustruje návaznosti z tabulky 1.

Tabulka 1: Navazující elementy elementu **1r**

elementy navazující barvou	**1z**, **1r**, **2r**, **2y**, **2b**, **3z**, **4i**, **4y**, **4b**, **4d**
elementy navazující tvarem	**1z**, **1r**, **2z**, **3z**, **4z**, **4r**

Tabulka 2: Navazující elementy k **1r** dle pravidel.

pravidlo	**navazující elementy k 1r**
V=0; pokračuje barva, pokračuje tvar	1z, 1r, 3z
V=1; pokračuje barva, nepokračuje tvar	2r, 2y, 2b, 4i, 4y, 4b, 4d
V=2; nepokračuje barva, pokračuje tvar	2z, 4z, 4r
V=3; nepokračuje barva, nepokračuje tvar	1i, 1y, 1b, 1d, 3r, 3y, 3b



(a) Elementy navazující barvou.

(b) Elementy navazující tvarem.

Obrázek 14: Návazující elementy k elementu **1r**. Zdroj:vlastní.

2.3 Algoritmus umisťování elementů

Základem pro spuštění algoritmu je zadání od autora. Autor definuje následující:

- Struktura obsahující elementy a pravidla,
- počet sloupců,
- počet řádků,

- koeficient přechodu c; c > 0, Sýkora v článku [6] uvádí příklad 0,75,
- zvolené pravidlo,
- do libovolných buněk volitelně umístí:
  - plus, minus nebo element.

Tím končí tvůrčí činnost autora a následuje činnost algoritmu.

1. Algoritmus začíná v buňce [1,1] (první řádek, první sloupec).
2. Prochází liché řádky zleva doprava a sudé řádky zprava doleva.
3. K aktuální buňce najde jeho sousední buňky a hodnoty v nich. Algoritmus zajímají pouze buňky, které obsahují elementy – symboly plus a minus nehrají roli.
   3.1. Z okolních buněk obsahujících elementy zjistí, do které skupiny patří a z hodnot spočte průměr.
   3.2. Pokud buňka obsahuje „+“ nebo „-“, přičte, případně odečte koeficient.
      3.2.1. Pokud je v tomto momentě výsledek x,5 (např. 3,5) a zároveň větší než 0 a menší než 4, algoritmus se vrací do bodu 3 a prohledává širší okolí. Nejdále však z okruhu vzdáleného o 4 buňky od počítaného.
   3.3. Výslednou hodnotu zaokrouhlí na celé číslo. Pokud je hodnota menší než 1, zaokrouhlí se na 1. Pokud je hodnota větší než 4, zaokrouhlí se na 4. [1] Výsledek pak určuje, ze které skupiny se vybírá element pro umístění do buňky.
4. Algoritmus aplikuje pravidlo pro výběr elementu. Pravidlo se skládá ze dvou částí – barva a tvar, kde barva má přednost.
   4.1. Z příslušné skupiny se vyhledají elementy navazující/nenavazující [2] barvou. Hledání probíhá tak, že algoritmus vyhodnocuje sousední strany buněk na severu, jihu, západě a východě, pokud jsou k dispozici. Nejprve se snaží najít

[1] Obecně pokud je hodnota větší než počet skupin, zaokrouhlí se na číslo poslední skupiny.
[2] Záleží na zvoleném pravidle

shodu na všech možných stranách. Pokud se to nepodaří, hledá shodu alespoň na jedné straně (viz příklad v 2.3.2 na straně 35).

Pokud neodpovídá žádný element, vybírá se z celé skupiny.

4.2. Z elementů z kroku 4.1. se vyhledají prvky navazující/nenavazující[2] tvarem. Hledání návaznosti tvaru je stejné jako v kroku 4.1..

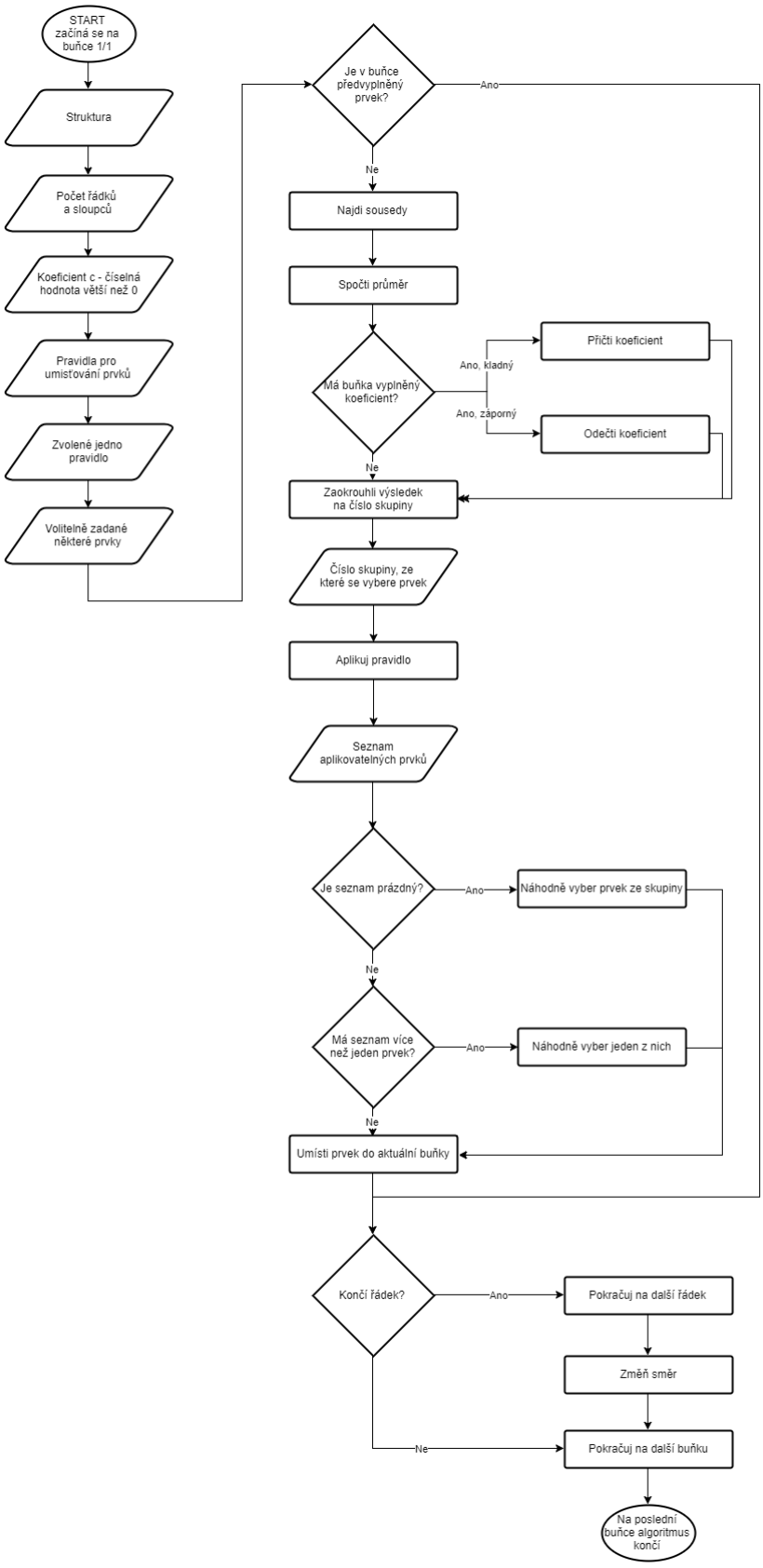
Pokud neodpovídá žádný element, vybírá se z celé skupiny.

4.3. Výsledný element z kroku 4.2. se uloží do aktuální buňky. Pokud je výsledkem kroku 4.2. více než jeden element – algoritmus náhodně vybere jeden z nich a ten se uloží do aktuální buňky.

5. Algoritmus pokračuje na další buňku.

6. Algoritmus končí na poslední buňce.

Podtržené položky naznačují data, se kterými algoritmus pracuje.



Obrázek 15: Vývojový diagram algoritmu pro umisťování elementů. Zdroj: vlastní.

2.3.1 Příklad činnosti algoritmu

Jako příklad uvedu buňku [1,9], kterou uvádí Sýkora v článku *Computer-aided multielement geometrical abstract paintings* [6]. Program má nyní k dispozici následující položky:

- zadání od autora:
  - velikost mřížky,
  - umístěné elementy v některých buňkách, plusy nebo minusy,
  - koeficient přechodu c = 0,75,
  - zvolené pravidlo V=2 (nepokračuje barva, pokračuje tvar),
- vyplněné buňky [1,1], [1,2], [1,3], [1,4], [1,5], [1,6], [1,7] a [1,8].

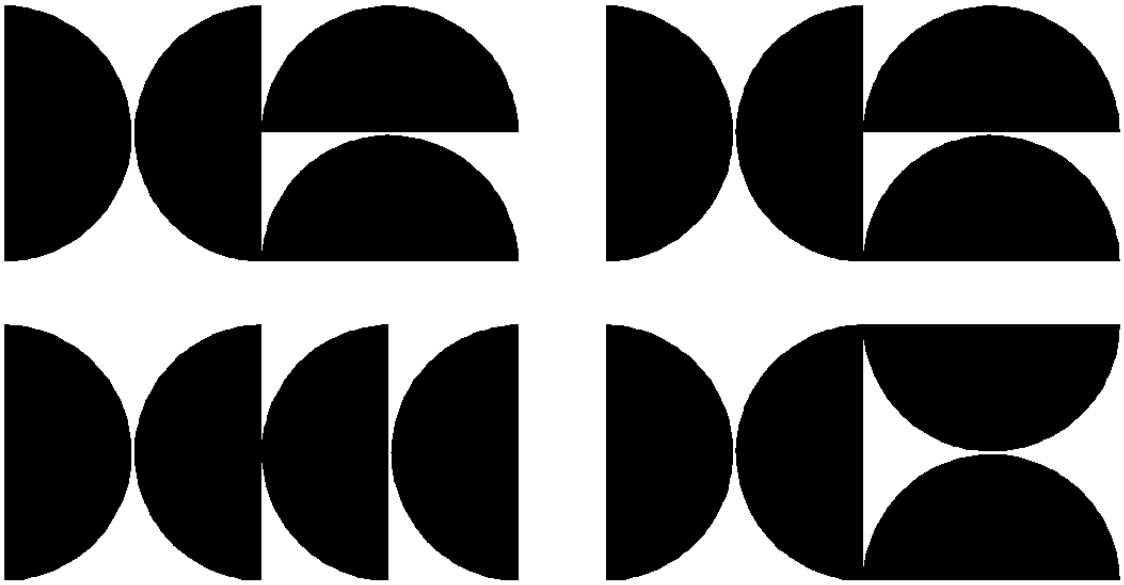
Aktuální stav mřížky ukazuje tabulka 3. Bod 3 z algoritmu – hledání sousedů. V tabulce 3 je aktuální vyhodnocovaná buňka označena červeně a prohledávané okolí zeleně.

Algoritmus v tomto bodě nalezl v sousedních buňkách elementy - na západě **4r** a na jiho-západě **4z**. Následuje bod 3.1. výpočet průměru: $\frac{4+4}{2} = 4$. V bodě 3.2. algoritmus vyhodnocuje, zda buňka obsahuje „+“ nebo „-“. Buňka nic neobsahuje, takže se koeficient nepřičítá ani neodečítá. V bodě 3.3. algoritmus zaokrouhluje vypočtený průměr tj. 4. Element pro umístění do buňky [1,9] tedy algoritmus vybírá ze čtvrté skupiny.

Následuje aplikace pravidla V=2 (nepokračuje barva, pokračuje tvar). Jediná buňka, podle které se bude aplikovat pravidlo, je západní buňka s elementem **4r**. Algoritmus tedy nejprve hledá elementy ze skupiny 4, které nenavazují barvou k elementu **4r**. Toto splňují elementy **4i**, **4y**, **4b** a **4d**. Ilustruje to obrázek 16.

Tabulka 3: Aktuální stav mřížky při výpočtu buňky [1,9].

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	1y	1b	1r	2z	3r	1r	3y	4r		-	-
2	-	1r	+	+	-	-	-	4z	+	-	2r
3	-	+	+	4i	-	1i	-				
4	-	+	+	+	-	-	-	3b	+	+	
5	1z				-	1d	-	+	+		-
6	-	+	+		-	-	1y	+	+	+	1d
7	-	+	4z	+	1r	-	-		+		-
8		+	+	+		-	-	3b		-	-
9	+	+	+	1d		-	-	+	+	4r	-
10	+		3b	+			1r	+	+		-
11	1z	-	-	-	1b	1z		-	-	-	1i
12	1d		1d				1d	-	-	-	1b
13		4z	-	-	1d				+	+	4r
14							1y		+	+	
15	+	+		1b					4z		
16	2r			+	+	1r		+	+		-
17	-		4y		+			+	+		-
18	-			+	+		1d		3y	-	-
19	-			3r	+		+	+	+	-	1y
20	1b					1y	+	+	+		-
21	-		+		-	-	+	+	4i		-
22	-		3y		+	1r		+	+	+	

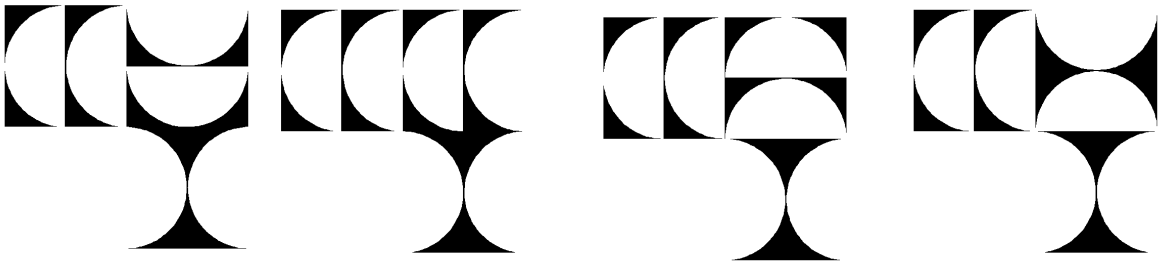


Obrázek 16: Barevně navazující prvky k **4r** dle pravidla V=2. Zdroj: vlastní.

Z těchto elementů hledá algortimus takové, které pokračují tvarem. Žádný takový element není a tak algoritmus náhodně vybírá z elementů z předchozího kroku. V případě zadání dle článku [6] vybral algoritmus element **4i**. Algoritmus pokračuje na buňku [1,10].

2.3.2 Příklady aplikace pravidel

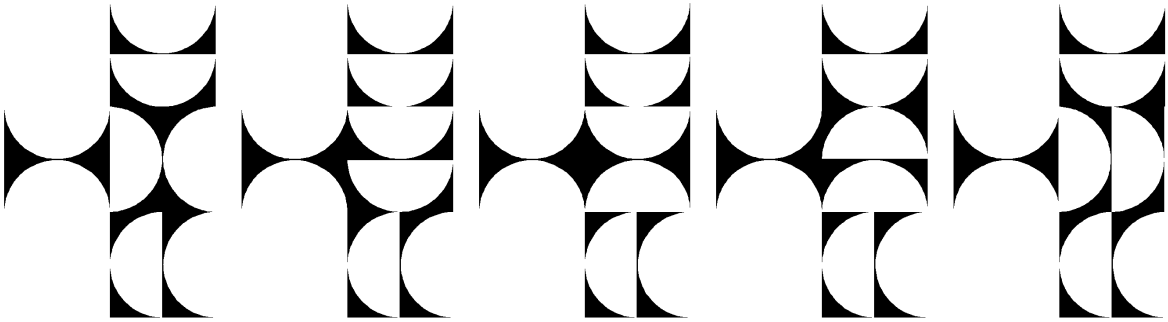
Buňka [1,2] dle zadání ze Sýkorova článku [6]:
Podle elementů v okolních buňkách vybírá algoritmus element/y z první skupiny. V prvním kroku hledá algoritmus elementy, které nenavazují barvou. Ze západu (prvek **1y**) splňují tuto podmínky prvky **1i**, **1y**, **1b** a **1d**. Ovšem ze zadání je definován element i v buňce [2,2] (**1r**) a proto je nutné vzít v potaz i jej. To znamená, že prvky **1i** a **1y** toto pravidlo nesplňují. Naopak elementy **1b** a **1d** splňují nepokračování barvou na západní i na jižní straně. Algoritmus tedy vybírá z těchto dvou. V případě zadání dle článku [6] vybral algoritmus element **1b**.



Obrázek 17: Ilustrace k příkladu buňky [1,2]. Zdroj: vlastní.

Buňka [5,7] dle zadání ze Sýkorova článku [6]:

Zde lze ilustrovat splnění pravidla „alespoň na jedné straně“. Dle okolních buněk vybírá algoritmus ze skupiny 1. Buňka má sousedy na severu (**1i**), na jihu (**1y**) a na západě (**1d**). Dle zvoleného pravidla V = 2 (nenavazuje barva, navazuje tvar) hledá algoritmus prvky, které nenavazují barvou. To nelze splnit na všech třech sousedních stranách. Algoritmus tedy hledá splnění pravidla, alespoň na jedné straně. Tuto situaci ilustruje obrázek 18. Algoritmus tak vybírá z prvků **1z**, **1i**, **1b**, **1r** a **1d**. Návaznost tvarem lze splnit nejvýše na dvou stranách a splňují to výše zmíněné prvky. Množina pro výběr se tedy nemění. V případě zadání dle článku [6] vybral algoritmus element **1z**.



Obrázek 18: Ilustrace k příkladu buňky [5,7]. Zdroj: vlastní.

3 Implementace aplikace

Aplikace je naprogramována v jazyce C#. Jako cílový framework byl zvolen Microsoft .NET Framework ve verzi 4.5. V následujících kapitolách je nejprve popsána základní struktura aplikace z hlediska tříd a také jejich základní popis. Další kapitoly pak popisují implementaci některých stěžejních částí a funkcionalit aplikace.[3]

K vývoji aplikace byl použit nástroj Microsoft Visual Studio Enterprise 2017 s doplňkem JetBrains ReSharper Ultimate 2017.1.3. ReSharper, mimo jiné, usnadňuje vývoj doplňováním částí kódu a dodržování standardů jako například pojmenovávání metod, proměnných, konstant atd.

3.1 Jazyk C#

C# je programovací jazyk, určený pro vytváření aplikací, které běží na rozhraní .NET Framework. C# je jednoduchý, výkonný, typově bezpečný a objektově orientovaný. Patří mezi vyšší programovací jazyky, podobně jako Java nebo C++. Tyto jazyky jsou charakterizovány vyšší mírou abstrakce, menší závislostí na konkrétním technickém vybavení a vyšší čitelností zdrojového kódu. Jazyk C# patří mezi nejpoužívanější programovací jazyky na světě.

Platforma Microsoft .NET Framework obrahuje kromě programovacích jazyků (C#, VB.NET, F#, J# a další) ještě sadu vývojářských nástrojů a knihoven. Poskytuje také prostředí pro běh programů napsaných v jazycích z rodiny .NET. Toto prostředí se nazývá **Common Language Runtime**.

Zdrojové kódy programů napsaných v jazyce C# jsou uloženy do souborů s příponou *.cs* a kompilátorem jsou překládány do spustitelných programů s příponou *.exe*, případně do knihoven s příponou *.dll*. Pro spuštění programů musí být na počítači nainstalován .NET Framework v minimální verzi, pro kterou byla daná aplikace vytvořena. [20] [21]

[3] Uvedené zdrojové kódy jsou upraveny pro snažší čitelnost.

3.1.1 Windows forms

Windows Form (zkráceně nazýváno WinForms) je prvním frameworkem z rodiny Microsoft .NET. Umožňuje vývojáři snadnou tvorbu formulářových aplikací pomocí grafického designéru. Framework obsahuje několik připravených grafických komponent a zároveň umožňuje tvorbu vlastních.

Dalším frameworkem je modernější **Windows Presentation Foundation** (WPF). Ten umožňuje lepší oddělení logiky od grafického výstupu, rychlejší vykreslování díky použití hardwarové akcelerace, animace, tzv. bindování dat a dalších nových technologií. K definici grafické části využívá jazyk **XAML**. Díky novým technologiím a postupům je vývoj WPF aplikaci znatelně složitější. Windows Forms nejsou považovány za zastaralé a stále se využívají, především v případech, kdy připravené formulářové komponenty dostačují. Pro potřeby aplikace, která je součástí této diplomové práce, je framework Windows Form plně dostačující, a proto byl zvolen. [16]

3.2 Struktura aplikace a popis tříd

Aplikace se skládá z několik tříd, které jsou uloženy v souborech s příponou *.cs*. Zdrojové kódy jsou na přiloženém CD. Pro přehlednost jsou třídy umístěny ve složkách.

Ve složce **Forms** jsou třídy pro okna aplikace, ve složce **Helpers** jsou pomocné třídy a ve složce **Objects** jsou datové třídy definující strukturu objektů, se kterými aplikace pracuje. V kořenovém adresáři je třída *Program*. Tato třída obsahuje pouze metodu `Main`, která spouští okenní aplikaci.

Složka Forms obsahuje:

- Třídu *AboutForm.cs* – okno „O aplkaci“. Obsahuje základní informace o aplikaci a umožňuje otevřít text diplomové práce ve formátu PDF,
- třídu *BusyWindow.cs* – okno informující uživatele, že je aplikace zaneprázdněna. Toto okno je zobrazeno při načítání/ukládání souboru a při běhu algoritmu pro umisťování elementů,
- Třídu *HelpForm.cs* – okno „Nápověda“. Obsahuje informace o ovládání programu a umožňuje otevřít uživatelskou přiručku ve formátu PDF,

- třídu *MainForm.cs* – hlavní okno aplikace. Toto okno obsahuje ovládací prvky, mřížku pro umisťování elementů a náhled výsledného obrázku. Podrobněji v kapitole 3.2.1 na straně 39,
- třídu *PartitionInputWindow.cs* – okno pro zadání počtu řádků a počtu sloupců nové partitury.

Složka Helpers obsahuje:

- Třídu *Algorithm.cs*. V této třídě je implementován algoritmus pro umisťování elementů, který je popsán v kapitole 2.3 na straně 29. Detailnější popis implementace je v 3.2.5 na straně 48,
- třídu *DataService.cs*. Tato třída slouží jako úložiště dat v rámci aplikace. Podrobnější popis v 3.2.4 na straně 47,
- třídu *JsonWorker.cs*. Tato třída obsluhuje ukládání a načítání JSON souborů s definicí struktury a partiturou. Podrobější popis v 3.2.3 na straně 45.

Složka Objects obsahuje:

- Třídu *EdgeProperty.cs* definující vlastnosti elementu na konkrétní hraně,
- třídu *Element.cs* definující element,
- třídu *Partition.cs* definující partituru,
- třidu *PartitionItemcs* definující konkrétní položku partitury,
- třídu *Rule.cs* definující pravidlo pro umisťování elementů,
- třídu *Structure.cs* pro definici Struktury.

  Podrobněji o těchto třídách a jejich návaznostech hovoří kapitola 3.2.6 na straně 51.

3.2.1 Hlavní okno aplikace – *Mainform.cs*

Tato třída definuje hlavní okno aplikace. Okno obsahuje několik formulářových komponent. Všechna tlačítka jsou realizována komponentou Button [4].

[4] Platí i pro ostatní třídy.

V záhlaví aplikace je umístěna komponenta ToolStripMenu, která slouží k zobrazení nabídky. Kliknutím na první dvě položky (**Informace** a **Nápověda**) se otevřou okna s nápovědou případně informacemi o aplikaci. Otevření těchto oken zajišťují metody `napovedaToolStripMenuItem_Click` a `informaceToolStripMenuItem_Click`. Jejich kód:

```
private void napovedaToolStripMenuItem_Click(object sender, EventArgs e)
{
    new HelpForm().Show(); //otevreni okna Napoveda
}
private void informaceToolStripMenuItem_Click(object sender, EventArgs e)
{
    new AboutForm().Show(); //otevreni okna O aplikaci
}
```

Další položkou je **Načíst definici Struktury**. Kliknutím na tuto položku se otevře dialogové okno (komponenta OpenFileDialog) pro vybrání JSON souboru s definicí Struktury. Tuto položku obstarává metoda `toolStripMenuItemLoadStructureDefiniton_Click`.

Dále je v nabídce položka **Partitura**, která po sebou ukrývá další podpoložky pro práci s partiturou:

- **Nová partitura**,
- **Načíst partituru**,
- **Vykreslit Strukturu z uložené partitury**,
- **Uložit partituru**.

Kliknutím na podpoložku **Nová partitura** se otevře okno **Nová partitura** s dvěma číselníky (komponenty NumericUpDown) pro zadání počtu řádků a počtu sloupců nové partitury a dvěma tlačítky (**Nová partitura** a **Zrušit**). Kliknutí na tlačítko **Nová partitura** obstarává metoda `buttonNewPartition_Click`. Tato metoda volá metodu `NewPartiton` z třídy *Mainform*, která vykreslí mřížku. Kliknutím na tlačítko **Zrušit** nebo stisknutím klávesy *Esc* se okno zavře.

Podpoložky **Načíst partituru**, **Vykreslit Strukturu z uložené partitury** a **Uložit partituru** slouží k načítání/ukládání partitury. Kliknutí na tyto podpoložky obsluhují

metody `toolStripMenuItemLoadPartition_Click`, `toolStripMenuItemDrawStructureFromSavedPartition_Click` a `toolStripMenuItemSavePartition_Click`. Načítací metody otevírají dialogové okno pro vybrání souboru se strukturou (komponenta OpenFileDialog). Ukládací metoda otevírá dialogové okno pro vybrání názvu a umístění JSON souboru s partiturou (komponenta SaveFileDialog. Metoda `toolStripMenuItemDrawStructureFromSavedPartition_Click` zkontroluje načtenou partituru, jestli je kompletní. Pokud ano, vykreslí z ní výsledný obrázek. Pokud partitura kompletní není, informuje o tom uživatele s dotazem, zda chce znovu načíst soubor. Využívá k tomu komponentu MessageBox s příznakem **MessageBoxButtons.YesNo**. Zmíněné metody pro ukládání a načítání JSON souborů využívají metod z třídy *JsonWorker.cs*. Jejich podrobnější popis je v 3.2.3 na straně 45.

Dále okno obsahuje komponentu FlowLayoutPanel, která slouží k automatickému uspořádání formulářových komponent. Do něj je umístěna komponenta PictureBox, která slouží pro vykreslení obrázku. FlowLayoutPanel má nastavenou vlastnost **AutoScroll** na hodnotu **True**, což zajistí, že pokud obrázek bude větších rozměrů, než je velikost panelu, zobrazí se automaticky posuvníky. Do PictureBoxu je vykreslena mřížka, která slouží pro umisťování elementů uživatelem před spuštěním algoritmu. Případně jsou sem umístěny elementy a znaménka načtená ze souboru. Po konci algoritmu jsou do mřížky doplněny chybějící elementy.

Mřížku vykresluje metoda `DrawEnvironment`, která je volána při inicializaci hlavního okna a kliknutím na podpoložku v nabídce v záhlaví aplikace **Nová partitura**. Tato metoda volá metodu `DrawGrid`, která vypadá následovně:

```
private void DrawGrid()
{
    var columnsCount = DataService.Instance.GetPartition().ColumnsCount;
    var rowsCount = DataService.Instance.GetPartition().RowsCount;
    pictureBoxMesh.Width = (int)MeshItemSize * columnsCount;
    pictureBoxMesh.Height = (int)MeshItemSize * rowsCount;
    Bitmap bitmap = new Bitmap(pictureBoxMesh.Width, pictureBoxMesh.Height);
    Graphics g = Graphics.FromImage(bitmap);
    var pen = new Pen(Color.Black, 1);
    for (int i = 1; i <= columnsCount; i++)
```

```
    {
        g.DrawLine(pen, MeshItemSize * i, 0, MeshItemSize * i,
            pictureBoxMesh.Height);
    }

    for (int i = 1; i <= rowsCount; i++)
    {
        g.DrawLine(pen, 0, MeshItemSize * i, pictureBoxMesh.Width, MeshItemSize *
            i);
    }
    pictureBoxMesh.Image = bitmap;
}
```

K vložení hodnot do mřížky se využívá kliknutí myši. Tuto událost obstarává metoda `pictureBoxMesh_MouseClick`. Její kód:

```
private void pictureBoxMesh_MouseClick(object sender, MouseEventArgs e)
{
    int row = (int)Math.Ceiling(e.Y / MeshItemSize);
    int column = (int)Math.Ceiling(e.X / MeshItemSize);
    DataService.Instance.SetClickedCellCoordination(row, column);
    if ((e.Button & MouseButtons.Left) != 0)
    {
         var itemToDraw = GetRandomItem();
         DataService.Instance.UpdatePartitionItem(row, column, itemToDraw);
         DrawToMesh(row, column, itemToDraw);
    }
}
```

Metoda má jako vstupní argument **MouseEventArgs**, díky čemuž jsou k dispozici souřadnice kurzoru. Toho se využívá pro identifikaci buňky v mřížce. Pokud uživatel klikne levým tlačítkem myši, program náhodně vybere element nebo znaménko. Pokud uživatel klikne pravým tlačítkem, zobrazí se kontextová nabídka.

O vytvoření kontextové nabídky se stará metoda `SetupContextMenu`, která je volána při inicializaci okna a načtení definice Struktury. Metoda do kontextové nabídky přidá: prázdnou položku pro vyprázdnění buňky, plus, minus a všechny dostupné elementy,

respektive jejich názvy, protože metoda obsluhující kontextovou nabídku pracuje s řetězci. Kontextová nabídka je složena z komponent MenuItem. Každá položka této nabídky ma specifikovaný Tag. Ten slouží k předávání pole objektů jako parametr. Aplikace toho využívá pro předání řetězce, jímž je položka v kontextové nabídce definována.

Kliknutí na položku kontextové nabídky obstará metoda `MeshContextMenuItemClick`. Obě zmíněné metody uloží informaci o změně hodnoty buňky a zároveň změnu vykreslí do mřížky. Vykreslení zajišťuje metoda `DrawToMesh`, která vypadá následovně:

```
private void DrawToMesh(int row, int column, string itemToDraw)
{
   Image b = new Bitmap(pictureBoxMesh.Image);
   Graphics g = Graphics.FromImage(b);
   Point cellPoint = GetCellXY(row, column);
   g.FillRectangle(Brushes.White, cellPoint.X + 1, cellPoint.Y + 1, MeshItemSize -
       1, MeshItemSize - 1);
   g.DrawString(itemToDraw, new Font(FontFamily.GenericSansSerif, 17),
       Brushes.Black, cellPoint.X, cellPoint.Y);
   pictureBoxMesh.Image = b;
}
```

Nad mřížkou je umístěn popisek (komponenta Label), který informuje uživatele o pozici kurzoru v mřížce. Změnu textu v popisku obstarává metoda `pictureBoxMesh_MouseMove` a vymazání textu při opuštění mřížky, zajišťuje metoda `pictureBoxMesh_MouseLeave`. Jejich kód:

```
private void pictureBoxMesh_MouseMove(object sender, MouseEventArgs e)
{
    int row = (int)Math.Ceiling(e.Y / MeshItemSize);
    int column = (int)Math.Ceiling(e.X / MeshItemSize);
    labelClickPosition.Text = "Pozice kurzoru v mrizce: " + row + ". radek, " +
        column + ". sloupec";
}
private void pictureBoxMesh_MouseLeave(object sender, EventArgs e)
{
    labelClickPosition.Text = "";
}
```

Tlačítko **Spustit algoritmus** obstarává metoda `buttonStartAlgorithm_Click`. Pokud je načtena definice Struktury, zavolá metodu `StartAlgorithm`. Pokud již algoritmus proběhl, informuje o tom uživatele dialogovým oknem s tlačítky Ano/Ne (komponenta MessageBox s příznakem **MessageBoxButtons.YesNo**) a s dotazem, zda chce i přesto obraz překreslit. Pokud uživatel klikne ne tlačítko Ne, je dotázám dalším dialogovým oknem, zda chce načíst soubor s partiturou. Pokud není načtena definice Struktury, aplikace o tom uživatele informuje a vyzve jej k načtení definice Struktury.

Metoda `StartAlgorithm` uloží hodnoty z rolovací nabídky **Zvolené pravidlo** (komponenta ComboBox) a z číselníku **Koeficient** (komponenta NumericUpDown), spustí algoritmus umisťování elementů, následně vykreslí výsledný obrázek a doplní do mřížky chybějící elementy. Kód metody:

```
private void StartAlgorithm()
{
   DataService.Instance.SetCoefficient((double)numericUpDownCoefficient.Value);
   DataService.Instance.SetSelectedRule((int)comboBoxRules.SelectedValue);
   Algorithm.Start();
   DrawPicture();
   DrawPartition();
}
```

Třída *Algorithm* je statická, tzn. že pro volání jejich metod není potřeba vytvářet instanci třídy. Popis metody `Start` z této třídy je v 3.2.5 na straně 48.

Vykreslení výsledného obrázku zajišťuje metoda `DrawPicture`. Výsledný obrázek uloží do adresáře, ze kterého je spuštěna aplikace. Uložen je ve formátu PNG, který je určen pro bezeztrátovou kompresi rastrové grafiky.

Metoda `DrawPartition` doplní do mřížky elementy dle výsledku činnosti algoritmu. Tato metoda využívá již dříve zmíněné metody `DrawToMesh` [5].

Pod tlačítky je umístěn ještě jeden PictureBox, který slouží pro zobrazení náhledu výsledného obrázku. Tento obrázek se zobrazí po dokončení činnosti algoritmu. Pro zajištění zobrazení celého obrázku, má komponenta nastavenu vlastnost **SizeMode** na hodnotu **PictureBoxSizeMode.Zoom**. Dvojklikem na náhled se otevře výsledný obrázek v plně

[5] Kód metody je na straně 43.

kvalitě. To zajišťuje metoda `pictureBoxPicture_DoubleClick`.

Operace jako načítání, ukládání, spuštění algoritmu, vykreslení mřížky či vykreslení obrázku mohou být časově náročné. Z toho důvodu jsou všechny operace v metodách obstarávajících tyto činnosti, „obaleny“ kódem pro otevření okna signalizující činnost aplikace. Zkrácená ukázka kódu:

```
BusyWindow busyWindow = new BusyWindow();
busyWindow.Show();
busyWindow.Update();
... //kod metody
busyWindow.Close();
```

3.2.2 Ostatní okna aplikace

Okno **Nápověda** má nadpis (komponenta Label), textové pole (komponenta RichTextBox) a tlačítko. Textové pole má nastavenou vlastnost **ReadOnly** na hodnotu **True**, díky čemuž není možné text editovat a komponenta se tak chová jako zobrazovač textu. Dále je zde tlačítko **Zobrazit uživatelskou příručku**, které otevírá uživatelskou příručku z kapitoly 4 ve formátu PDF.

Okno **Informace** je struktorováno stejně. Pouze má jiné tlačítko a to **Zobrazit diplomovou práci**, které otevírá celý text této práce ve formátu PDF.

Okno signalizující činnost aplikace má pouze komponentu Label s textem „Pracuji.“. Okno má nastavenou vlastnost **ControlBox** na hodnotu **False**, díky čemuž nejsou zobrazeny ovládací prvky pro minimalizování, maximaliování a zavření okna. Uživatel tak nemůže okno zavřít a musí čekat na dokončení operace, po jejímž dokončení je okno zavřeno aplikací.

3.2.3 Práce s JSON soubory – *JsonWorker.cs*

Tato třída obstarává načítání a ukládání JSON souborů. Pro implementaci těchto metod je použita knihovna *NewtonSoft.Json*[14]. Tato knihovna umožňuje serializaci objektu do JSON souboru a deserializaci ze souboru zpět do objektu. Metoda pro uložení partitury využívá serializace objektu do souboru. Vypadá následovně:

```
public static void SavePartition(Partition partition, string fileName)
{
   string json = JsonConvert.SerializeObject(partition); //serialiazce objektu
   using (StreamWriter sw = new StreamWriter(fileName))
   {
      sw.Write(json);
   }
}
```

Metoda pro načítání využívá deserializace JSON souboru do objektu. Výsledek deserializace (metoda `JsonConvert.DeserializeObject()`) se uloží do dynamicky typované proměnné (*dynamic*[18]). Ve skutečnosti se jedná o proměnnou typu *object*, ovšem s typovou kontrolou odloženou do doby provádění programu[19] . Využívá se toho při získávání informací z JSON souboru. Následující zkrácená ukázka kódu metody `LoadPartition` ukazuje přístup k informacím v JSON souboru:

```
public static Partition LoadPartition(string filePath) {
   using (StreamReader streamReader = new StreamReader(filePath)) {
    string json = streamReader.ReadToEnd(); //nacteni souboru
    dynamic fromJson = JsonConvert.DeserializeObject(json); //deserializace do
        objektu
    int rowsCount = fromJson.RowsCount; //pristup k informaci z JSON souboru pres
        nazev polozky
    int columnsCount = fromJson.ColumnsCount;
    double coefficientValue = fromJson.Coefficient;
    int selectedRule = fromJson.SelectedRule;
    var items = new List<PartitonItem>();
    int itemsCount = fromJson.Items.Count;
    for (int i = 0; i < itemsCount; i++) {
      int row = fromJson.Items[i].Row; //pristup k informaci v kolekci polozek
      int column = fromJson.Items[i].Column;
      string elementName = fromJson.Items[i].ElementName;
      PartitonItem.Coefficients coefficient = fromJson.Items[i].Coefficient;
      ...
}
```

Při použití dynamicky typované proměnné Visual Studio neví, o jaký typ se jedná, a proto nenapovídá názvy proměnných. Pro načtení hodnot musí názvy odpovídat položkám v JSON souboru.

3.2.4 Třída pro práci s daty – *DataService.cs*

Tato třída, nazveme ji datová služba, slouží k centralizaci dat ve spuštěné aplikaci. Jsou sem např. uložena data načtená z JSON souborů, algoritmus pro umisťování elementů pracuje nad touto třídou a jsou zde uloženy i další pomocné informace jako např. informace o buňce, na kterou uživatel klikl, či seznam dostupných element, který aplikace načítá do kontextové nabídky.

Třída je realizována návrhovým vzorem *Singleton* – česky se nazývá *Jedináček*. Implementace tohoto návrhového vzoru zajistí, že třída má v aplikaci pouze jednu instanci, což je pro datovou službu žádoucí. Jedinou instanci třídy zajistí použití privátního konstruktoru [15].

Třída má několik atributů, které jsou privání a přístup k nim zvenčí zajišťují metody. Tyto metody se nazývají gettery (metody pro získání informace, z ang. get – získat) a settery (metody pro nastavení hodnoty, z ang. set – nastavit).

Ukázka kódu (privátní konstruktor a atribut Instance s getterem):

```
private DataService(){} //privatni konstruktor
public static DataService Instance
{
   get { //getter vraci instaci - pokud neexistuje, vytvori ji
      if (_instance == null) {
            _instance = new DataService();
         }
            return _instance;
   }
}
```

3.2.5 Algoritmus umisťování elementů – *Algorithm.cs*

Tato třída implementuje algoritmus umisťování elementů popsaný v kapitole 2.3 na straně 29. Algoritmus spouští metoda `Start`. Tato metoda prochází položky partitury stylem liché řádky zleva doprava a sudé řádky zprava doleva. Pro každou položku partitury volá metodu `PutElement`, která umístí element do partitury, pokud tam již není.

Získání elementu pro umístění do partitury předchází zjištění vhodných elementů metodou `FindItemsTofit`, z nichž se metodou `GetRandomElement` jeden vybere. Metoda `FindItemsToFit` má dva vstupní parametry – seznam krajních vlastností sousedících elementů na příslušných stranách a číslo skupiny, ze které se bude vybírat element pro umistění do partitury.

Metoda`FindNeighboursElementsEdgeProperties` zjišťuje krajní vlastnosti sousedících elementů. Nejprve zjistí, zda má zkoumaná buňka souseda na severu, východě, jihu a západě. Poté pro každého existujícího souseda najde jeho krajní vlastnosti na straně, která je důležitá pro aplikaci pravidel. Kombinace jsou v tabulce 4.

Tabulka 4: Zkoumané strany pro sousední elementy

Soused	**Zkoumaná strana**
Severní	Jižní
Východní	Západní
Jižní	Severní
Západní	Východní

Číslo skupiny, ze které se bude vybírat element pro umistění do partitury, vrací metoda `GetGroupNumberToSelectElement`. Ta nejprve zavolá metodu `GetAverage`. Tato metoda volá `CountAverage`, která počítá průměr ze skupin sousedních elementů. Seznam čísel sousedních elementů (List<int>) vrací metoda `FindNeighboursGroups`, která prohledá okolí zkoumané buňky (tato metoda nehledá sousedy jen na severu, východě, jihu a západě, ale prohledává všech 8 okolních buněk) a zjišťuje čísla skupin elementů v obsazených buňkách. Metoda `GetAverage` dále volá metodu `CheckCoefficient`, která zjistí, zda je v aktuálně vyhodnocované buňce koeficient. Na základě toho vrací hodnotu koeficientu nebo zápornou hodnotu koeficientu nebo 0. Tento výsledek se připočte k průměru zjiště-

ném metodou `CountAverage`. Pokud v tomto momentě končí průměr 5 desetinami (př. 1,5), hledají se čísla skupin v širším okolí. To zajišťuje opět metoda `GetAverage`. Tato metoda má totiž jako vstupní parametr kromě čísla řádku a čísla sloupce i úroveň hledání. Opět se opakuje aplikace koeficientu a vyhodnocení, zda průměr nekončí 5 desetinami. Finální číslo skupiny, ze které se bude vybírat element pro umístění do partitury obstará metoda `RoundResult`. Tato metoda zaokrouhlí průměr na celé číslo. Pokud je průměr menší než 1, vrací 1. Pokud je průměr větší než číslo poslední skupiny, vrací číslo poslední skupiny.

Nyní má metoda `FindItemsToFit` potřebné informace k nalezení skupiny elementů vhodných k umístění do partitury. Metoda ze zvoleného pravidla zjistí, zda má hledat elementy navazující nebo nenavazující podle barvy a tvaru. Dále zjistí, jaké má k dispozici sousedy. Na základě těchto informací hledá vhodné elementy. K hledání využívá metodu `FindAll`[23], která vrací seznam prvků splňujících podmínku. Jak již bylo zmíněno v popisu algoritmu pro umisťování elementů v kapitole 2.3, hledají se nejprve elementy, které splňují pravidlo na všech možných stranách, a pokud žádný takový element nevyhovuje, hledají se elementy, které splňují pravidlo alespoň na jedné straně. V metodě `FindItemsTofit` je to realizováno následujícícm způsobem (zkrácená ukázka pro hledání elementů nenavazujících barvou, sousedi jsou na všech světových stranách):

```
elementsByColor = DataService.Instance.GetStructure().Elements.FindAll(
         x => x.Group == group
         && x.North.Color != north.Color
         && x.East.Color != east.Color
         && x.South.Color != south.Color
         && x.West.Color != west.Color);
         if (elementsByColor.Count == 0)  {
               elementsByColor =
                   DataService.Instance.GetStructure().Elements.FindAll(
               x => x.Group == group && (
               x.North.Color != north.Color
               || x.East.Color != east.Color
               || x.South.Color != south.Color
               || x.West.Color != west.Color)); }
```

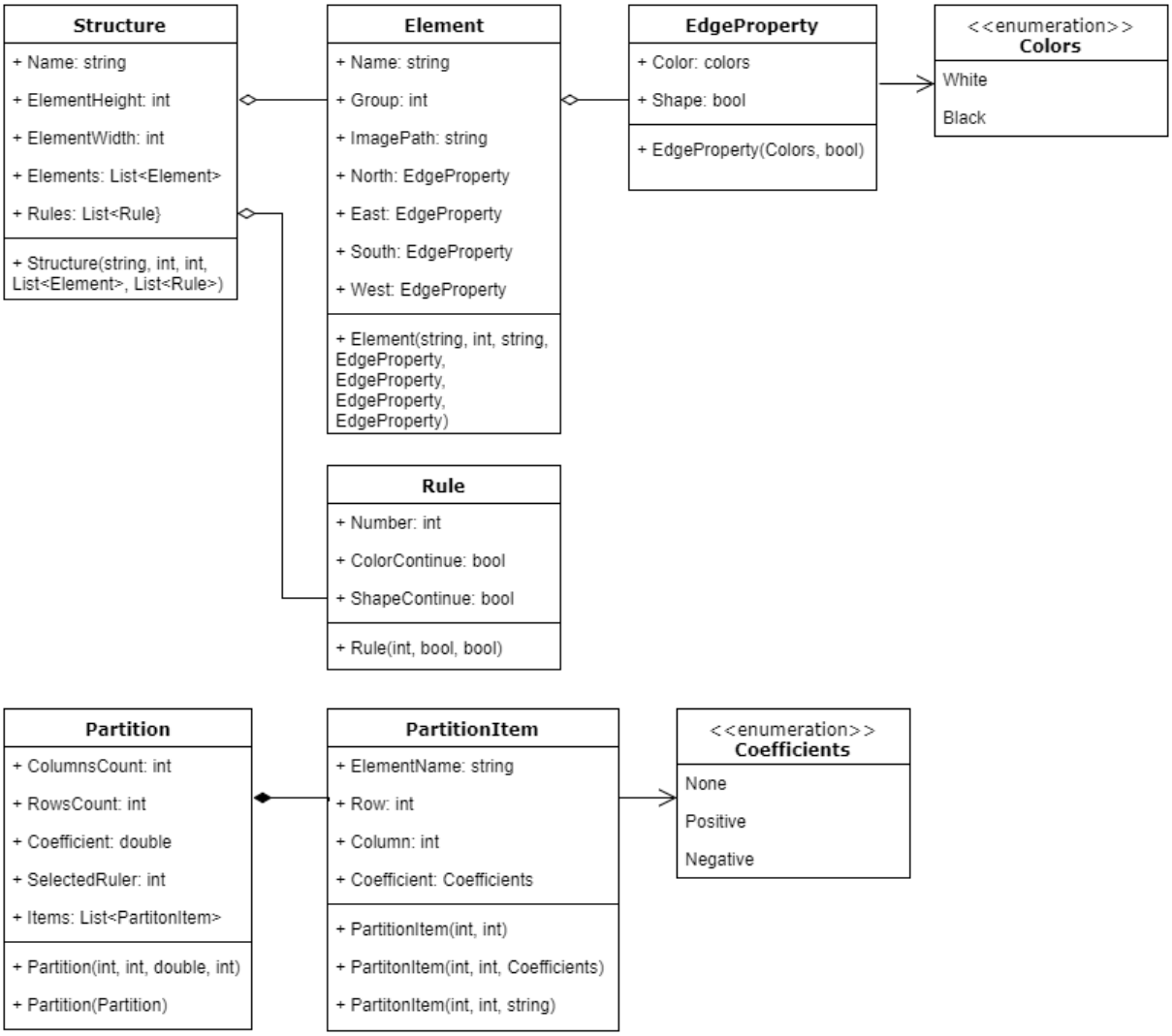
Protože má pokračování či nepokračování barvy přednost před tvarem, udržuje si metoda jeden seznam pro elementy pokračující barvou (**elementsByColor**) a druhý pro elementy pokračující tvarem (**elementsByShape**). Po aplikaci pravidel metoda projde seznam **elementsByShape** a zjišťuje, zda se elementy nachází v seznamu **elementsByColor**. V případě shody jsou elementy přidání do finálního seznamu vhodných elementů. V případě že seznam **elementsByShape** neobsahuje žádné prvky, stává se finálním seznamem seznam **elementsByColor**. V případě že i tento seznam neobsahuje žádné prvky, je výsledným seznamem celá skupina elementů. Ta je výsledkem i v případě, že aktuálně vyhodnocovaná buňka nemá žádné sousedy.

Následuje již změněná metoda `GetRandomElement`, která náhodně vybere jeden ze skupiny vhodných elementů a ten metoda `PutElement` uloží do partitury. Pro výběr náhodného prvku je použita vestavěná třída *Random* a její metoda `Next`. Parametrem této metody je maximální hodnota, která může byt vrácenna. V tomto případě je maximální hodnotou počet elementů vhodných k umístění do partitury. Kód metody:

```
private static string GetRandomElement(List<Element> elements)
   {
      Random rnd = new Random();
      var element = elements[rnd.Next(elements.Count)].Name;
      return element;
   }
```

3.2.6 Datové třídy aplikace

Aplikace pracuje v zásadě s dvěma hlavními objekty – definice Struktury (třída *Structure* a partitura (třída *Partition*). Ty jsou složeny z dalších objektů. K definici těchto objektů slouží datové třídy. Jejich strukturu, definici a vazby mezi nimi zobrazuje diagram tříd (obrázek 19). Třídy obsahují parametrické konstruktory a nemají žádné další metody.



Obrázek 19: Diagram datových tříd. Zdroj: vlastní.

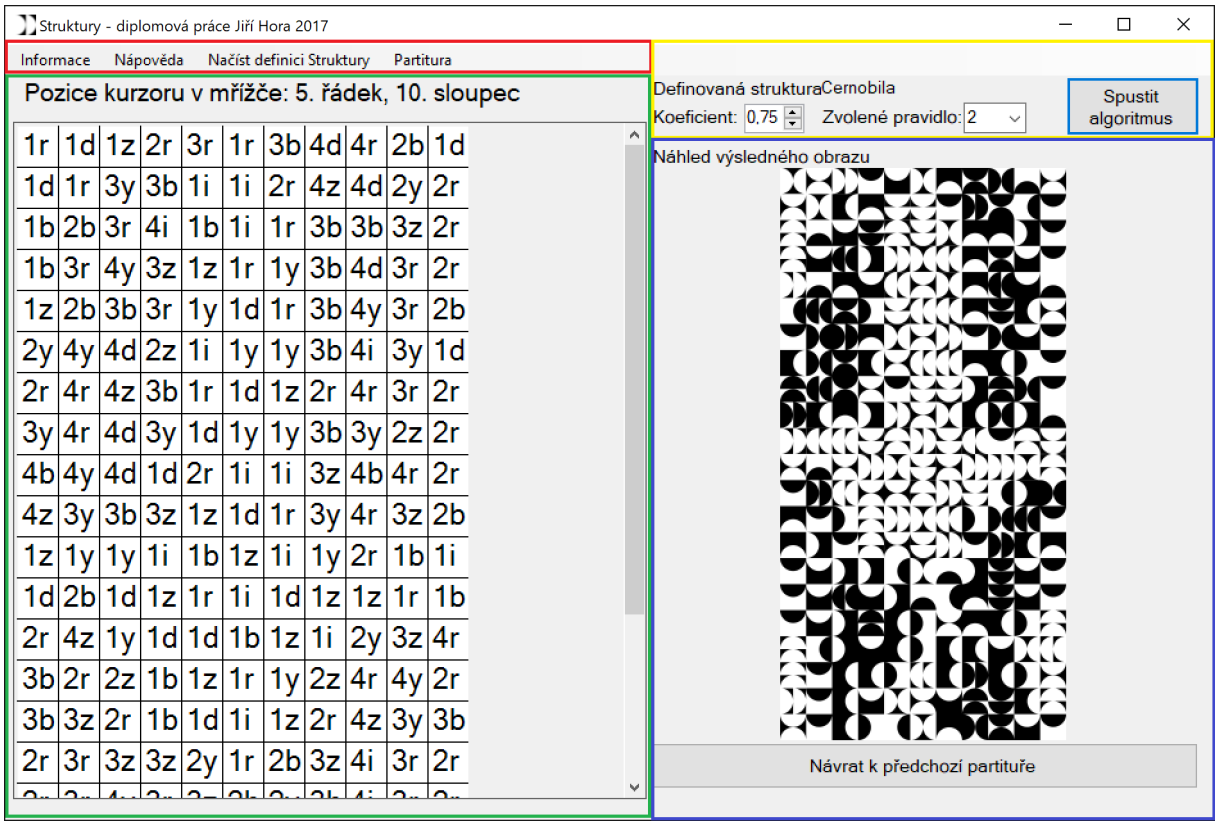
4 Uživatelská příručka

Aplikace umožňuje vytváření strukturálních obrazů podle Zdeňka Sýkory, tak jak je popsal v článku *Computer-aided multielement geometrical abstract paintings* v časopise *Leonardo 3*[6]. Uživatelská příručka popisuje ovládání aplikace.

Aplikaci lze spustit na systému Microsoft Windows 7 a novějších verzích. Ke své činnosti vyžaduje nainstalovaný framework .NET ve verzi 4.5 nebo novější. Pro správné fungování vyžaduje rozlišení obrazovky alespoň fullHD (1920 x 1080).

4.1 Uživatelské rozhraní

Hlavní okno aplikace je rozděleno do 4 částí.



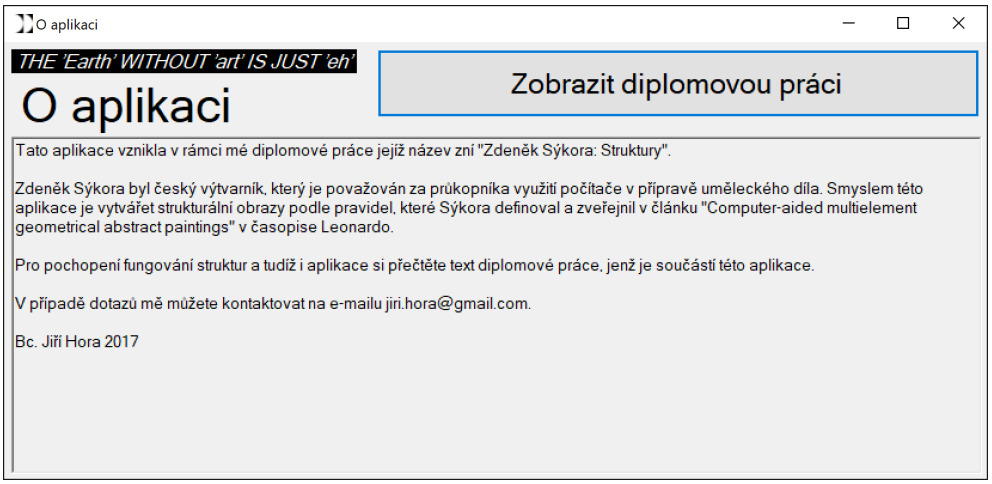
Obrázek 20: Hlavní okno uživatelského rozhraní – 4 části aplikace. Zdroj: vlastní.

4.1.1 Nabídka v záhlaví aplikace

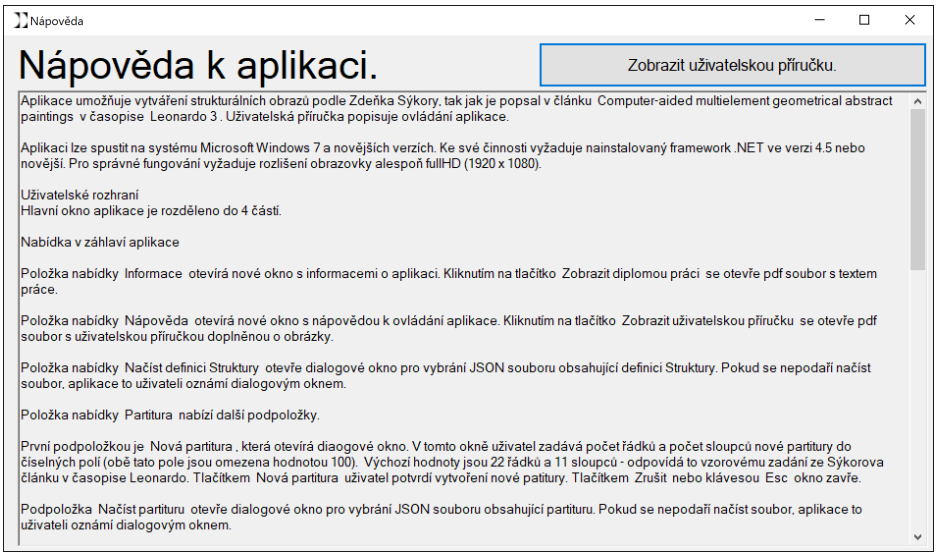
Oblast v obrázku 20 označená červeně

Položka nabídky **Informace** otevírá nové okno s informacemi o aplikaci. Kliknutím na tlačítko **Zobrazit diplomou práci** se otevře pdf soubor s textem práce.

Položka nabídky **Nápověda** otevírá nové okno s nápovědou k ovládání aplikace. Kliknutím na tlačítko **Zobrazit uživatelskou příručku** se otevře pdf soubor s uživatelskou příručkou doplněnou o obrázky.



Obrázek 21: Okno O aplikaci. Zdroj: vlastní.



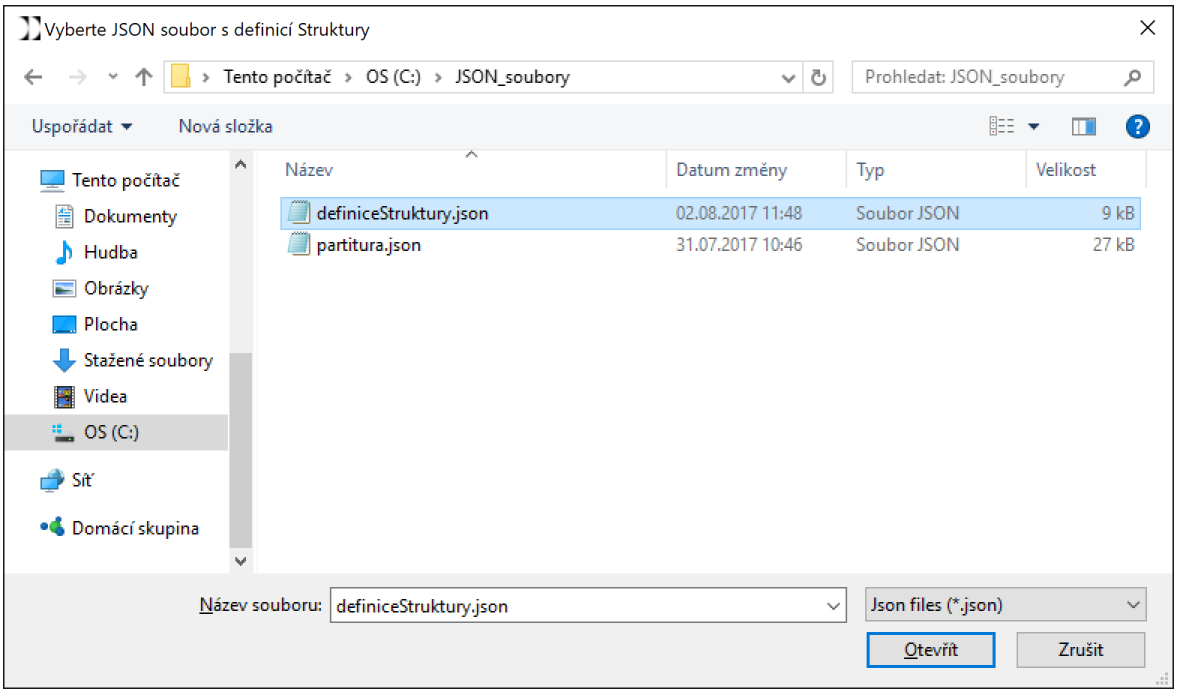
Obrázek 22: Okno Nápověda. Zdroj: vlastní.

Položka nabídky **Načíst definici Struktury** otevře dialogové okno (obrázek 23) pro vybrání JSON souboru obsahující definici Struktury. Pokud se nepodaří načíst soubor, aplikace to uživateli oznámí dialogovým oknem (obrázek 24).

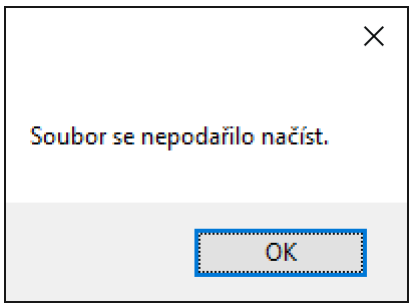
Ukázka JSON souboru definujícího Strukturu. Soubor obsahuje definici jednoho elementu a jednoho pravidla, aby byla zřejmá očekávaná struktura.


```
{
  "Name": "Cernobila",
  "ElementHeight": 500,
  "ElementWidth": 500,
  "Elements": [
    {
      "Name": "1z",
      "ImagePath": "./cernobilaElements500/1z.png",
      "Group": 1,
      "North": {
        "Color": "Black",
        "Shape": false
      },
      "East": {
        "Color": "Black",
        "Shape": false
      },
      "South": {
        "Color": "Black",
        "Shape": false
      },
      "West": {
        "Color": "White",
        "Shape": true
      }
    }
  ],
  "Rules": [
    {
```

```
      "Number": 0,
      "ColorContinue": true,
      "ShapeContinue": true
    }
  ]
}
```

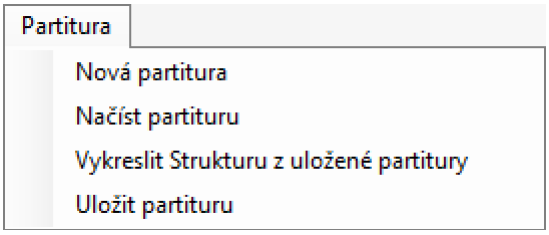


Obrázek 23: Dialogové okno pro načtení definice Struktury. Zdroj: vlastní.



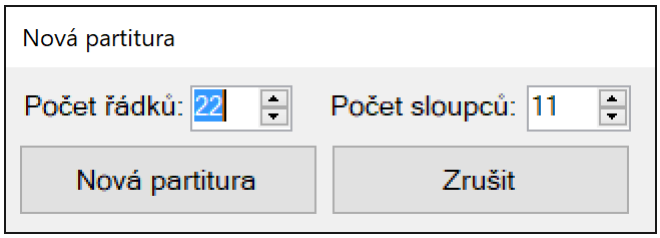
Obrázek 24: Dialogové okno oznamující chybu při načítání souboru. Zdroj: vlastní.

Položka nabídky **Partitura** nabízí další podpoložky (obrázek 25).



Obrázek 25: Možnosti práce s partiturou. Zdroj: vlastní.

První podpoložkou je **Nová partitura**, která otevírá diaogové okno (obrázek 26). V tomto okně uživatel zadává počet řádků a počet sloupců nové partitury do číselných polí (obě tato pole jsou omezena hodnotou 100). Výchozí hodnoty jsou 22 řádků a 11 sloupců – odpovídá to vzorovému zadání ze Sýkorova článku v časopise Leonardo [6]. Tlačítkem **Nová partitura** uživatel potvrdí vytvoření nové patitury. Tlačítkem **Zrušit** nebo klávesou *Esc* okno zavře.



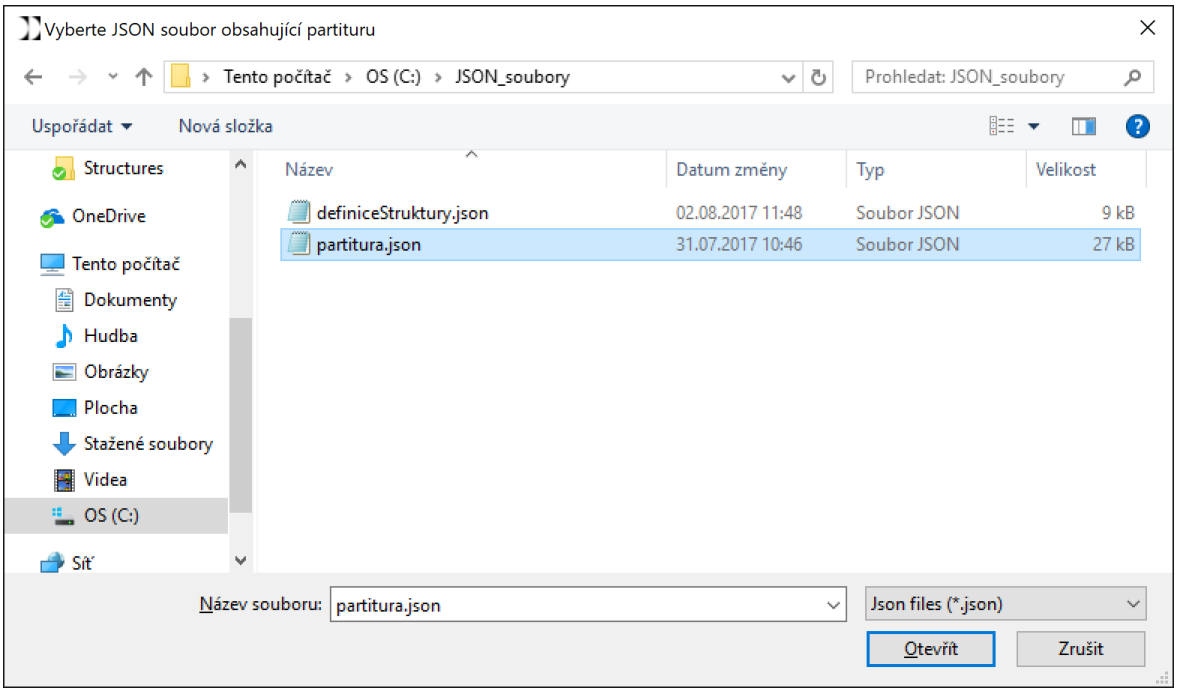
Obrázek 26: Okno pro zadání rozměrů nové partitury. Zdroj: vlastní.

Podpoložka **Načíst partituru** otevře dialogové okno (obrázek 27) pro vybrání JSON souboru obsahující partituru. Pokud se nepodaří načíst soubor, aplikace to uživateli oznámí dialogovým oknem (obrázek 24).

Ukázka JSON souboru definujícího partituru. Soubor obsahuje 3 prvky, přičemž první má záporný koeficient, druhý má kladný koeficient a třetí má element **4d**.

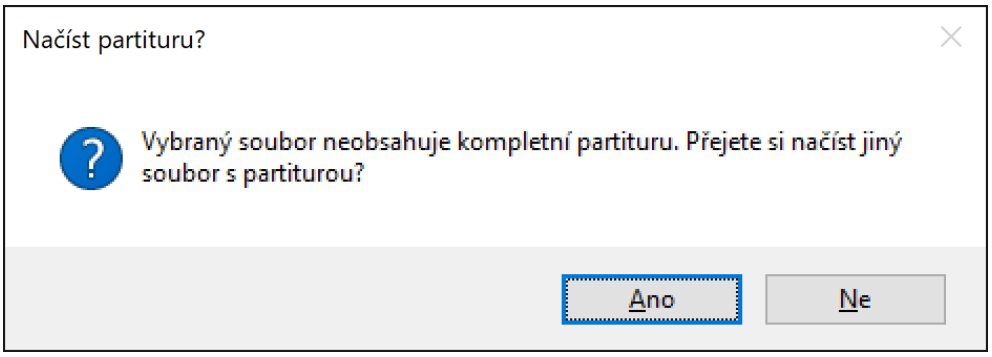
```
{
  "ColumnsCount": 3,
  "RowsCount": 1,
  "Coefficient": 0.75,
  "SelectedRule": 2,
  "Items": [
```

```
    {
      "ElementName": null,
      "Row": 1,
      "Column": 1,
      "Coefficient": "Negative"
    },
    {
      "ElementName": null,
      "Row": 1,
      "Column": 2,
      "Coefficient": "Positive"
    },
    {
      "ElementName": "4d",
      "Row": 1,
      "Column": 3,
      "Coefficient": "None"
    }
  ]
}
```



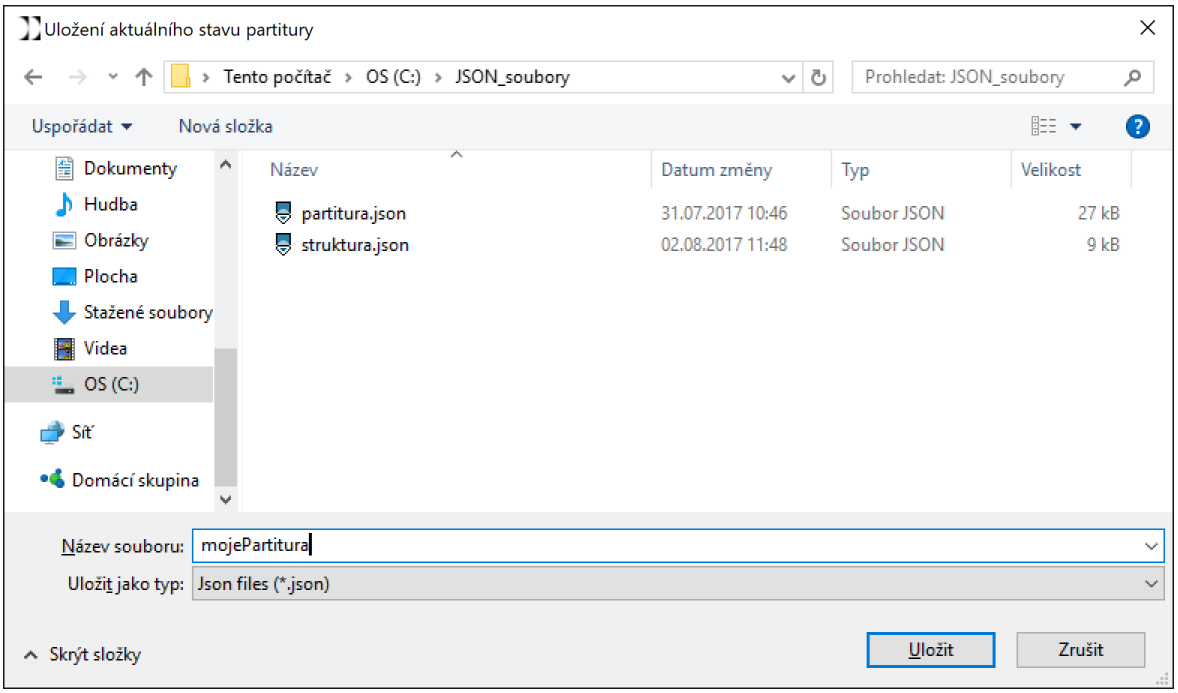
Obrázek 27: Dialogové okno pro načtení partitury. Zdroj: vlastní.

Další podpoložkou nabídky **Partitury** je možnost **Vykreslit Strukturu z uložené partitury**. Tato možnost otevírá dialogové okno pro vybrání JSON souboru obsahujícího partituru. Pokud je partitura kompletní, aplikace vykreslí a uloží obrázek Struktury. Pokud partitura komplení není, uživatel je o tom informován a dotázán, zda chce znovu načíst soubor (obrázek fig:dialogZnovuNacteniPartitury). Kliknutím na tlačítko **Ano** se znovu otevře dialogové okno pro vybrání JSON souboru obsahujcího partituru. Kliknutím na tlačítko **Ne** operací zruší.



Obrázek 28: Dialogové okno pro znovunačtení partitury. Zdroj: vlastní.

Poslední podpoložkou je možnost **Uložit partituru**. Kliknutím na ni se otevře dialogové okno (obrázek 29) pro vybrání názvu a umístění souboru. Souboru bude typu JSON a bude v něm uložena partitura obsahující elementy a znaménka koeficientu, zvolené pravidlo a hodnotu koeficientu dle aktuálního stavu mřížky.

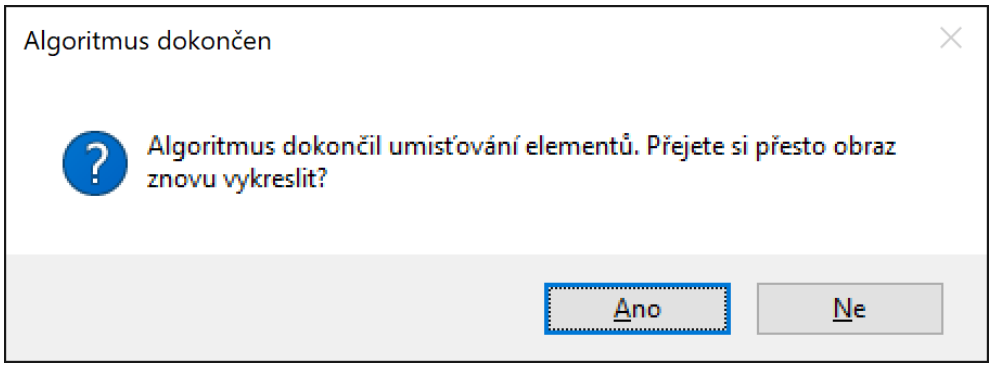


Obrázek 29: Dialogové okno pro uložení partitury. Zdroj: vlastní.

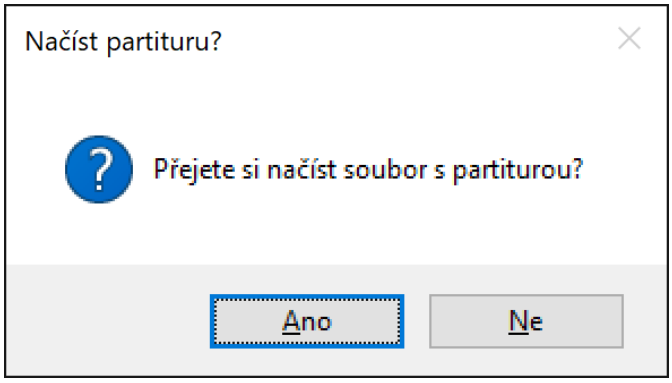
4.1.2 Spuštění algoritmu

Oblast v obrázku 20 označená žlutě

Tlačítko **Spustit algoritmus** spouští algoritmus pro umístění elementů do mřížky. Bere v potaz hodnoty z rolovací nabídky **Zvolené pravidlo** a z číselného pole **Koeficient**. Pokud již byl algoritmus nad aktuální partiturou spuštěn, aplikace o tom uživatele informuje dialogovým oknem s dotazem, zda chce přesto znovu vykreslit výsledný obraz. Pokud uživatel klikne na tlačítko Ne, je dotázán, zda chce načíst JSON soubor se strukturou.



Obrázek 30: Dialogové okno Spustit znovu. Zdroj: vlastní.



Obrázek 31: Dialogové okno Načíst partituru?. Zdroj: vlastní.

4.1.3 Mřížka pro umisťování elementů a znamének koeficientů

Oblast v obrázku 20 označená zeleně

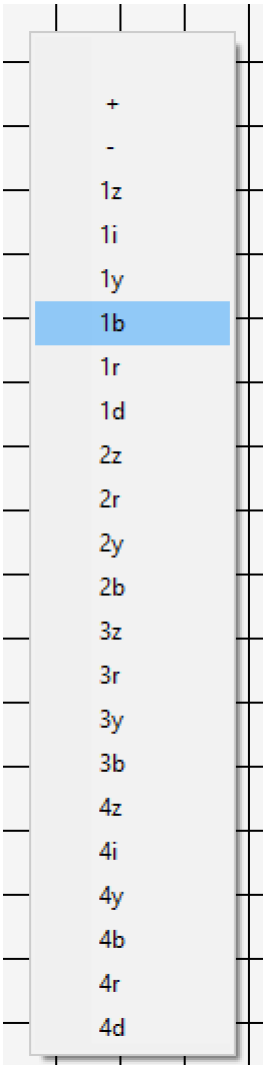
Mřížka slouží pro volitelné umístění elementů či znamének koeficientů před spuštěním algoritmu. Všechny buňky mají ve výchozím stavu šedé pozadí. V případě, že v buňce proběhla změna, má bílé pozadí.

Oblast pro vykreslení mřížky se do rozměru 16 řádků x 15 sloupců vejde do připravené oblasti. Při překročení těchto hodnot přibude dle potřeby horizontální či vertikální posuvník.

Pro orientaci v mřížce slouží popisek nad ní. Ten ukazuje aktuální pozici kurzoru.

Obrázek 32: Pozice kurzoru v mřížce. Zdroj: vlastní.

Element či znaménko koeficientu lze do buňky umístit kliknutím. **Klinutí pravým talčítkem** vyvolá **kontexovou nabídku** (obrázek 33), která obsahuje všechny dostupné elementy, znaménka plus a minus a prázdnou položku pro vyprázdnění buňky. **Kliknutí levým tlačítkem** náhodně vybere jednu položku z kontextové nabídky.



Obrázek 33: Kontextová nabídka mřížky. Zdroj: vlastní.

4.1.4 Náhled výsledného obrazu

Oblast v obrázku 20 označená modře

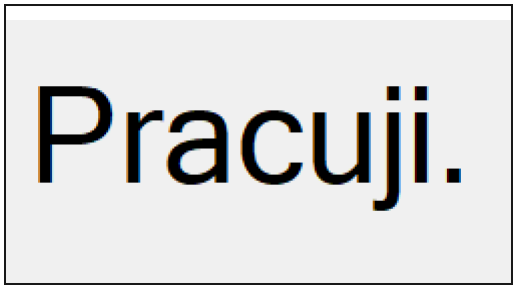
Po dokončení činnosti algoritmu se v této oblasti zobrazí výsledný obrázek, tak aby pasoval do připravené oblasti. Dvojklikem na náhled obrázku se otevře uložený obrázek. Výsledný obrázek v plné velikosti je zároveň uložen do adresáře, ze kterého je aplikace spuštěna. Obrázek je formátu PNG (Portable Network Graphics), který je určen pro bezeztrátovou kompresi rastrové grafiky.

Název uloženého obrázku má následující formát: *název struktury*_*počet řádků*_x_*počet sloupců*_V*číslo zvoleného pravidla*_c *hodnota koeficientu*_*RRRRMMDD*_*hh*_*mm*_*ss*.png. Příklad: Cernobila_22x11_V2_c0,75_20170801_22_40_14.png.

Pod náhledem výsledného obrázku je tlačítko **Návrat k předchozí partituře**. Toto tlačítko vrátí partituru do stavu před spuštěním algoritmus.

4.1.5 Okno signalizující činnost aplikace

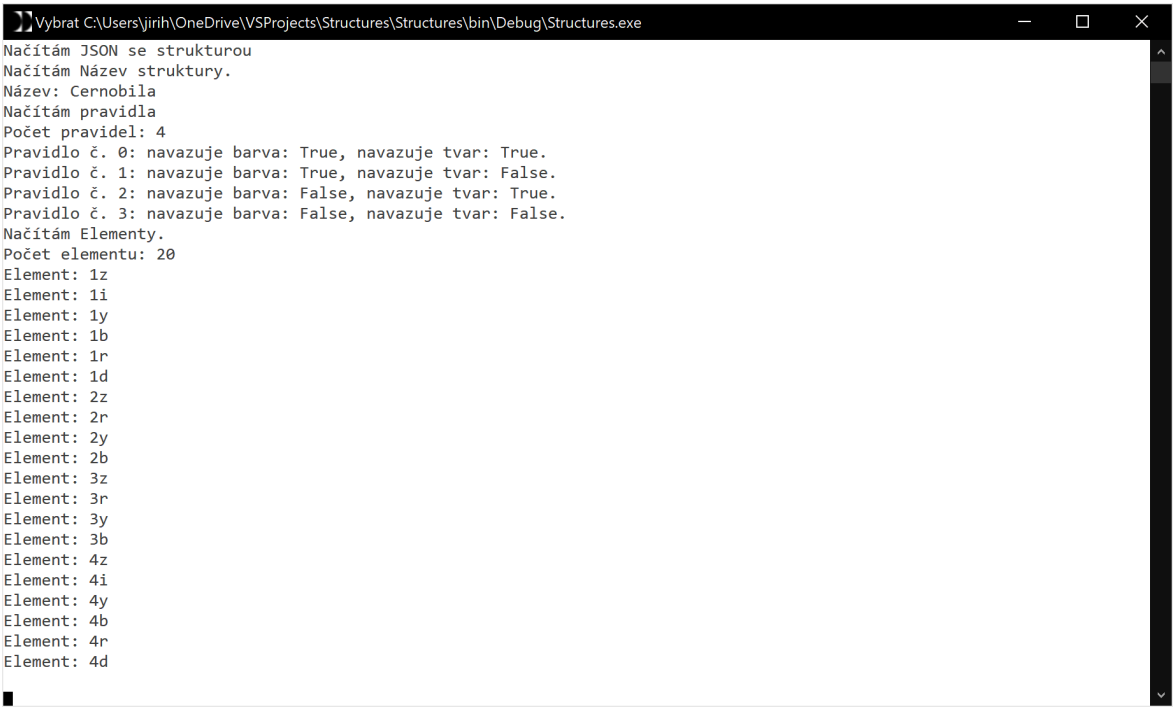
V případě že je aplikace v činnosti (načítá nebo ukládá soubor nebo je spuštěn výpočet) je uživateli zobrazeno okno, signalizující činnost aplikace.



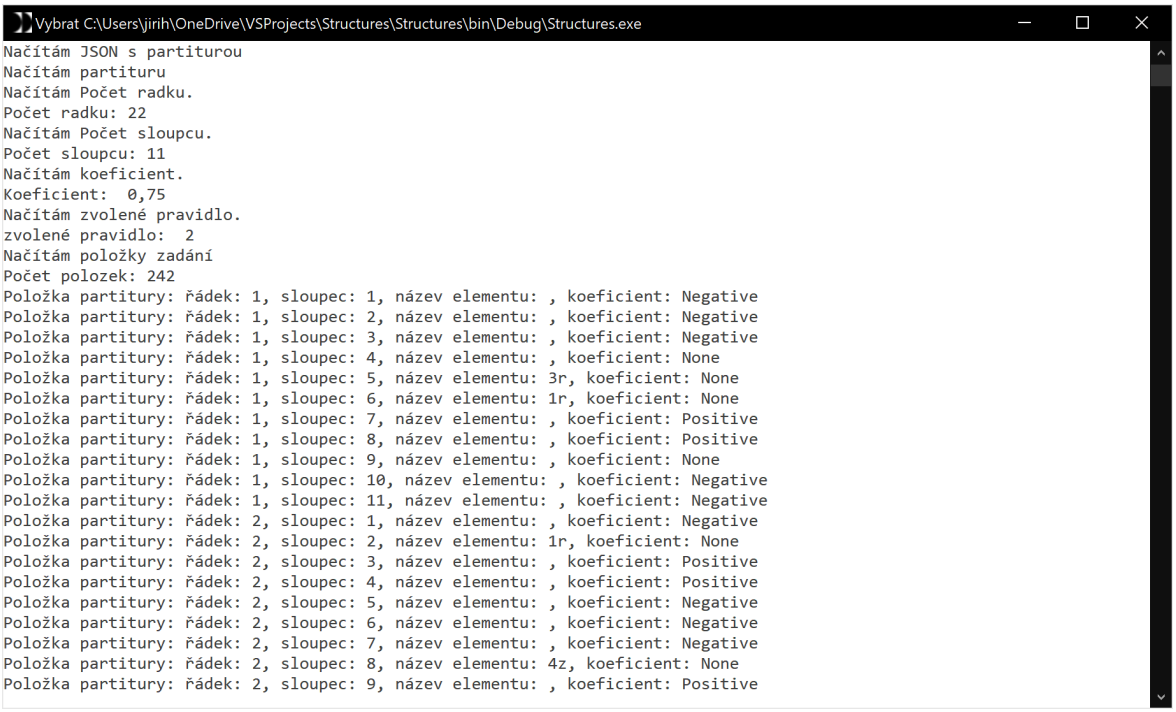
Obrázek 34: Okno signalizující činnost aplikace. Zdroj: vlastní.

4.2 Konzolové okno

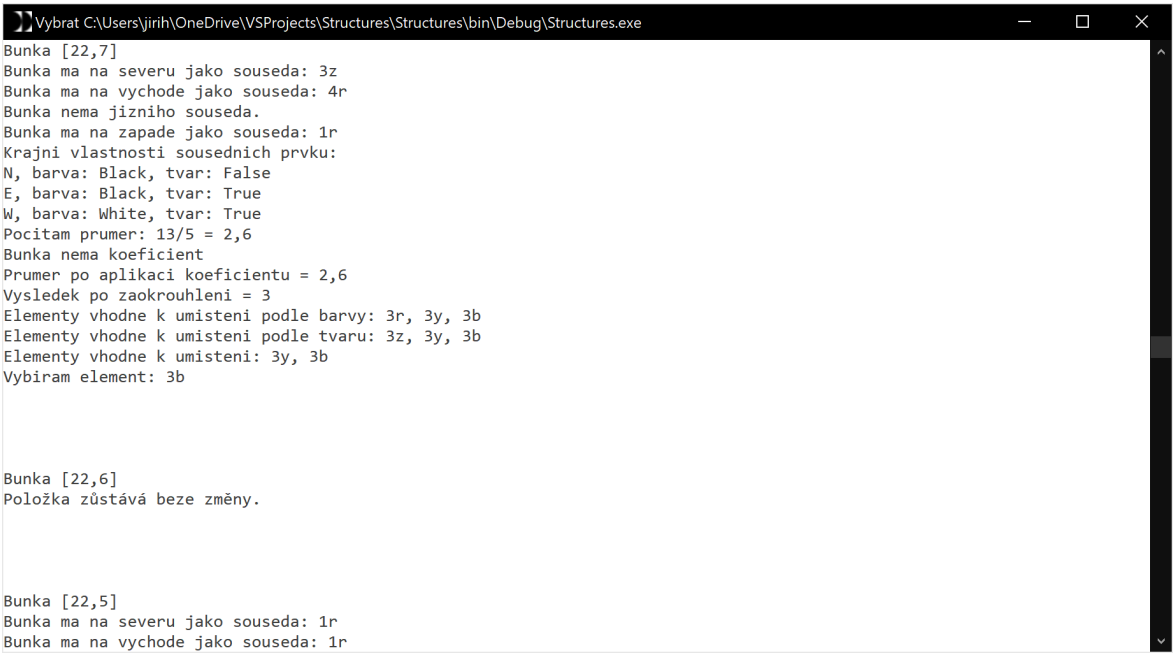
Aplikace kromě okna s ovládacími prvky zobrazuje i okno konzole, které slouží pro výpis některých informací o činnosti aplikace. Jedná se o informace o načtených hodnotách z JSON souborů a průběh algoritmu umisťování elementů do mřížky.



Obrázek 35: Ukázka konzolového okna – načtení struktury. Zdroj: vlastní.



Obrázek 36: Ukázka konzolového okna – načtení partitury. Zdroj: vlastní.



Obrázek 37: Ukázka konzolového okna – průběh algoritmu. Zdroj: vlastní.

5 Generování obrazů pro různé modifikace parametrů partitury z časopisu Leonardo

Součástí článku v časopise Leonardo [6] je i ukázka jedné partitury. Tato partitura byla využita v kapitole 2.3.1. Partitura má:

- 22 řádků,
- 11 sloupců,
- ve 45 buňkách umístěný element,
- v 63 buňkách kladný koeficient,
- v 60 buňkách záporný koeficient a
- 74 volných buněk.

5.1 Generování Černobílé Struktury

Pro generování byla využita dříve zmíněná partitura. Elementy Černobílé struktury jsou popsány v kapitole 2 na straně 26. Generovalo se celkem 12 různých kombinací. Měnila se hodnota koeficientu (c) a zvolené pravidlo (V). Kombinace jsou v tabulce 5.

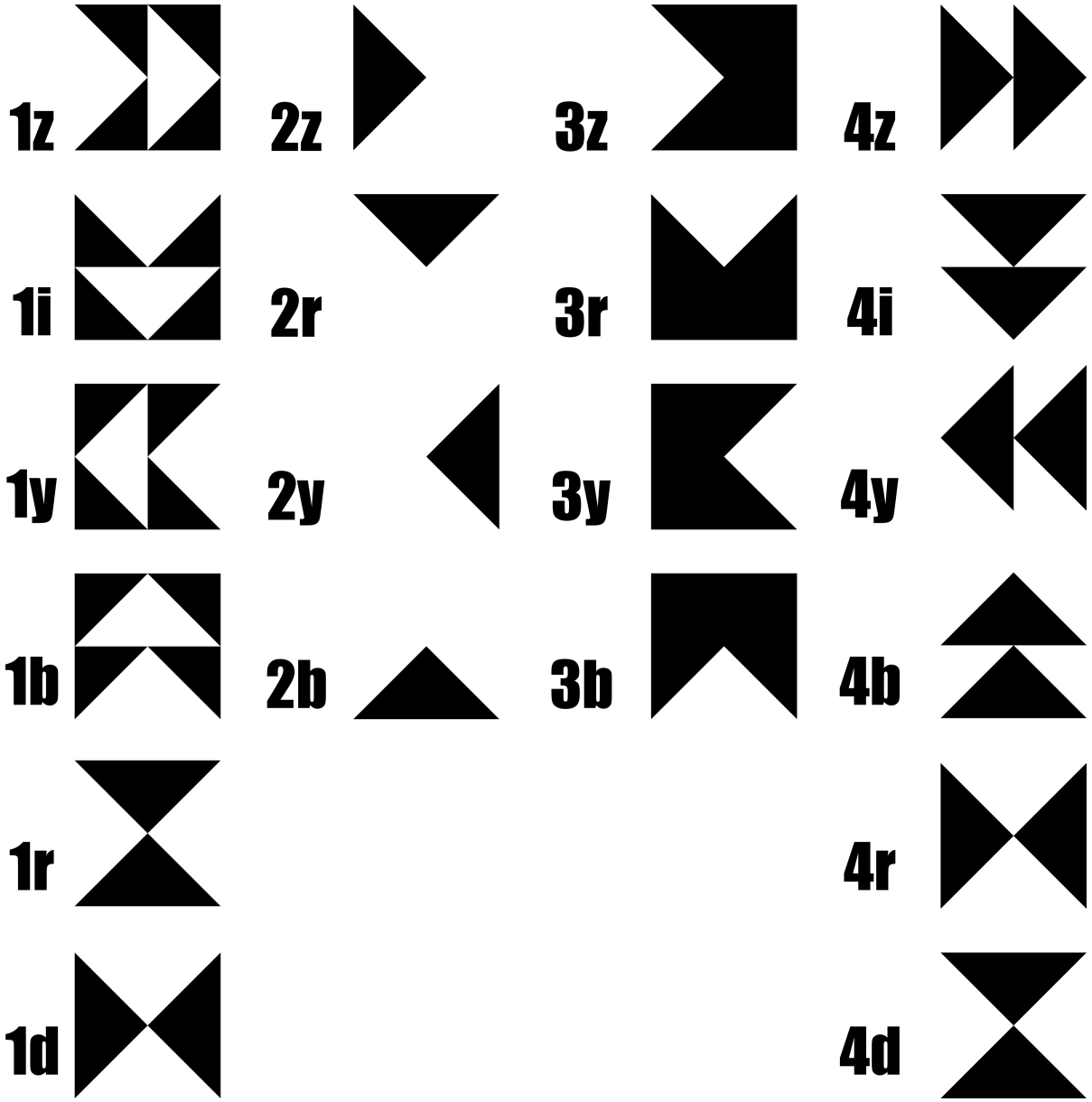
Tabulka 5: Vstupní kombinace pro generování Černobílé Struktury

Označení kombinace	**c**	V
k1	0,75	0
k2	0,75	1
k3	0,75	2
k4	0,75	3
k5	0,5	0
k6	0,5	1
k7	0,5	2
k8	0,5	3
k9	1,5	0
k10	1,5	1
k11	1,5	2
k12	1,5	3

Vygenerované Struktury jsou v příloze.

5.2 Generování Trojúhelníkové Černobílé Struktury

K demonstrování možnosti využít aplikaci pro generování i jiné než Černobélé Struktury jsem vytvořil Trojúhelníkovou Černobílou Strukturu. Tato struktura z Černobílé vychází. Má stejný počet elementů se stejnými názvy, elementy mají stejné vlastnosti. Změna je pouze v tom, jak elementy vypadají. Místo půlkruhu je použit trojúhelník.



Obrázek 38: Dvacet elementů pro Trojúhelníkovou Černobílou strukturu. Zdroj: vlastní.

Ke každému výsledku z generování Černobílé Struktury z předchozí části byla vykreslena Trojúhelníková Černobílá Struktura. Výsledky byly označeny jako k13-k24. Vygenerované Struktury jsou v příloze.

5.3 Pozorování výsledků

Při pozorování výsledků je patrný Sýkorův záměr. Pomocí umístění koeficientů se snažil uprostřed vytvořit jakýsi světlý kříž. Ostatní elementy zřejmě umístil s nějakým smyslem pro dotažení záměru vytvoření kříže. Změnou koeficientu v kombinacích k5-k12 se kříž sice udržel, ale není tak patrný jako v kombinacích k1-k4.

Použitím Trojúhelníkové Černobílé Struktury se Sýkorův záměr výrazně ztrácí. Je to způsobeno tím, že podkladová barva (u skupin 1 a 3 černá; u skupin 2 a 4 bílá) v elementu zabírá větší plochu než u Černobílé Struktury. Naopak užitím této Struktury vznikly ve výsledném obraze nové zajímavé oblasti. Například „kříž“ ve spodní části Struktury k22.

Závěr

Hlavním cílem této práce bylo nastudovat principy systému pro tvorbu strukturálních obrazů z článku Computer-aided multielement geometrical abstract paintings, který Zdeněk Sýkora společně s Jaroslavem Blažkem publikovali v roce 1970 v časopise Leonardo. Poté na základě těchto principů vytvořit aplikaci, která bude vytvářet strukturální obrazy.

V první části je popsán život Zdeňka Sýkory a vývoj jeho tvorby. V druhé kapitole jsou vysvětleny principy systému pro tvorbu struktur společně s Černobílou strukturou. Jsou zde vysvětlena pravidla a algoritmus pro umisťování elementů včetně příkladů. Tyto příklady jsou postaveny na zadání, které je součástí článku v časopise Leonardo. Třetí kapitola se věnuje popisu implementace aplikace. Na úvod je krátce představen jazyk C# a framework Windows Forms. Dále se kapitola věnuje struktuře tříd v aplikaci a podrobněji stěžejním částem kódu s popisem některých metod. Čtvrtou kapitolou je uživatelská příručka, která vysvětluje užití programu a naznačuje strukturu očekávaných vstupních souborů. Poslední kapitola se věnuje generování Struktur na základě zadání z článku v časopise Leonardo s obměnou koeficientů a s použitím vlastní Struktury, založené na Černobílé.

Aplikace pracuje s tzv. partiturou, což je textové vyjádření obrazu. Partitura slouží jako zadání pro algoritmus umisťování elementů. Partituru může uživatel sám vytvořit nebo ji načíst z JSON souboru v očekávané struktuře. Výslednou partituru si též lze uložit. Aplikace umožňuje načítat různé struktury z JSON souboru v definovaném formátu. Potenciální rozšíření aplikace je ve vytvoření uživatelského rozhranní pro tvorbu struktur a automatizovat tak vytvoření JSON souboru. Výsledný obrázek aplikace ukládá ve formátu PNG. Rozlišení výsledného obrázku závisí na rozlišení obrázků jednotlivých elementů definovaných ve struktuře. Obrazy vznikající rotací a výřezem, jenž byly zmíněny v zadání této diplomové práce, směřují k Makrostrukturám. Tato část není implementována, neboť se jedná o otočení výsledného obrázku či jeho výřez. Aplikací pro takové operace je k dispozici mnoho. Na přiloženém CD je k dispozici soubor pro definici Černobílé struktury a soubor s partiturou se zadáním z článku v časopise Leonardo.

Téma jsem si zvolil, protože mě zaujalo spojení využití počítače s výtvarným uměním. V podání Zdeňka Sýkory se nejedná o abstraktní umění v pravém smyslu slova. I když značnou roli hraje náhoda, vnesl do svých obrazů řád.

Literatura

[1] SÝKORA, Zdeněk a Pavel KAPPEL. Zdeněk Sýkora 90. Prague: Verzone, 2010. ISBN 978- 80-904546-1-3.

[2] SÝKOROVÁ, Lenka. Zdeněk Sýkora – Vítek Čapek – Rozhovor se Zdeňkem Sýkorou [online].©2017. [cit. 8. 7. 2017]. Dostupné z: www.zdeneksykora.cz/?s=156.

[3] SÝKOROVÁ, Lenka. Zdeněk Sýkora – životopis [online].©2017. [cit. 2017-08-07]. Dostupné z: http://www.zdeneksykora.cz/?s=zivotopis.

[4] Zdeněk Sýkora. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-08-08]. Dostupné z: https://cs.wikipedia.org/wiki/Zden%C4%9Bk_S%C3%BDkora

[5] Wikipedie: Otevřená encyklopedie: Skupina Křižovatka [online]. ©2016 [cit. 2017-07-09]. Dostupné z:https://cs.wikipedia.org/w/index.php?title=Skupina_K%C5%99i%C5%BEovatka&oldid=14083417.

[6] SÝKORA, Zdeněk a Jaroslav BLAŽEK. Computer aided Multi element Geometrical Abstract Paintings. Leonardo. 1970, roč. 3, s. 409.

[7] ČT2, Evropané: 10. 8. 2002 - Zdeněk Sýkora. [online]. [cit. 2017-07-10]. Dostupné z: http://www.ceskatelevize.cz/ivysilani/1026666614-evropane/20136219568-zdenek-sykora/

[8] SVOBODA, M. - MAREK, J. Construction and statistical analysis of Zdeněk Sýkora's lines. In APLIMAT 2014: 13th Conference on Applied Mathematics: proceedings. Bratislava: Slovenská technická univezita v Bratislave, 2014. s. 387-392. ISBN 978-80-227-4140-8.

[9] MAREK, J. - NEDVĚDOVÁ, M. Historie digitálního umění: Náhoda, počítač a Linie Zdeňka Sýkory. In 37. mezinárodní konference Historie matematiky. Praha: Matfyzpress, 2016. s. 137-146. ISBN 978-80-7378-317-4.

[10] SÝKORA, Zdeněk, SÝKOROVÁ, Lenka, ed. Zdeněk Sýkora: grafika. Praha: Gallery, c2008. ISBN isbn978-80-86990-36-1.

[11] Portable Network Graphics. (2017, July 31). In Wikipedia, The Free Encyclopedia [cit. 2017-08-06]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Portable_Network_Graphics&oldid=793281727.

[12] Portable Document Format. (2017, July 31). In Wikipedia, The Free Encyclopedia [cit. 2017-08-06] Dostupné z:https://en.wikipedia.org/w/index.php?title=Portable_Document_Format&oldid=793153938.

[13] JSON. (2017, July 31). In Wikipedia, The Free Encyclopedia [citováno 2017-08-06] Dostupné z:https://en.wikipedia.org/w/index.php?title=JSON&oldid=792937164.

[14] Json.NET - Newtonsoft. Object moved [online]. Copyright © 2017 Newtonsoft [cit. 2017-08-06]. Dostupné z: http://www.newtonsoft.com/json.

[15] PECINOVSKÝ, Rudolf. Návrhové vzory: [33 vzorových postupů pro objektové programování]. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[16] Úvod do Windows Forms aplikací. ITnetwork.cz [online]. Praha [cit. 2017-08-07]. Dostupné z: http://www.itnetwork.cz/csharp/formulare/winforms/c-sharp-tutorial-windows-forms-okenni-aplikace-uvod.

[17] C# Windows Forms Controls Examples. Dot Net Perls [online]. Sam Allen [cit. 2017-08-07]. Dostupné z: https://www.dotnetperls.com/controls.

[18] dynamic (C# Reference) | Microsoft Docs. [online] [cit. 2017-08-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/dynamic.

[19] Novinky v C# 4.0 podle CTP | Augiho web. Augiho web | Programování a auta [online] [cit. 2017-08-07]. Dostupné z: https://www.augi.cz/programovani/novinky-v-c-40-podle-ctp/.

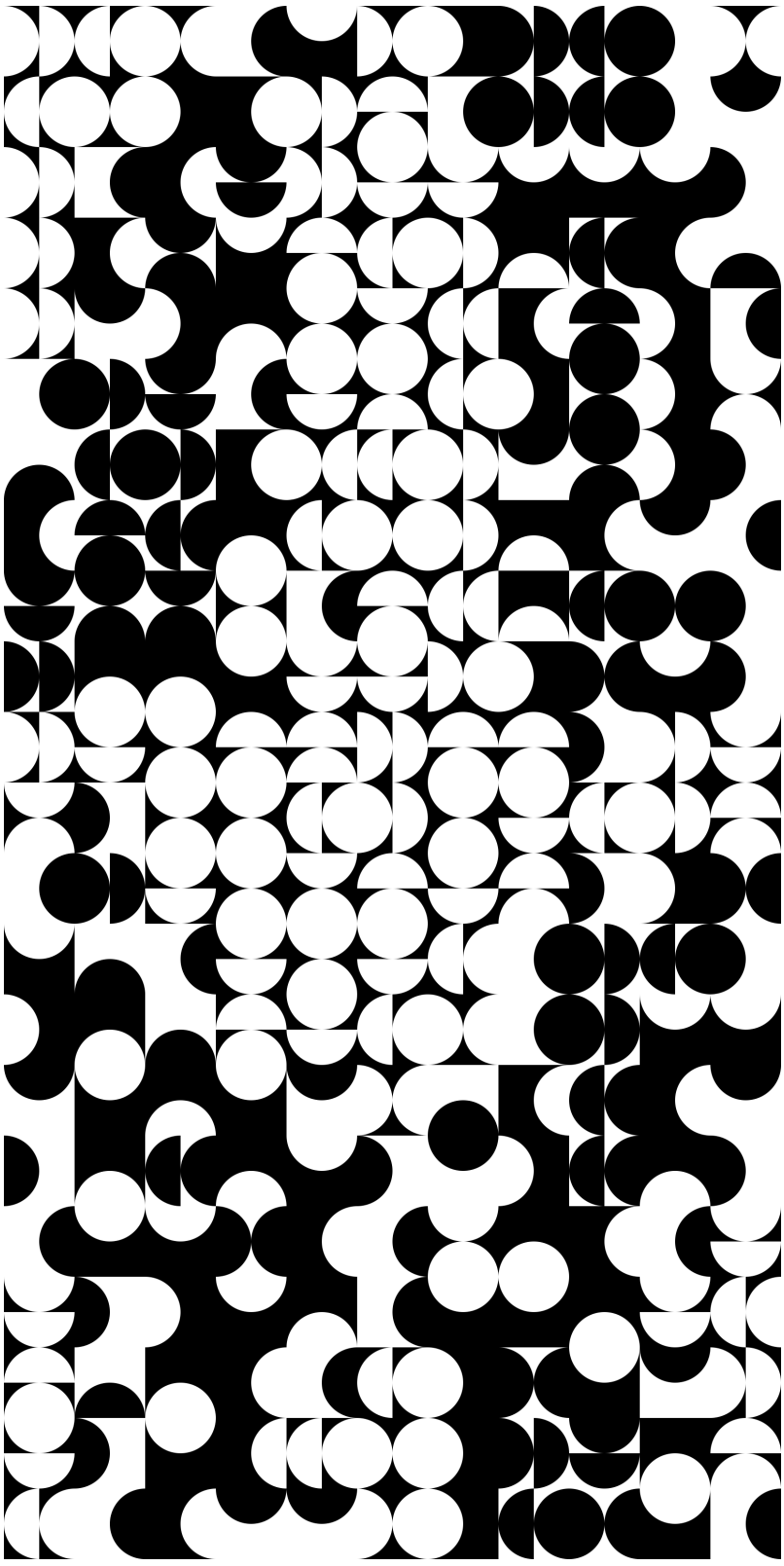
[20] NAGEL, Christian. C# 2005: programujeme profesionálně. Brno: Computer Press, 2006. Programujeme profesionálně. ISBN 80-251-1181-4.

[21] NAKOV, Svetlin, Veselin KOLEV a kol. Fundamentals of Computer Programming with C#: The Bulgarian C# Book. 1. Bulharsko: Faber Publishing, 2013. ISBN 978-954-400-773-7.

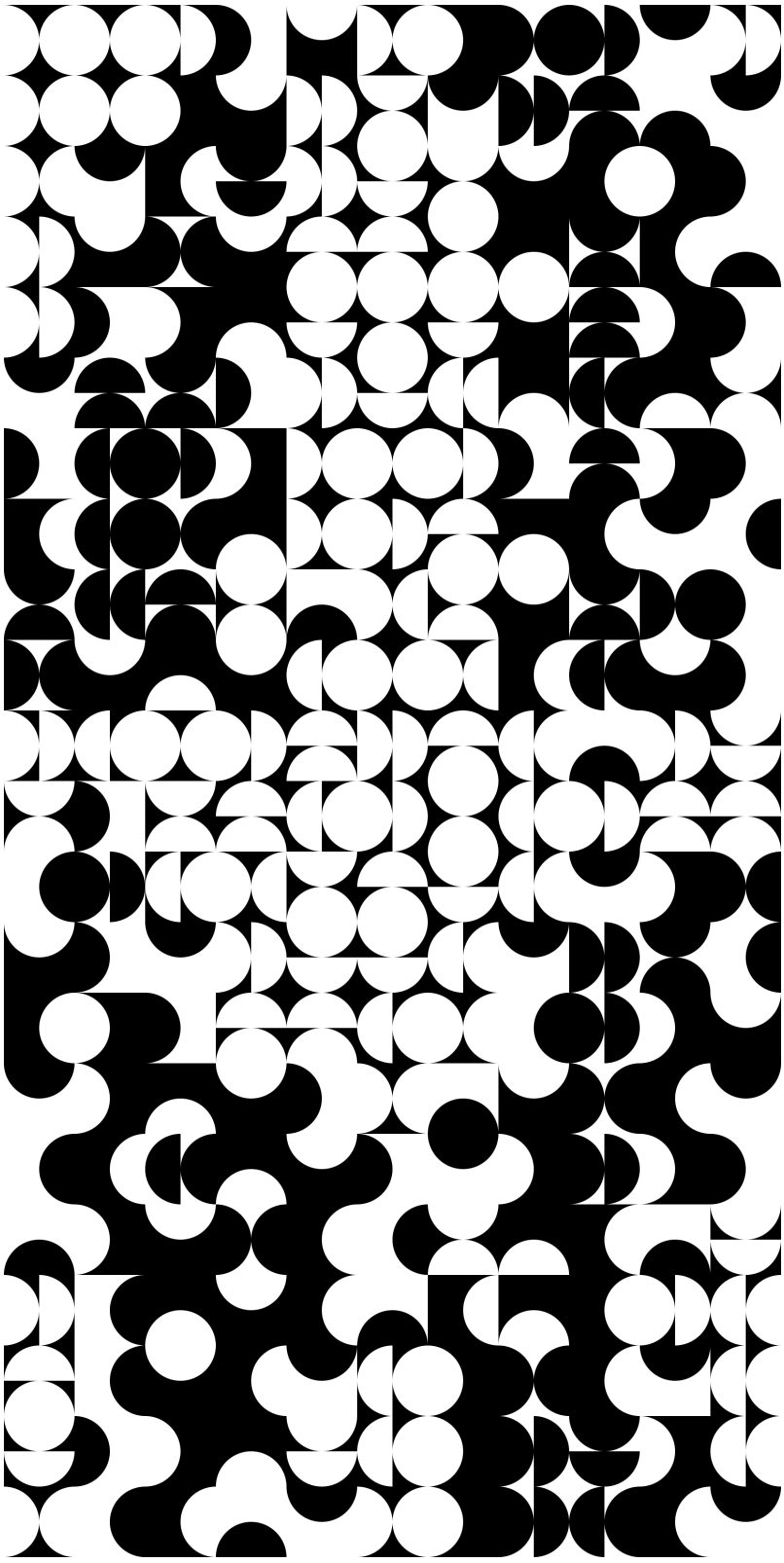
[22] SHARP, John. Microsoft Visual C# 2010: krok za krokem. Brno: Computer Press, 2010. Krok za krokem (Computer Press). ISBN 978-80-251-3147-3.

[23] List(T).FindAll Method (Predicate(T)) (System.Collections.Generic). Learn to Develop with Microsoft Developer Network | MSDN [online]. Copyright © 2017 Microsoft [cit. 2017-08-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/fh1w7y8z(v=vs.110).aspx.

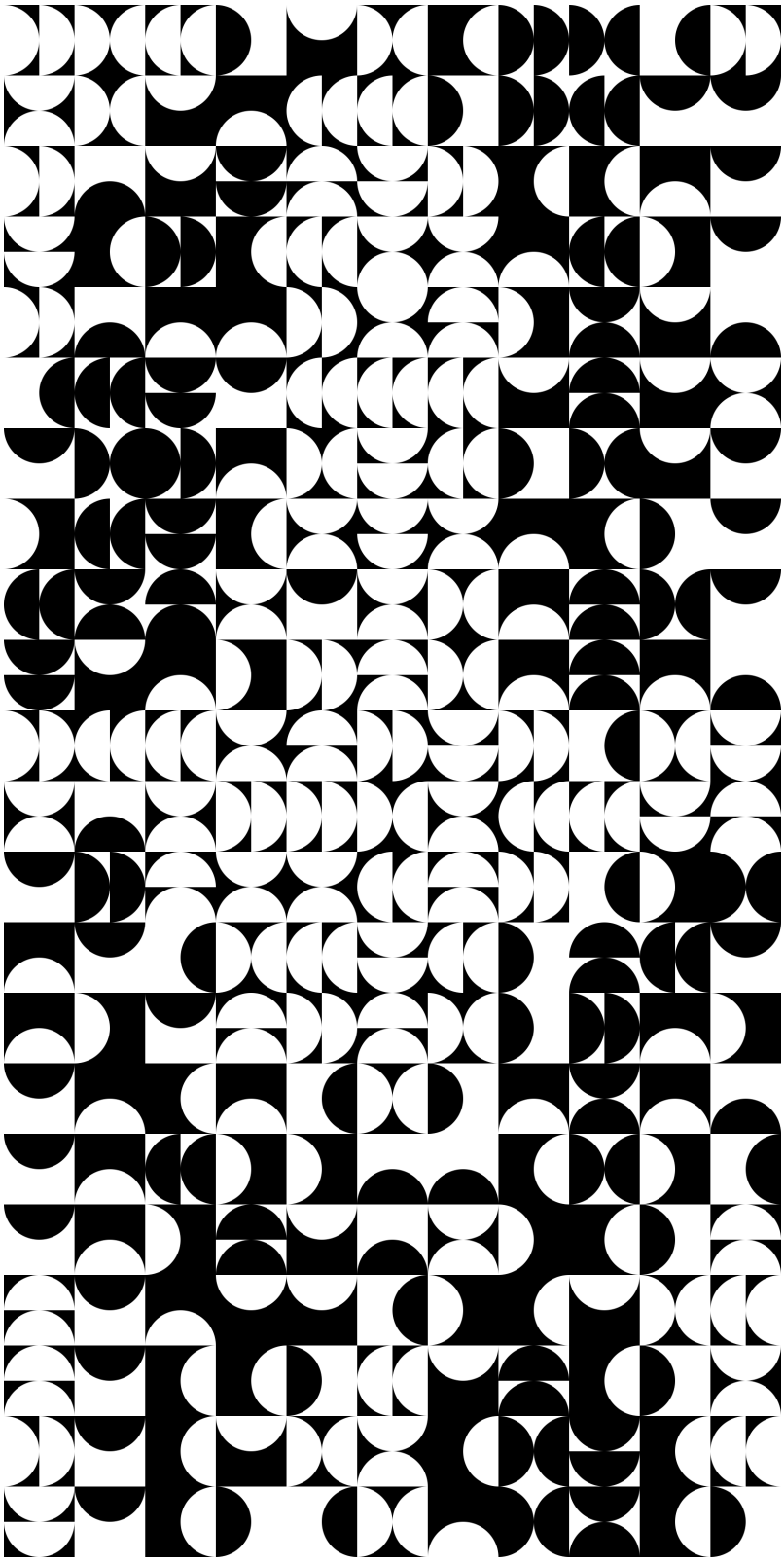
Příloha A – Vygenerované Struktury



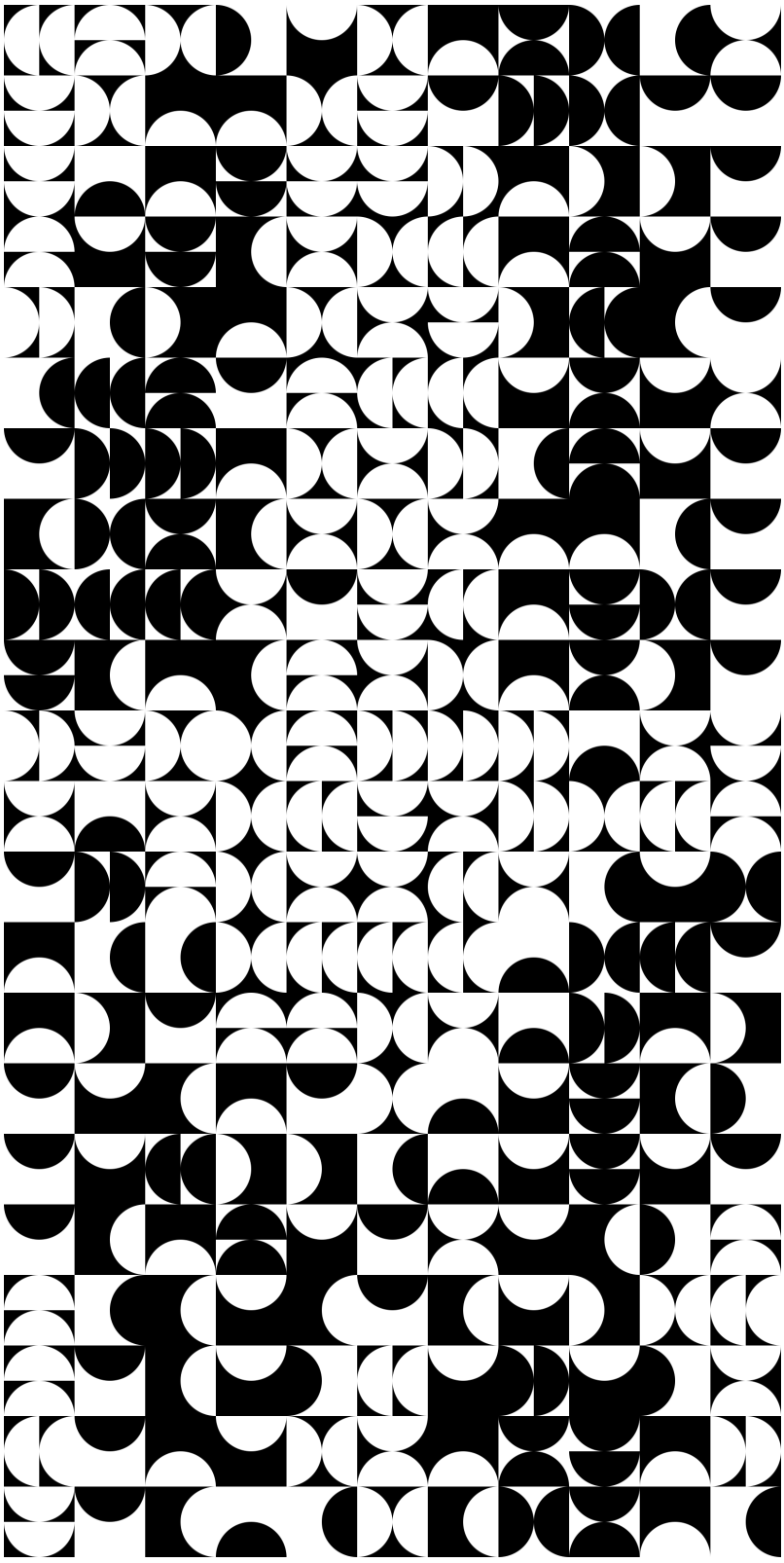
Obrázek 39: K1 - V = 0, c = 0,75. Zdroj: vlastní.



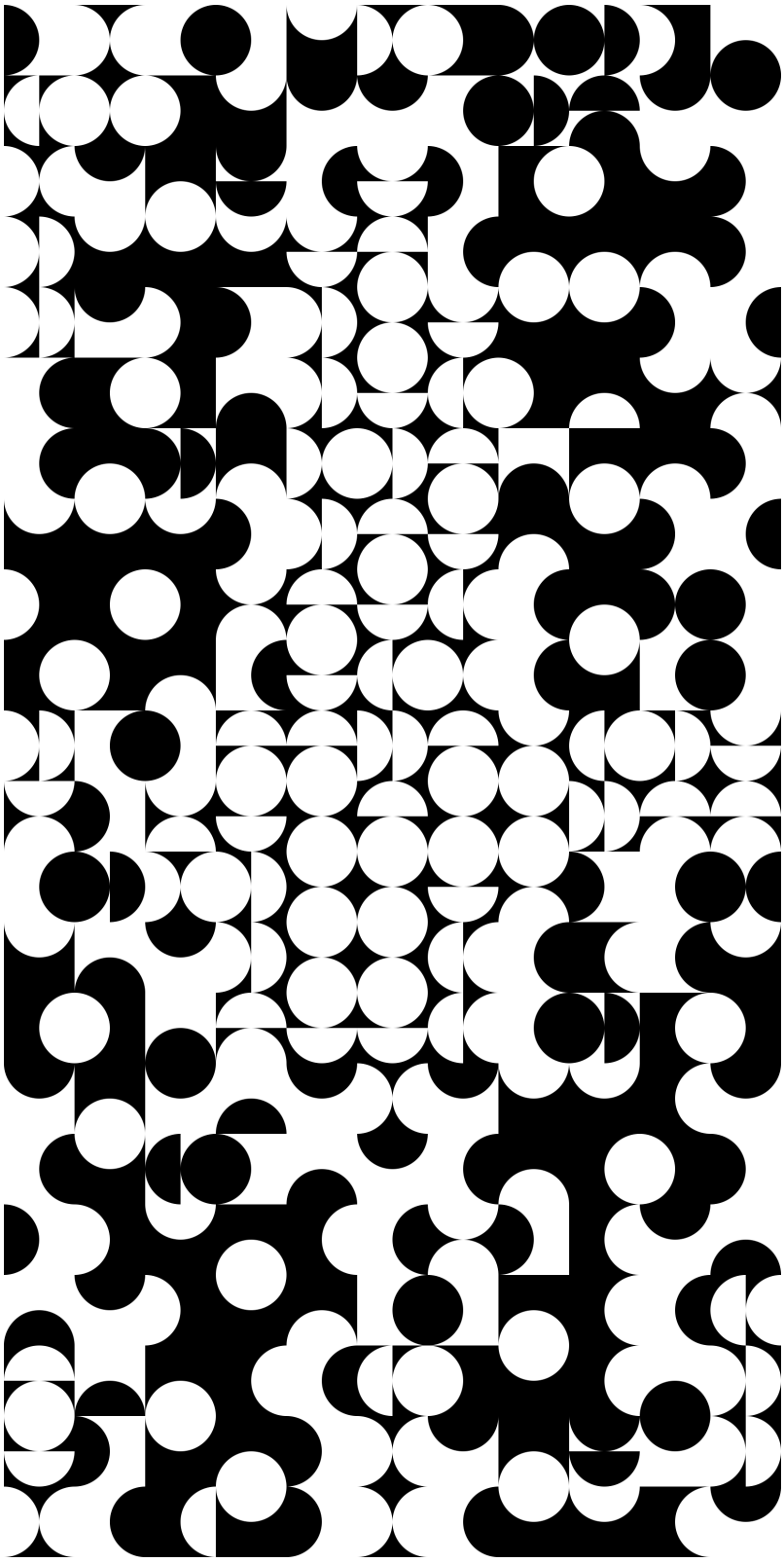
Obrázek 40: K2 - V = 1, c = 0,75. Zdroj: vlastní.



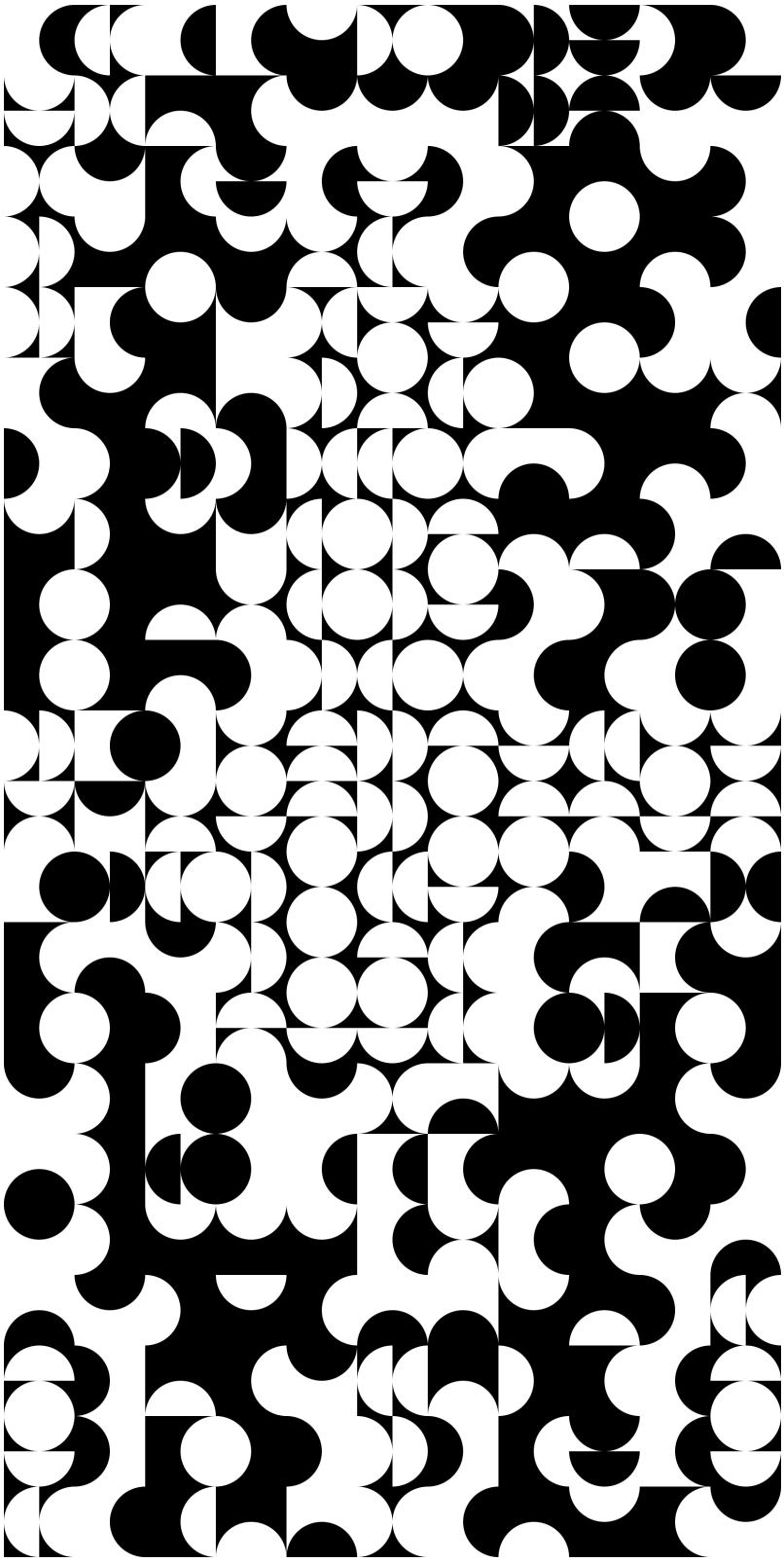
Obrázek 41: K3 - V = 2, c = 0,75. Zdroj: vlastní.



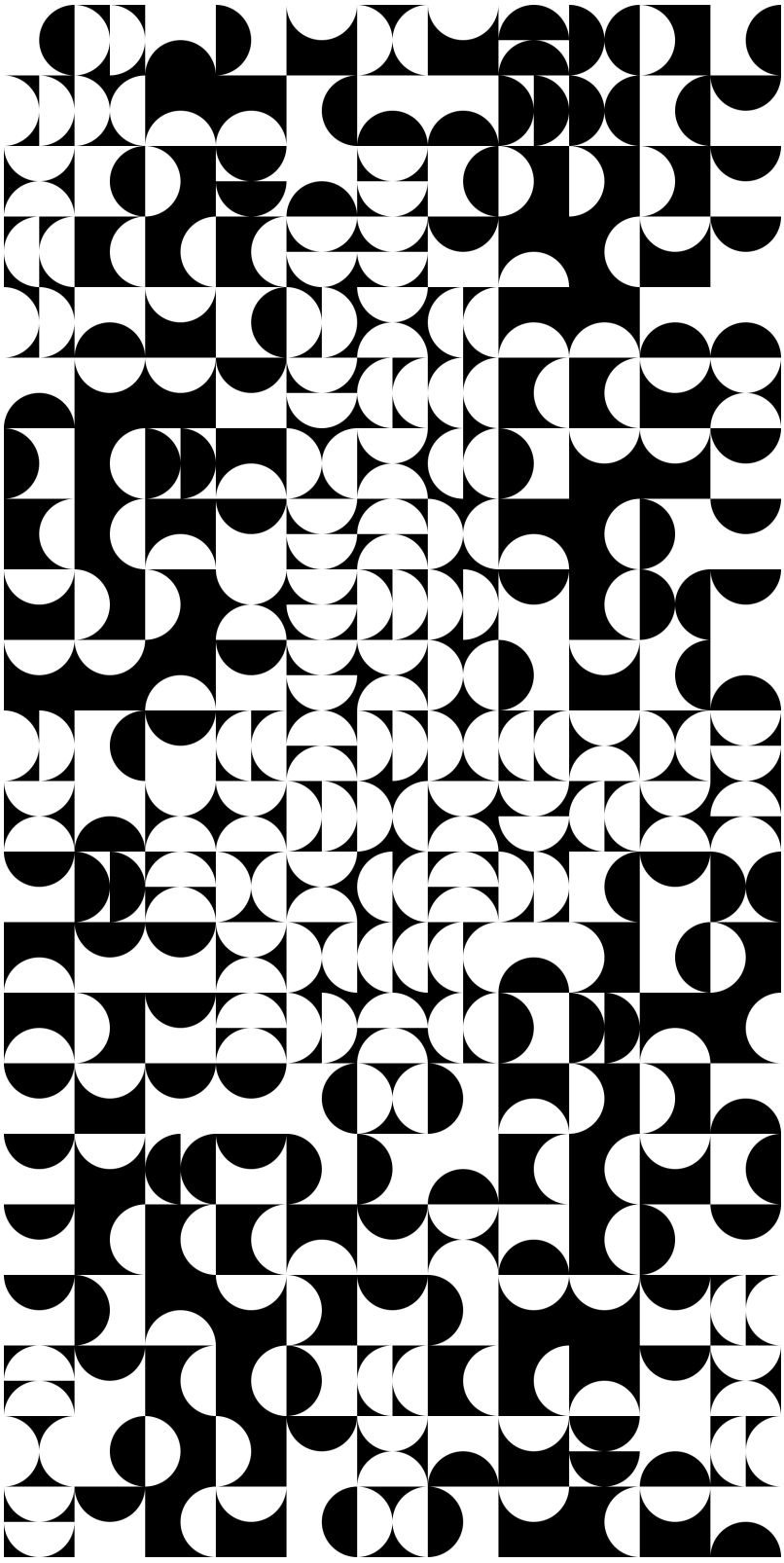
Obrázek 42: K4 - V = 3, c = 0,75. Zdroj: vlastní.



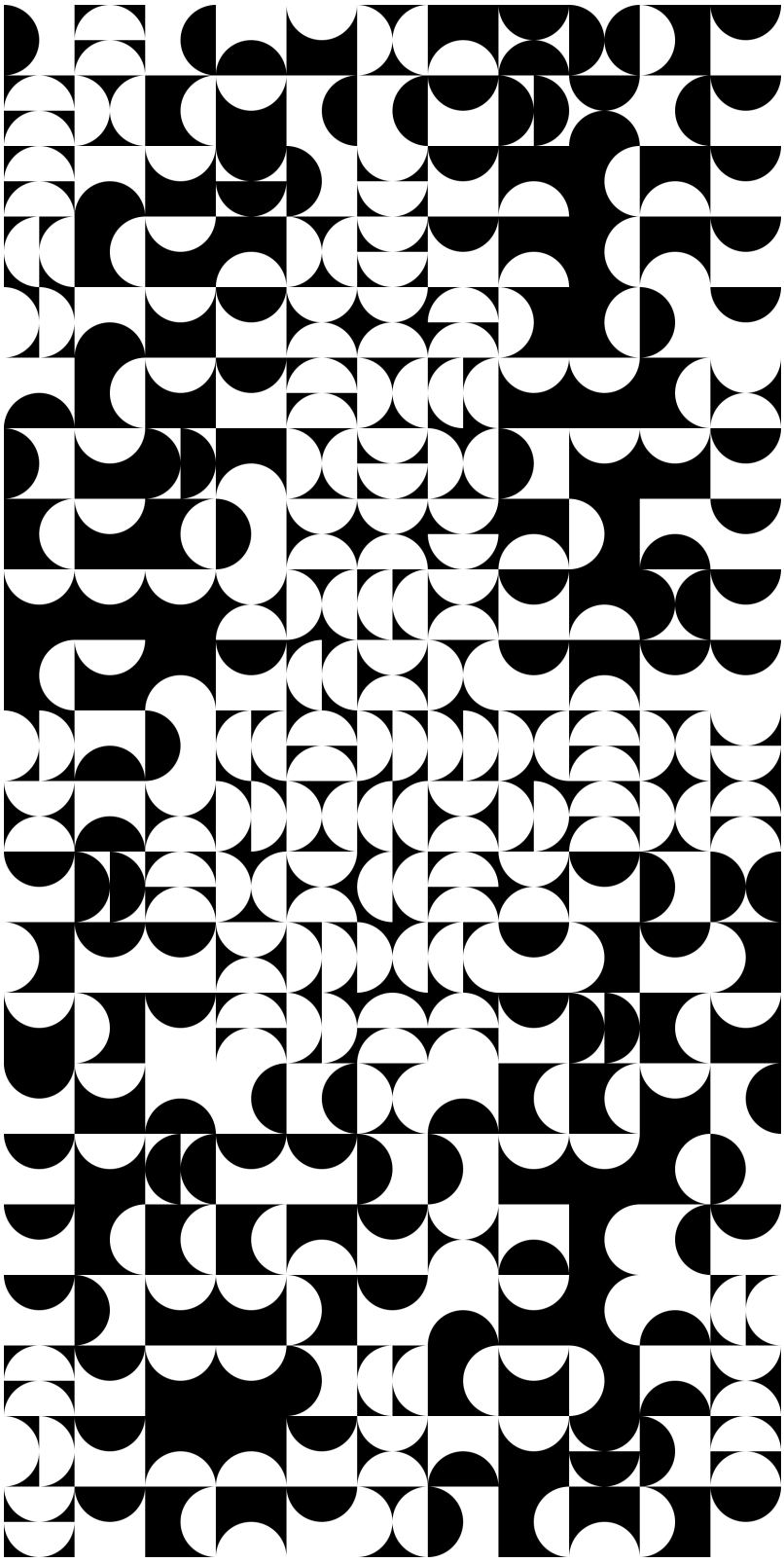
Obrázek 43: K5 - V = 0, c = 0,5. Zdroj: vlastní.



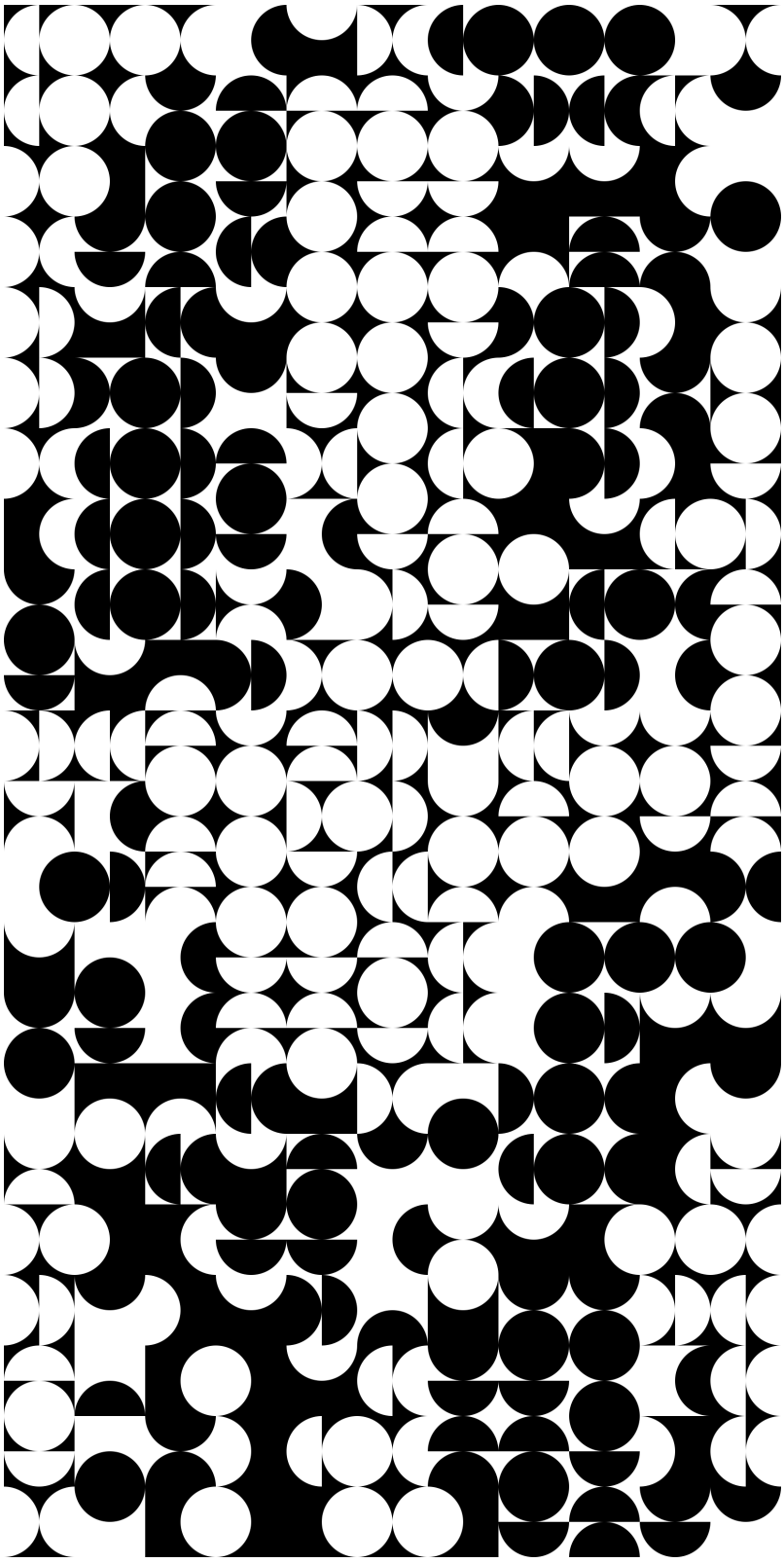
Obrázek 44: K6 - V = 1, c = 0,5. Zdroj: vlastní.



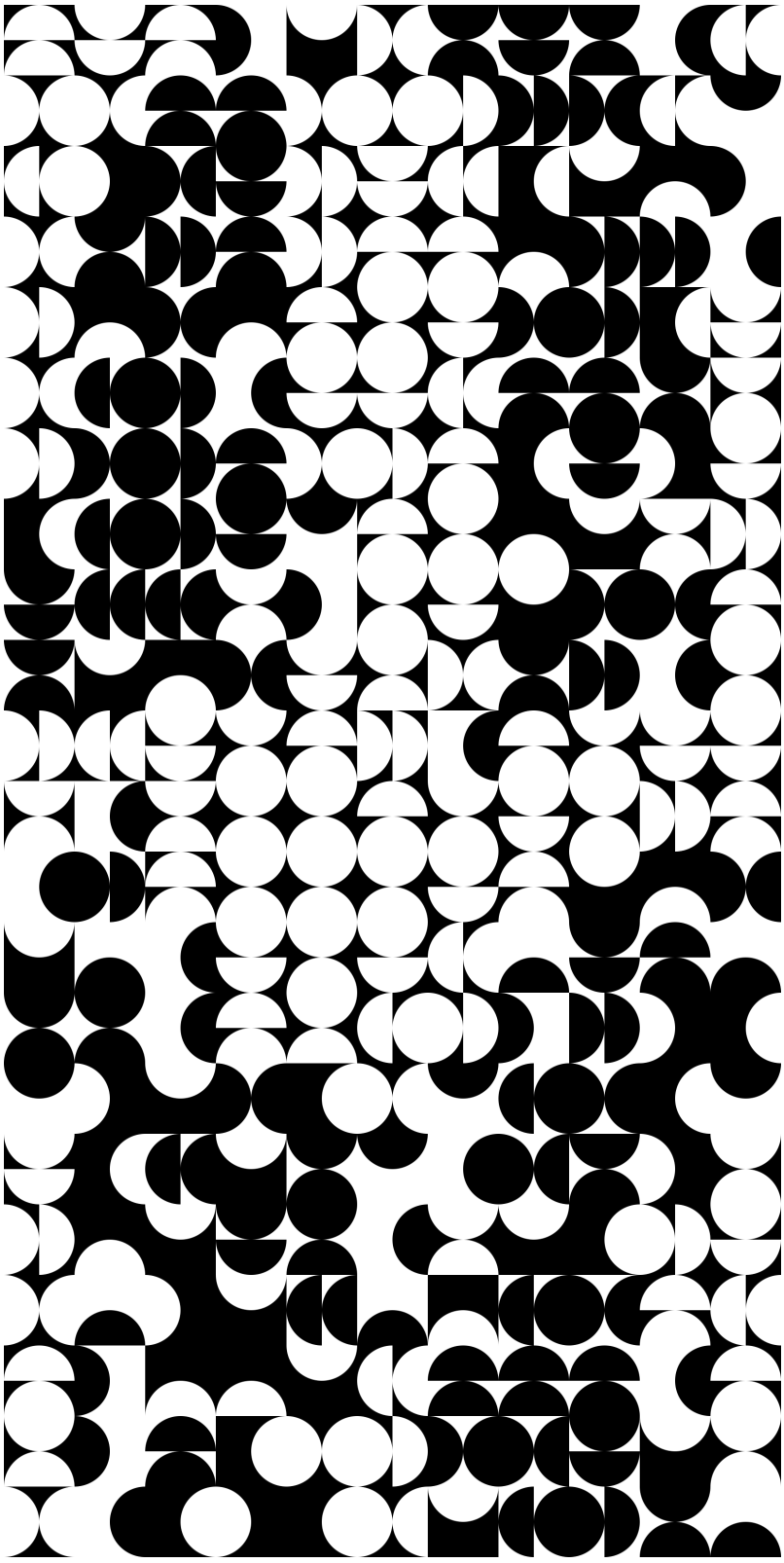
Obrázek 45: K7 - V = 2, c = 0,5. Zdroj: vlastní.



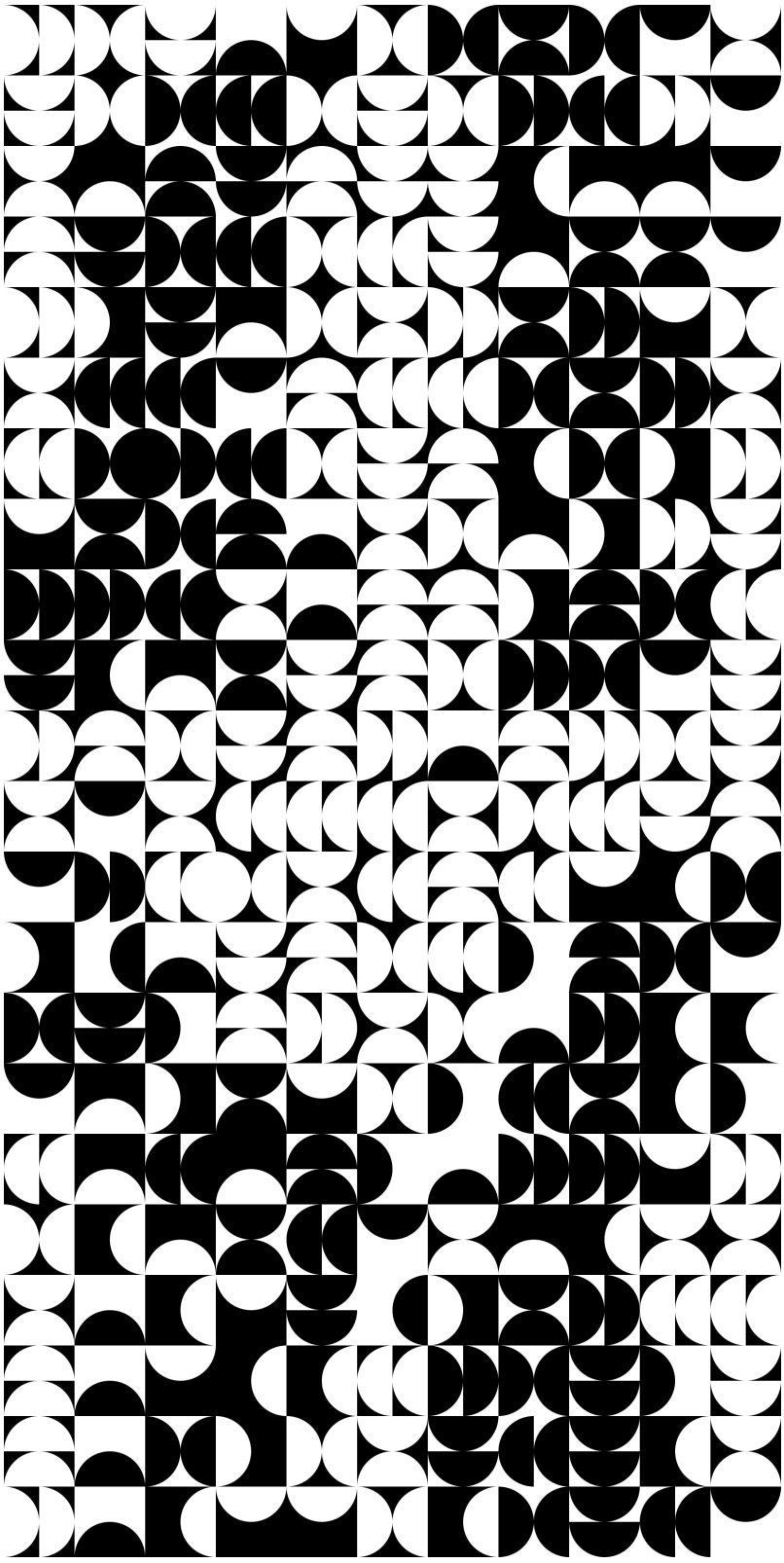
Obrázek 46: K8 - V = 3, c = 0,5. Zdroj: vlastní.



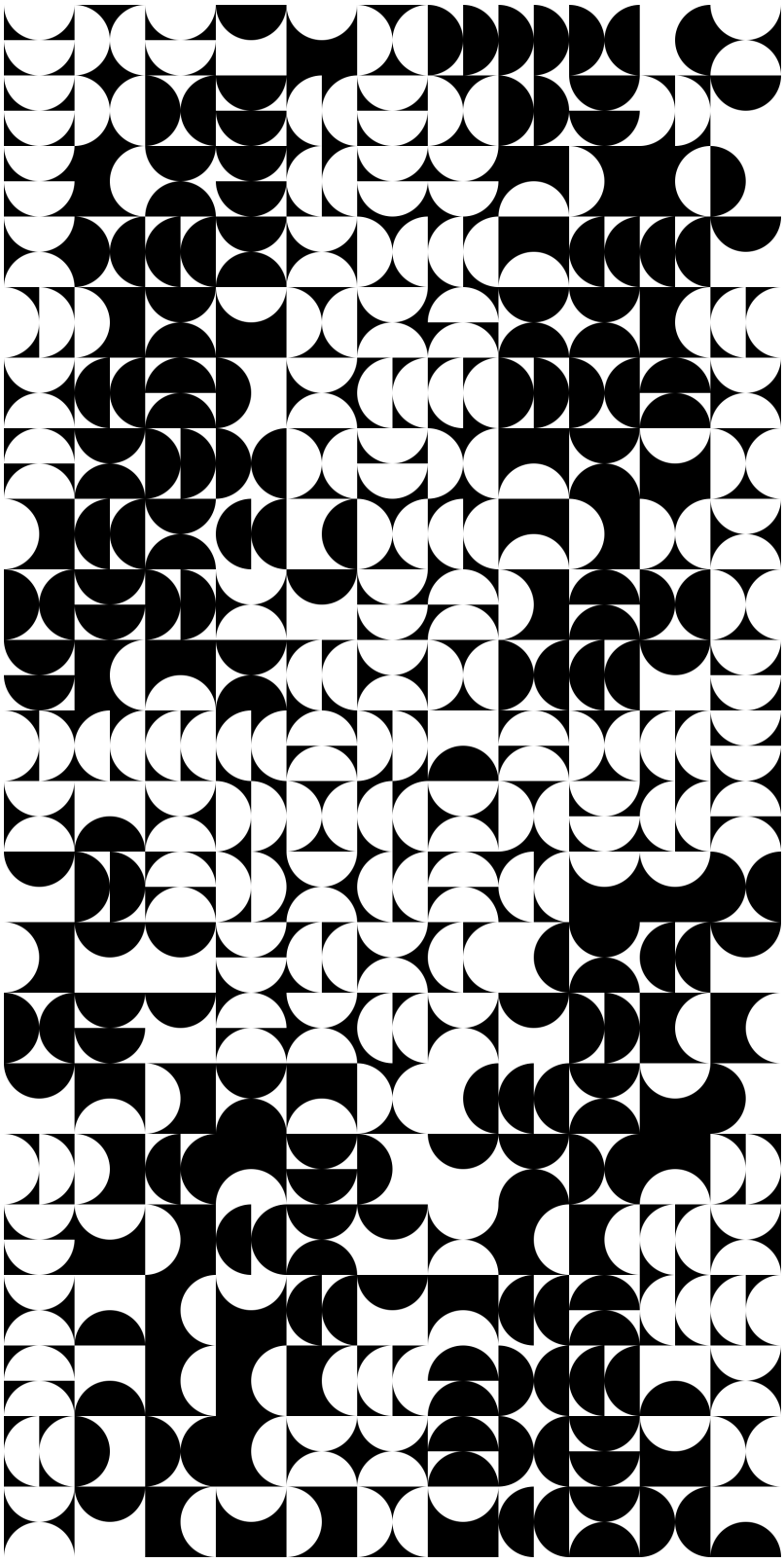
Obrázek 47: K9 - V = 0, c = 1,5. Zdroj: vlastní.



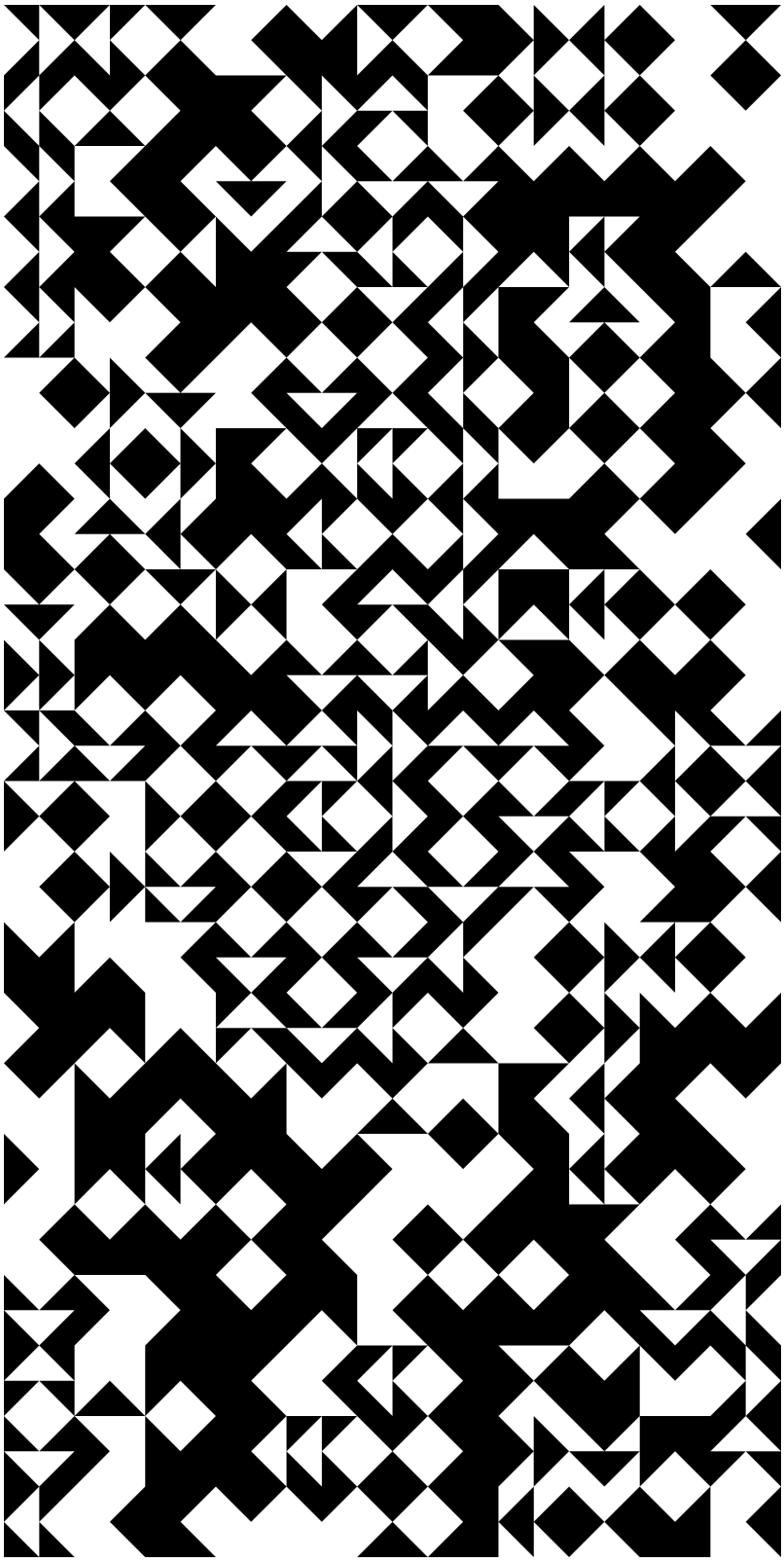
Obrázek 48: K10 - V = 1, c = 1,5. Zdroj: vlastní.



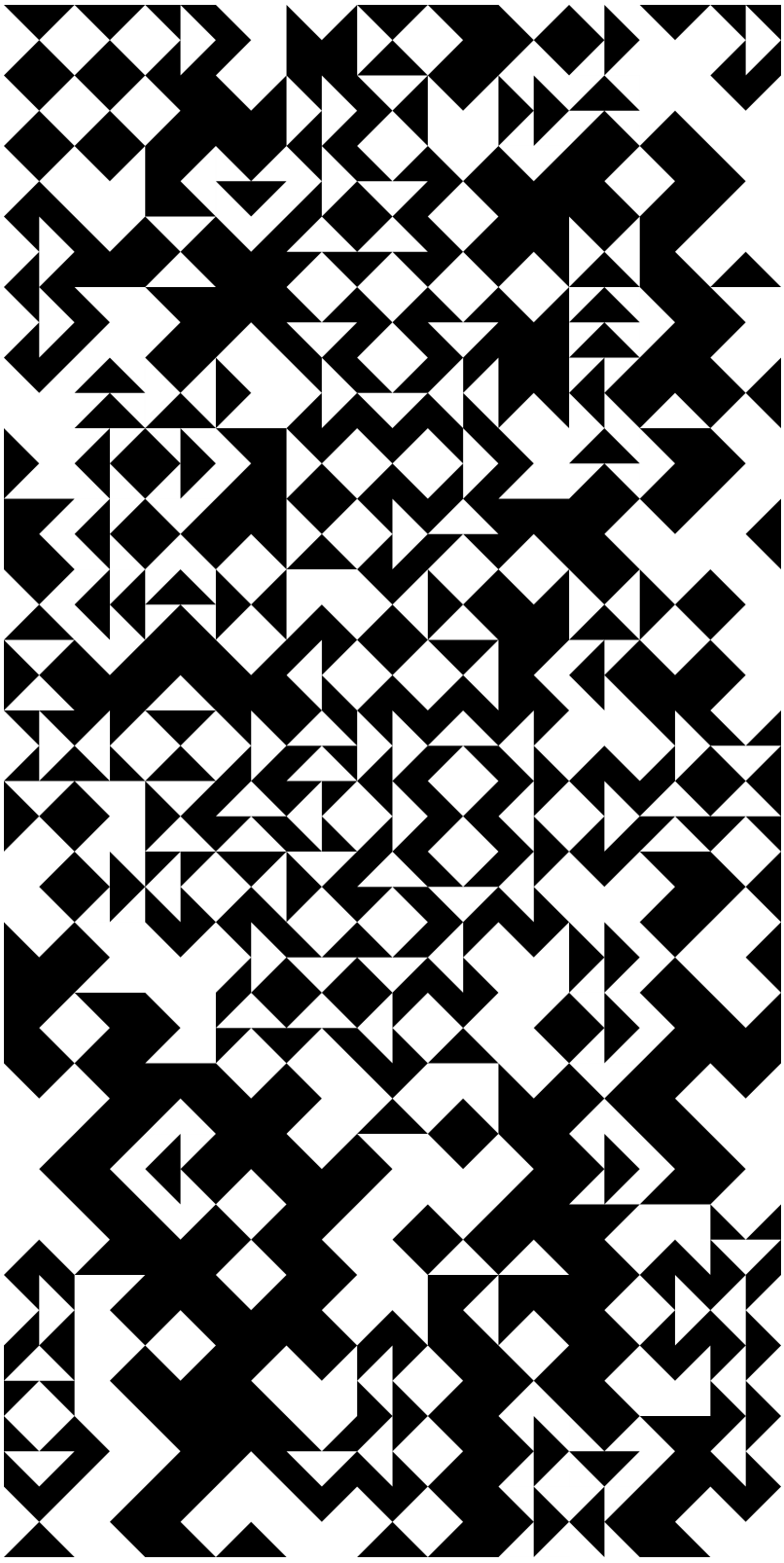
Obrázek 49: K11 - V = 2, c = 1,5. Zdroj: vlastní.



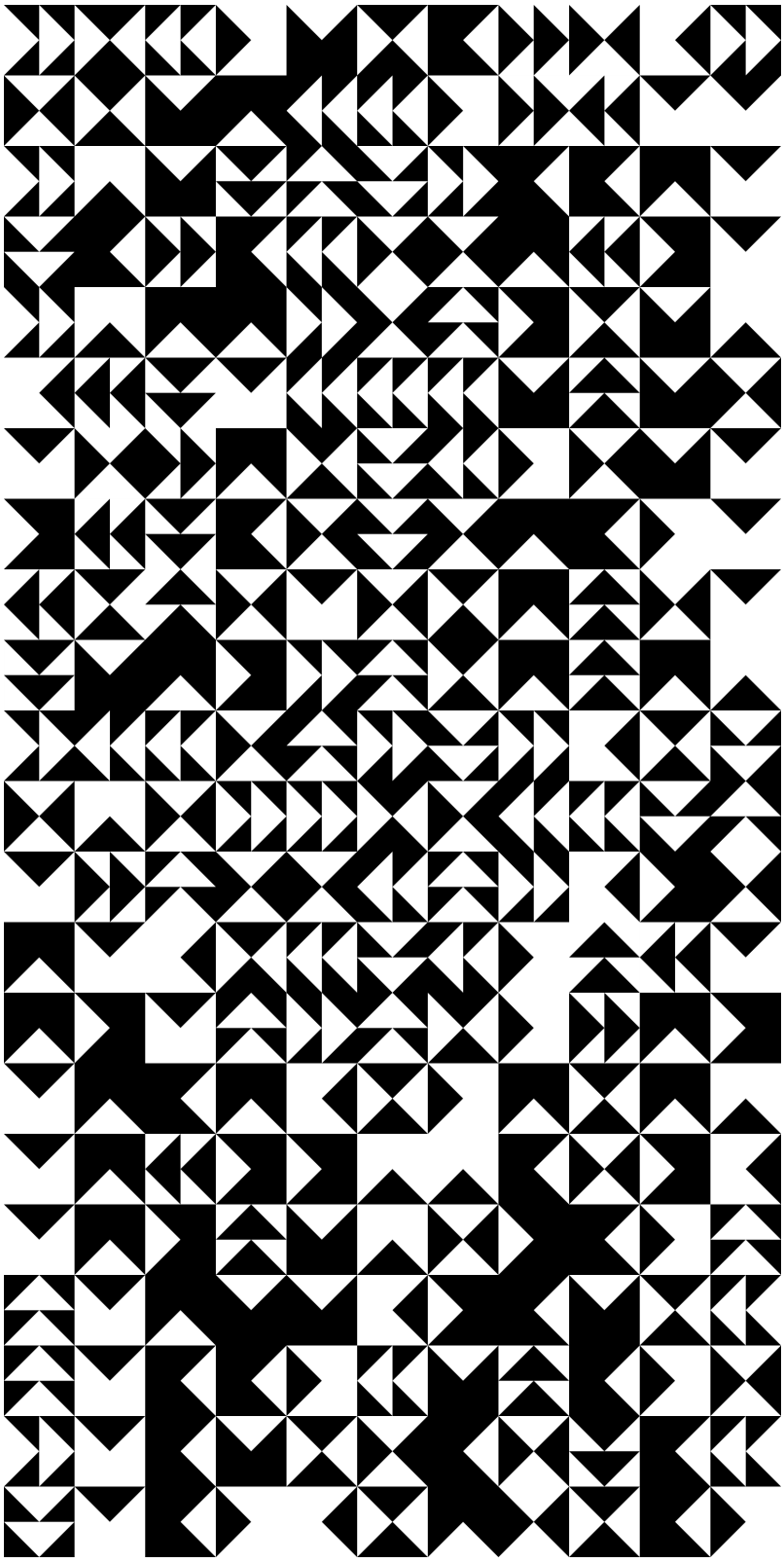
Obrázek 50: K12 - V = 3, c = 1,5. Zdroj: vlastní.



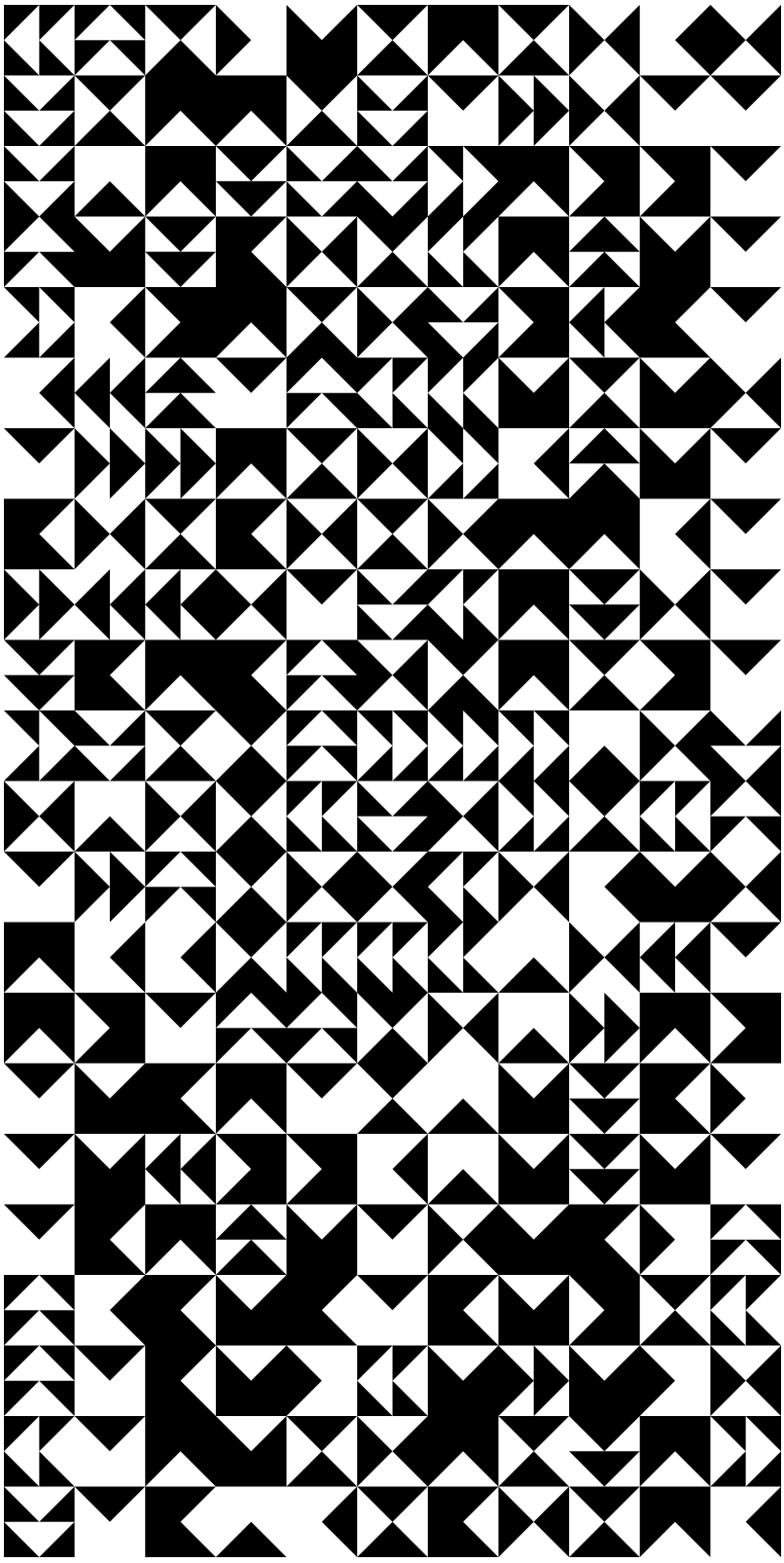
Obrázek 51: K13 - V = 0, c = 0,75. Zdroj: vlastní.



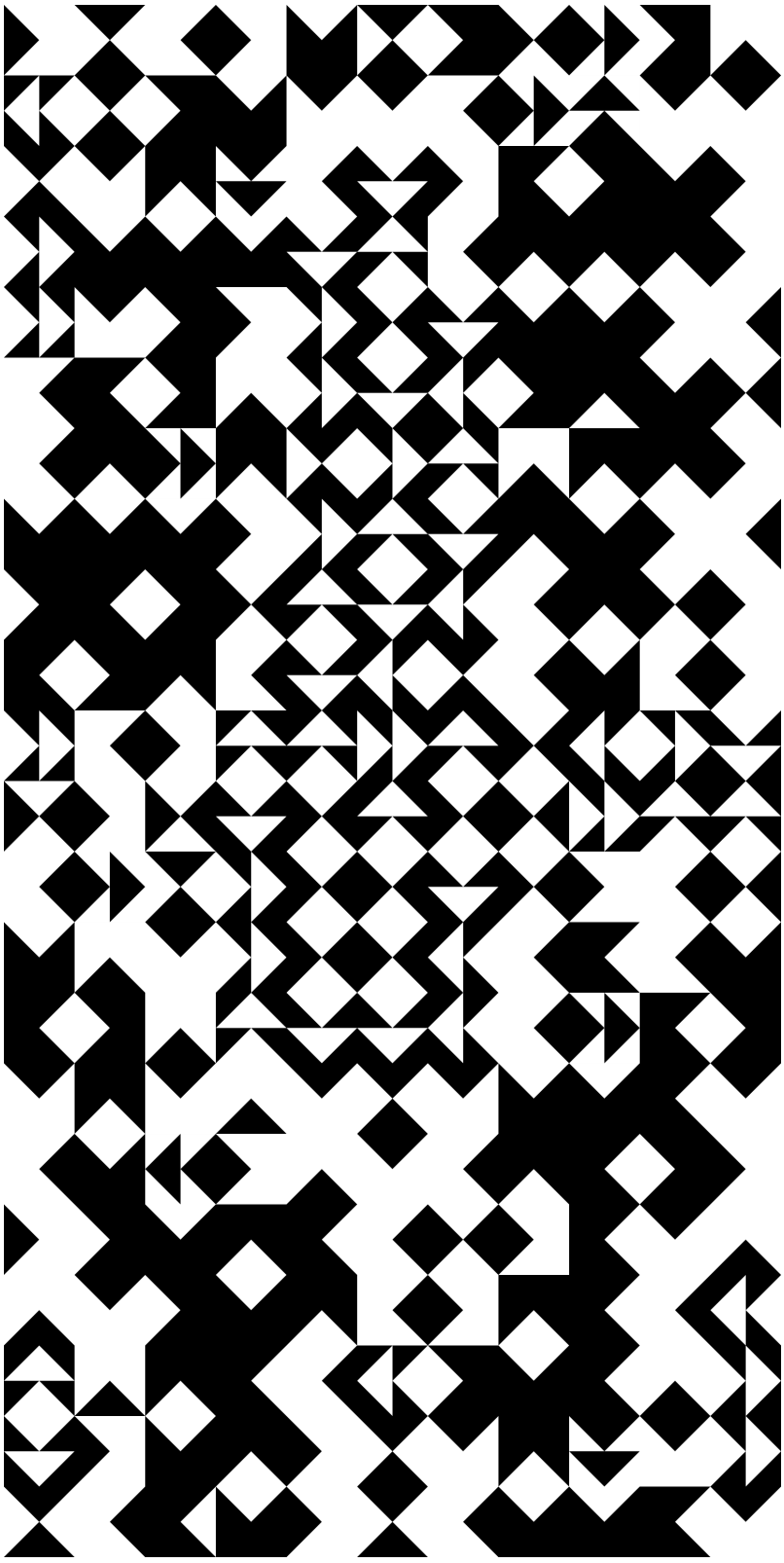
Obrázek 52: K14 - V = 1, c = 0,75. Zdroj: vlastní.



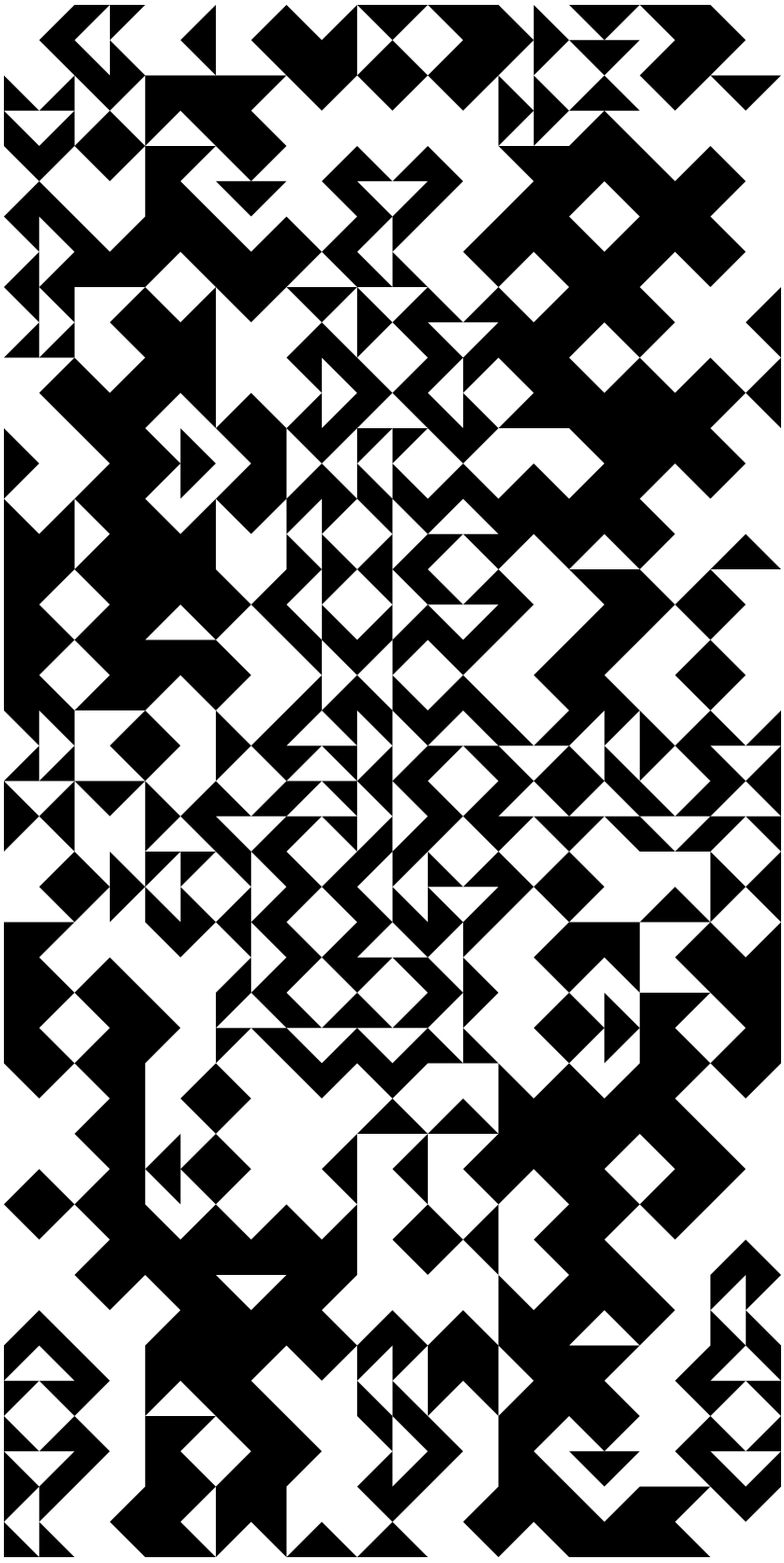
Obrázek 53: K15 - V = 2, c = 0,75. Zdroj: vlastní.



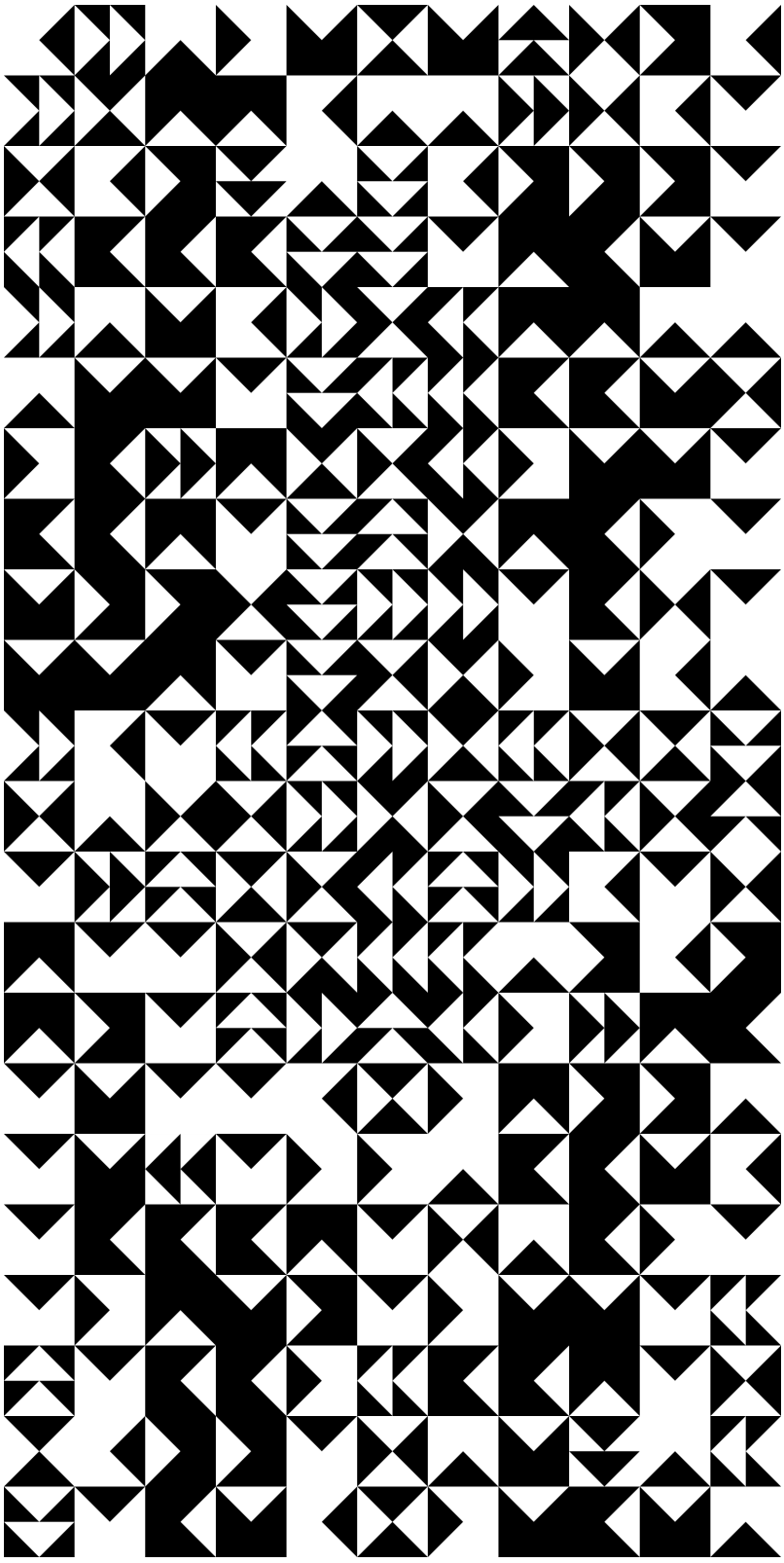
Obrázek 54: K16 - V = 3, c = 0,75. Zdroj: vlastní.



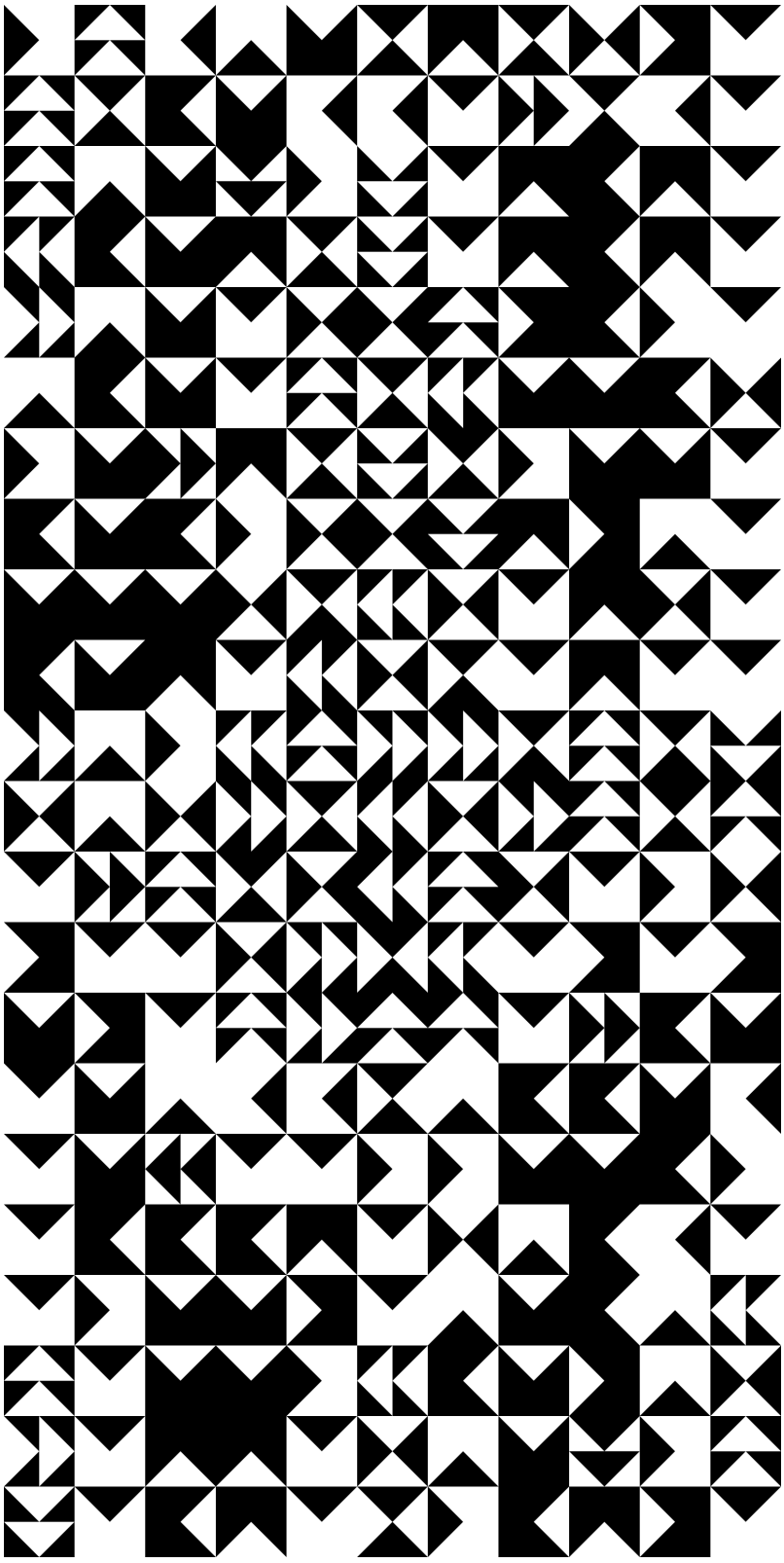
Obrázek 55: K17 - V = 0, c = 0,5. Zdroj: vlastní.



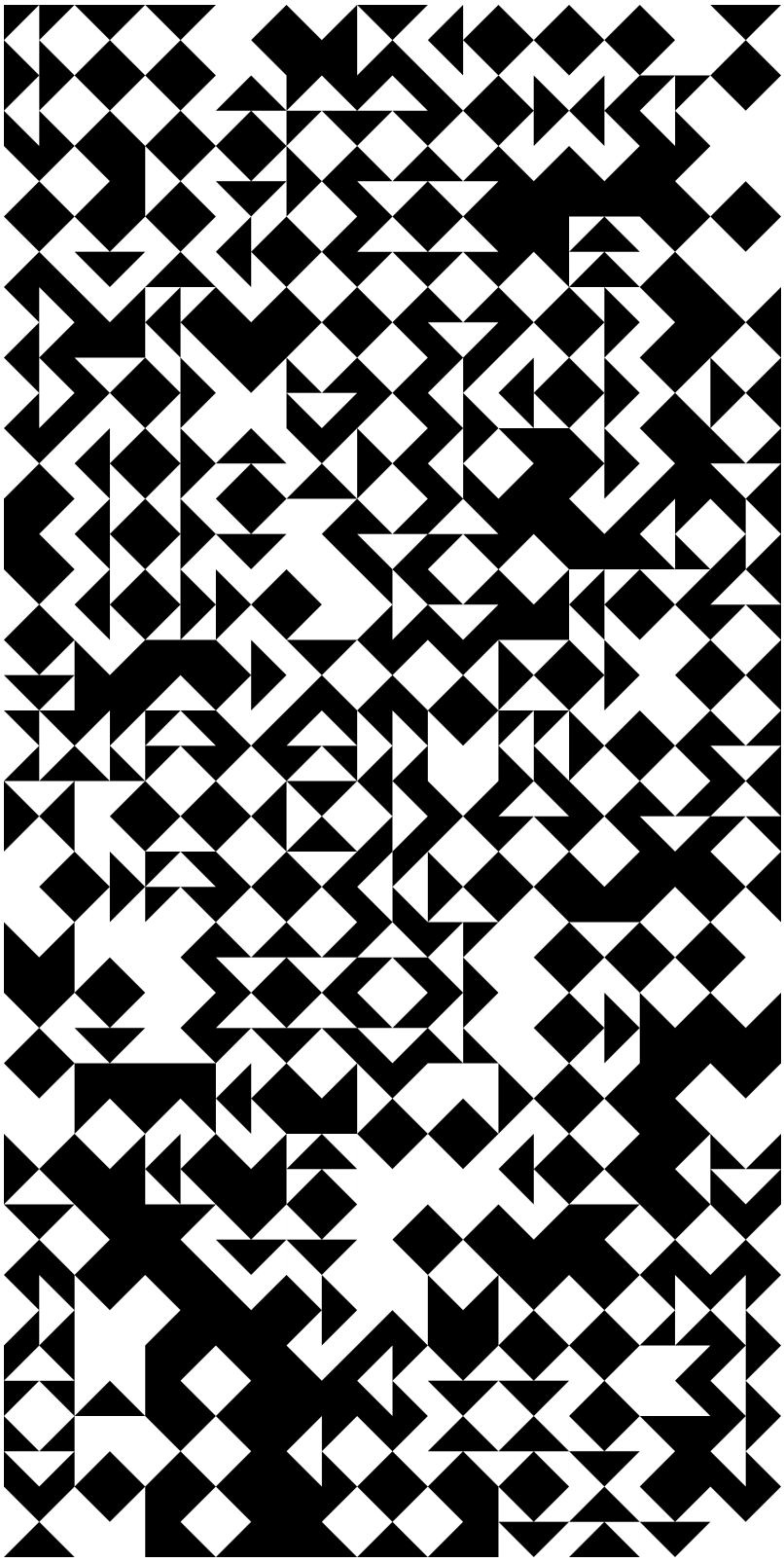
Obrázek 56: K18 - V = 1, c = 0,5. Zdroj: vlastní.



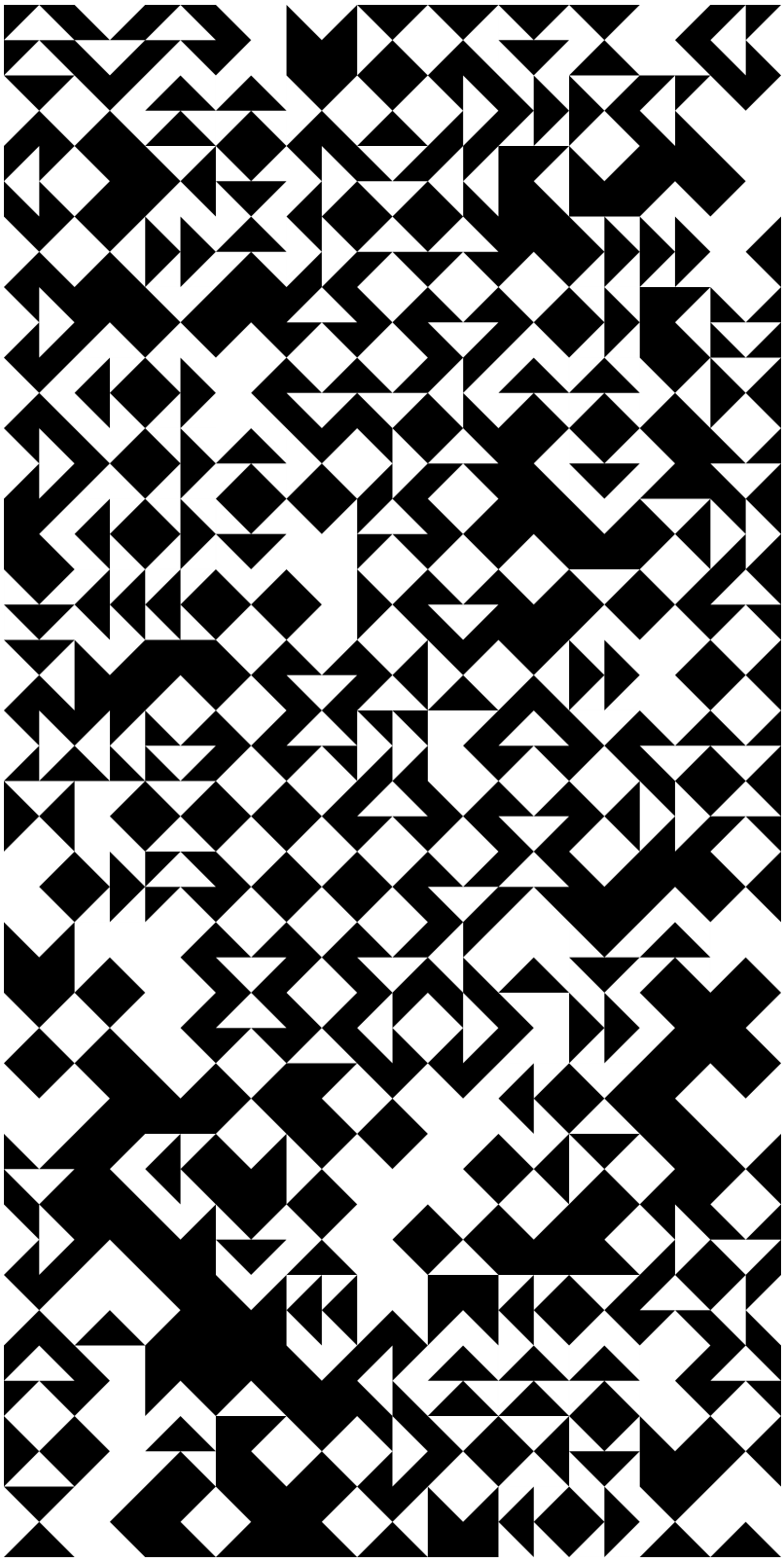
Obrázek 57: K19 - V = 2, c = 0,5. Zdroj: vlastní.



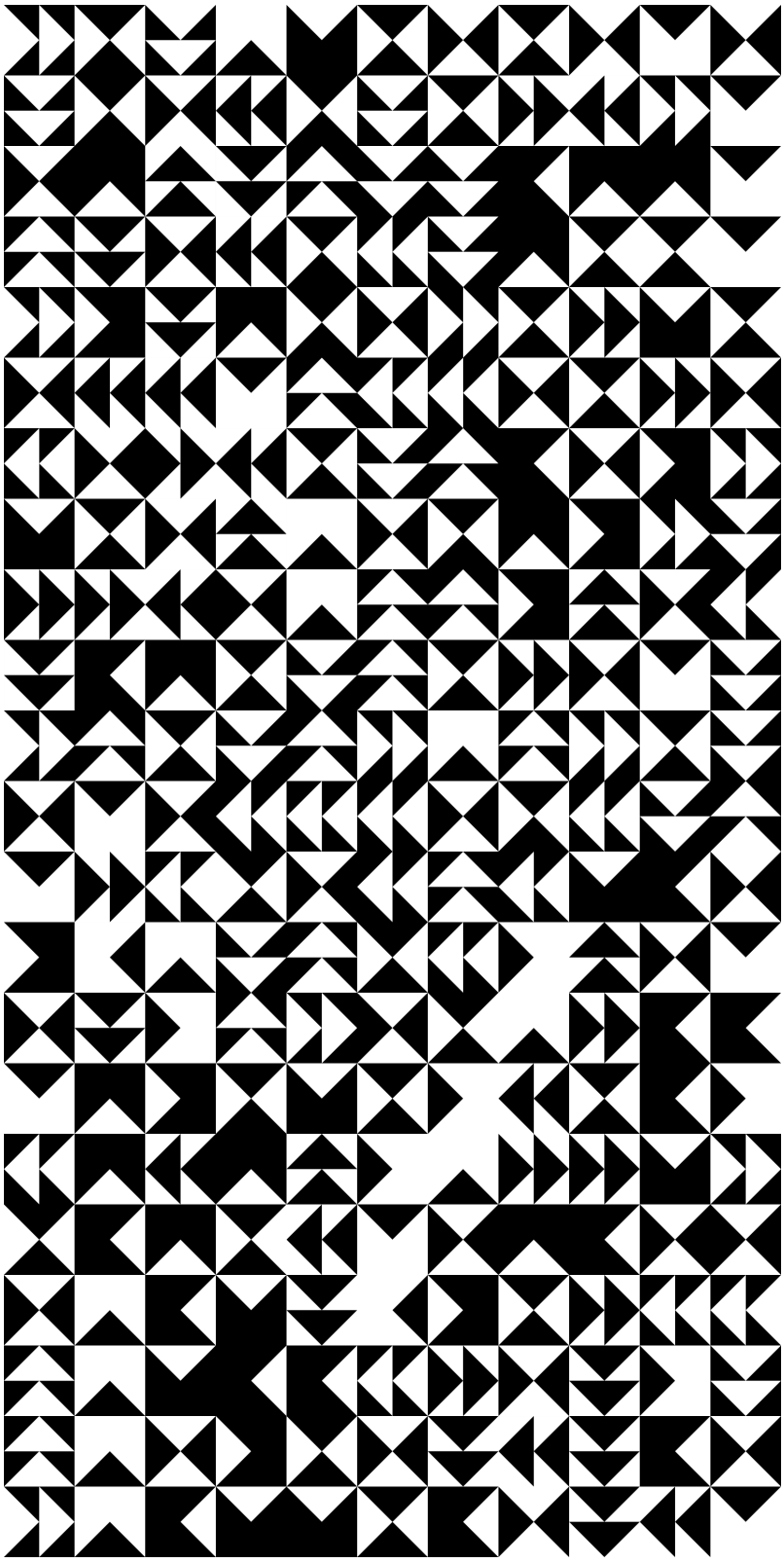
Obrázek 58: K20 - V = 3, c = 0,5. Zdroj: vlastní.



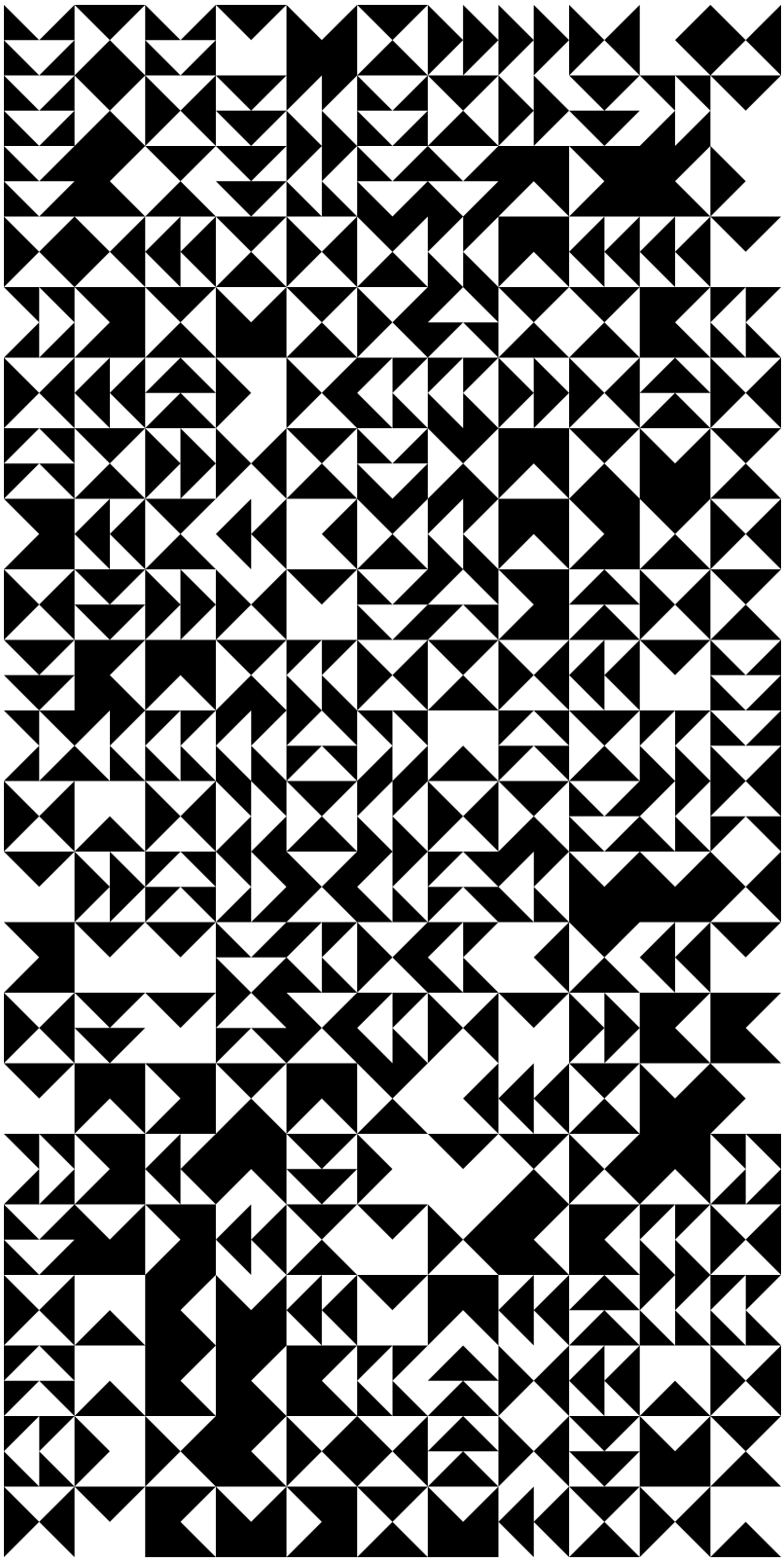
Obrázek 59: K21 - V = 0, c = 1,5. Zdroj: vlastní.



Obrázek 60: K22 - V = 1, c = 1,5. Zdroj: vlastní.



Obrázek 61: K23 - V = 2, c = 1,5. Zdroj: vlastní.



Obrázek 62: K24 - V = 3, c = 1,5. Zdroj: vlastní.

Obsah CD

Obsah přiloženého CD je následující:

- soubor `dp_Hora_Jiri_Zdenek_Sykora_Struktury.pdf` – elektronická verze práce,
- adresář `projekt` – kompletní projekt z Microsoft Visual Studio 2017 obsahující veškeré zdrojové soubory,
- adresář `aplikace` – spustitelná aplikace s potřebnými soubory. Jsou zde umístěny i definice Struktur a složky s obrázky elementů,
- adresář `generovani` - vygenerované Struktury K1 - K24 a jejich partitury.