VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VÍCEDIMENSIONÁLNÍ FORMÁLNÍ MODELY A JEJICH APLIKACE VE VIZUÁLNÍM UMĚNÍ

MULTI-DIMENSIONAL AUTOMATA AND THEIR APPLICATIONS IN VISUAL ART

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTINA ZLEVOROVÁ**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Studentka: **Zlevorová Martina**
Program: Informační technologie
Název: **Vícedimensionální formální modely a jejich aplikace ve vizuálním umění**
Multi-Dimensional Automata and Their Applications in Visual Art
Kategorie: Teoretická informatika

Zadání:

1. Dle instrukcí vedoucího se seznamte s různými vícedimensionální gramatikami a automaty.
2. Zaveďte nové verze těchto automatů dle instrukcí vedoucího.
3. Studujte vlastnosti těchto automatů a jazyků dle instrukcí vedoucího.
4. Dle instrukcí vedoucího se seznamte s výtvarným uměním, např. koláže od Ladislava Nováka, Ray Yoshida či John Wilde. Navrhněte aplikaci modelů z bodu 2 ve výtvarném umění dle instrukcí vedoucího.
5. Implementujte aplikace navržené v bodě 4.
6. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 26. října 2021

Abstrakt

Tato práce se věnuje propojení teoretické informatiky a výtvarného umění. Demonstruje využití formálních modelů při tvorbě nového druhu umění aplikováním dvoudimenzionálního automatu na znaky nalezené v obraze.

Abstract

This thesis deals with the connection between theoretical computer science and fine arts. It demonstrates the use of formal models in the creation of a new type of art by applying a twodimensional automaton on symbols found in an image.

Klíčová slova

konečný automat, vícedimenzionální automat, teoretická informatika, výtvarné umění

Keywords

finite automata, multidimensional automata, theoretical computer science, fine arts

Citace

ZLEVOROVÁ, Martina. *Vícedimensionální formální modely a jejich aplikace ve vizuálním umění*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Vícedimensionální formální modely a jejich aplikace ve vizuálním umění

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením prof. RNDr. Alexandra Meduny, CSc. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Martina Zlevorová
5. května 2022

Poděkování

Tímto bych chtěla poděkovat svému vedoucímu prof. RNDr. Alexandrovi Medunovi, CSc. za inspiraci, rady a trpělivost, se kterou mi pomohl dokončit tuto práci. Dále děkuji své rodině a přátelům za psychickou podporu.

Obsah

Kapitola 1

Úvod

Tato práce má za cíl demonstrovat využití formálních modelů při tvorbě vizuálního umění. Část formálních modelů je zastoupena dvoudimenzionálním automatem, jehož vstupem je dvoudimenzionální pole, které se do roviny umění nejlépe promítá na statický obraz. Převod vstupního obrazu na pole automatu lze realizovat podle algoritmu rozpoznávání textu v obraze.

Tvorbě umění za pomoci automatu se v minulosti věnoval Mário Gažo, který ve své bakalářské práci využil dvoudimenzionální automat k řízení pohybu znaků na obraze. Rozšíření, které představí má práce, spočívá v definování nových efektů tvořících základ nového automatu a ve změně povahy symbolů, se kterými automat pracuje. Nalezením znaků tvořících prvky automatu ve vstupním obraze je docíleno hlubšího propojení mezi uměním a automatem.

Následujících kapitoly se věnují teoretické stránce problematiky, návrhu praktického vypracování, popisu implementace a zhodnocení výsledků. Nejdříve jsou vysvětleny základní pojmy vztahující se k formálním modelům, dále jsou uvedeny definice konečných jednodimenzionálních a dvoudimenzionálních automatů a jejich konkrétní příklady.

Poznatky z první kapitoly jsou ve druhé využity při definici nového typu dvoudimenzionálního automatu, který řídí efekty pracující s písmeny a číslicemi.

V další kapitole je předveden význam textu ve vizuálním umění na dílech různých autorů, kteří se snaží svou tvorbou přimět pozorovatele k hlubšímu zamyšlení.

Dále je popsán proces rozpoznávání textu v obraze, který je základem propojení vizuálního umění s automatem. Jsou představeny kroky vedoucí ke zpřesnění výsledku, jenž tvoří jeden ze vstupních prvků navrženého automatu.

V následující části práce je popsáno konkrétní programové řešení a jeho rozdělení na moduly. U jednotlivých modulů jsou uvedeny metody, které definují, a data, se kterými pracují.

Dále je zdokumentován proces testování, který vede ke zhodnocení výsledků rozpoznávacího modulu a samotného automatu. Výsledky jsou jasně patrné na obrázcích, které jsou výstupem programu a představují znaky nalezené pomocí rozpoznávacího algoritmu a aktuální stav automatu.

Celá práce je uzavřena kapitolou shrnující proces vypracování a z něj nabyté poznatky. Je v ní zhodnocen výstup aplikace a navrženy způsoby, kterými by se dala práce v budoucnu rozšířit.

Kapitola 2

Formální modely

Formální modely jsou přesným zápisem komponent a vztahů mezi nimi. Jedná se o silné nástroje nabízející zjednodušení zápisu reálných systémů, které se využívá při návrhu či verifikaci ve vývoji systémů, ať už softwarových, nebo hardwarových. Nejběžnější podoby, se kterými se při vývoji softwaru setkáváme, jsou gramatiky a automaty. Gramatiky nachází uplatnění při definici programovacích jazyků, kde zastupují formální jazyk, což je abstrakce základních vlastností reálného jazyka. Automaty lze využít při modelování systému, který na základě vstupu mění svůj stav. Položení základů teorie automatů se datuje kolem roku 1930, kdy Alan Turing vymyslel abstraktní model, který odpovídá i moderním počítačům. Výzkum gramatik byl odstartován v padesátých letech, hlavně díky úsilí lingvisty Noama Chomského, který byl z prvních, kdo je začal studovat. [13]

V této kapitole jsou uvedeny definice základních pojmů souvisejících s formálními modely, zejména gramatikami a automaty. Jednotlivé komponenty jsou propojeny do souvislostí a nakonec jsou představeny i některé konkrétní modely automatů, které slouží jako inspirace pro návrh nového automatu v kapitole 3.

2.1 Základní pojmy

Následuje vysvětlení některých pojmů, se kterými se při tvorbě formálních modelů setkáváme.

Abeceda

Abeceda je konečná neprázdná množina znaků. Nejčastěji se označuje řeckým písmenem Σ. Mezi nejpoužívanější abecedy patří: [13]

- **binární** $\Sigma = \{0, 1\}$
- **malá písmena** $\Sigma = \{a, b, \ldots, z\}$
- **ASCII** znaky amerického standardního kódu pro výměnu informací

Slovo

Slovem se rozumí konečná sekvence symbolů náležející do jedné abecedy. Například 00 nebo 11001 jsou slova sestavená ze znaků binární abecedy. Množina všech slov, která jdou sestavit nad abecedou Σ, je uzávěr vůči operaci zřetězení, značený Σ^*. [13]

Jazyk

Jazyk je podmnožinou Σ^*, což je množina všech slov sestavených z abecedy Σ. Matematicky zapsáno $L \subseteq \Sigma^*$, čteme L je jazyk nad abecedou Σ. Této definici odpovídají běžně používané jazyky, například angličtina, což je jazyk nad latinskou abecedou, nebo programovací jazyky, které bývají sestaveny nad abecedou ASCII. V problémech řešených pomocí modelů se můžeme setkat i s abstraktními jazyky, jako třeba L nad binární abecedou, které je tvořeno slovy začínajícími 0, obsahujícími posloupnost 11. [13]

Gramatika

Gramatiky propojují studium jazyků s matematikou. Představují formální reprezentaci syntaxe jazyka, založenou na rekurzi, pomocí pravidel a daných symbolů. Matematicky je základ gramatiky čtveřice $G = \{V, T, S, P\}$, kde: [13]

- V je konečná množina proměnných, které představují jazyky, tedy množiny slov
- T je konečná množina terminálů, které odpovídají jednotlivým slovům
- $S \in V$ je počáteční proměnná značící celý definovaný jazyk
- P je konečná množina pravidel, která jsou tvaru $x \longrightarrow y; x \in (V \cup T)^+, y \in (V \cup T)^*$

Tato čtveřice odpovídá bezkontextové gramatice, což je jeden z typů gramatik. V hierarchii podle Chomského je na 2. úrovni, takže definuje jazyk, který je nadmnožinou jazyka definovaného pomocí regulární gramatiky (3. úroveň) a podmnožinou jazyků definovaných gramatikou kontextovou na 1. úrovni a na 0. gramatikou neomezenou. [18]

Regulární výraz

Regulární výrazy jsou další alternativou pro definici jazyků. Silou odpovídají regulárním gramatikám a deterministickým konečným automatům, od nichž se liší zejména jednoduchostí zápisu, který lze využít například jako vstup systémů, které vyhledávají v textu. Příkladem je UNIXový příkaz grep, který regex, jak lze označit regulární výraz, převede na konečný automat a s jeho pomocí vyhledá zadané řetězce. [13] Pro příklad uvádím regex, který přijímá jazyk L nad binární abecedou, který je tvořený slovy začínajícími 0, obsahujícími posloupnost 11:

$$0(0+1)^*11(0+1)^*$$

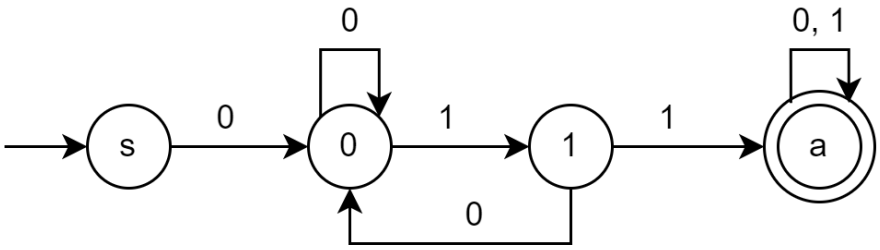
Prvky vyskytující se ve výrazu jsou z množiny abecedy $\{0, 1\}$ a řídících znaků $\{(,), ., +, ^* \}$. Znaky abecedy odpovídají podmnožinám abecedy, například 0 je $\{0\}$, závorky označují regex, ke kterému se váže řídící znak za pravou závorkou, plus je sjednocení, tečka (v zápisu vynechána) konkatenace a Kleeneho hvězdička vytváří množinu uzavřenou vůči operaci zřetězení obsahující prázdný řetězec. [18]

Automat

Automat je abstraktní model počítače, který zjednodušuje proces jeho činnosti na sadu stavů a přechodů mezi nimi. **Stav** reprezentuje relevantní část historie systému, ať už se jedná o interní proměnné, nebo polohu čtecí hlavice ve vstupním řetězci. Stav spolu se znakem na pozici čtecí hlavice tvoří **konfiguraci**. **Přechody** jsou definovány konfigurací, která je umožňuje, a stavem, ve kterém přechod končí. Pokud je každý přechod automatu

jednoznačně identifikovatelný konfigurací, jedná se o **deterministický** automat. V opačném případě je označen jako **nedeterministický**. Další dělení automatů je na **akceptační**, jejichž výstupem je odpověď ANO, nebo NE, a **převodníkové**, které generují výstup v podobě řetězce symbolů. [13, 18]

Příklad na obrázku 2.1 ukazuje diagram automatu odpovídající regexu z příkladu v předešlé části. Automat akceptuje slova tvořená binární abecedou, která začínají 0 a obsahují posloupnost 11.



Obrázek 2.1: Automat

Navržený automat prochází čtyřmi stavy, označenými {s, 0, 1, a}. S je vstupní, šipkou označený počáteční stav, a je koncový stav, zvýrazněný rámečkem, a symboly u šipek přechodů značí povolující vstup.

2.2 Konečný automat

Obecný jednodimenzionální konečný automat je čtveřice $M = (Q, \Sigma, R, s, F)$, pro kterou platí: [20]

- Q je konečná množina stavů automatu
- Σ je vstupní abeceda reprezentovaná neprázdnou konečnou množinou symbolů
- $R \subseteq Q \times \Sigma^* \times Q$ je konečná množina tranzitivních pravidel, které lze zapsat ve tvaru

$$py \longrightarrow q; p, q \in Q, y \in \Sigma$$

- s je počáteční stav automatu ($s \in Q$)
- F je konečná množina koncových stavů automatu ($F \subseteq Q$)

Automat začíná v počátečním stavu a do dalšího stavu přechází na základě pravidla z množiny R a aktuální konfigurace. Konfigurace je dvojice (p, a), kde $p \in Q, a \in \Sigma^*$, p je aktuální stav automatu a a je vstupní řetězec. Pokud existuje pravidlo $pa \longrightarrow q$, přejde automat při přečtení řetězce a ze stavu p do stavu q. Řetězec, při jehož přečtení automat přejde z počátečního stavu do jednoho z koncových, patří do množiny jazyka, který tento automat přijímá. Takový jazyk se označuje L(M). [20]

2.2.1 Dobře specifikovaný konečný automat

Označení dobře specifikovaný se užívá pro automaty, které v praxi vždy pracují správně a nehrozí u nich nejasný výsledek. Příčinou nejasnosti můžou být následující vlastnosti. [14]

Nejednoznačná pravidla Pokud jsou v definici automatu alespoň dvě pravidla pro jednu konfiguraci, která popisují přechod do různých stavů, není v této konfiguraci jasné, kterým pravidlem se má automat řídit a do kterého stavu má tedy přejít.
Př. $pa \longrightarrow q, pa \longrightarrow r \in R$

ϵ-přechody V definici automatu se vyskytují pravidla, která mu umožňují přejít do následujícího stavu bez přečtení symbolu na vstupu. U takového automatu není jasné, zda mají ϵ-přechody přednost před pravidly, pro jejichž uplatnění je zapotřebí přečíst symbol ze vstupního řetězce.

Nedostupné stavy Jsou takové, do kterých se automat nemůže z počátečního stavu dostat přečtením žádného řetězce.

Neukončující stavy Jsou stavy, ze kterých se nejde dostat do koncového stavu. Neexistuje posloupnost pravidel, která by vedla z neukončujícího stavu do koncového, ať už je na vstupu jakýkoliv řetězec.

2.3 Dvoudimenzionální konečný automat

Jedná se o jednodimenzionální konečný automat rozšířený o možnost číst na vstupu symboly vstupní abecedy uspořádané do dvoudimenzionálního pole. Krajní buňky vstupního pole jsou vyplněny hraničním symbolem, který nepatří do množiny vstupní abecedy. Pro jednoznačné označení buněk lze použít indexy $[< 0, m >, < 0, n >]$. Indexace použitá v tomto projektu je znázorněna na obrázku 2.2. Buňka s indexem $[0, 0]$ je v levém horním rohu, protože tento bod využívá většina frameworků pro tvorbu grafického uživatelského rozhraní, např. Qt. [30]

$$\begin{array}{ccccccc}
\overset{[0,0]}{\#} & \# & \# & \dots & \# & \overset{[m,0]}{\#} \\
\# & a_{1,1} & a_{2,1} & \dots & a_{m-1,1} & \# \\
\# & a_{1,2} & a_{2,2} & \dots & a_{m-1,2} & \# \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\# & a_{1,n-1} & a_{2,n-1} & \dots & a_{m-1,n-1} & \# \\
\overset{[0,n]}{\#} & \# & \# & \dots & \# & \overset{[m,n]}{\#}
\end{array}$$

Obrázek 2.2: Vstupní pole dvoudimenzionálního automatu

Obecně je dvoudimenzionální automat šestice $M = (Q, \Sigma, \delta, q_0, q_A, q_R)$, pro kterou platí: [28]

- Q je konečná množina stavů
- Σ je vstupní abeceda (hraniční symbol $\# \notin \Sigma$)
- δ je tranzitivní funkce

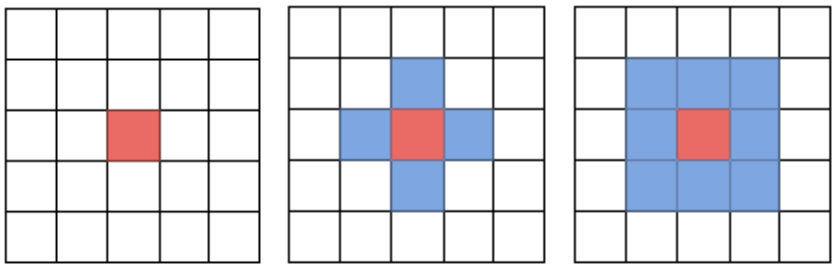
- q_0 je počáteční stav automatu ($q_0 \in Q$)
- q_A je akceptační stav automatu ($q_A \in Q$)
- q_R je odmítající stav automatu ($q_R \in Q$)

Automat začíná v počátečním stavu a čtením vstupního pole přechází do dalšího na základě tranzitivní funkce. Dostane-li se do akceptačního stavu, přijme automat vstupní pole, pokud je aktuální stav odmítající, vstupní pole je zamítnuto. Oba případy vedou ke správnému ukončení chodu automatu. Podle tranzitivní funkce rozlišujeme několik základních typů automatů.

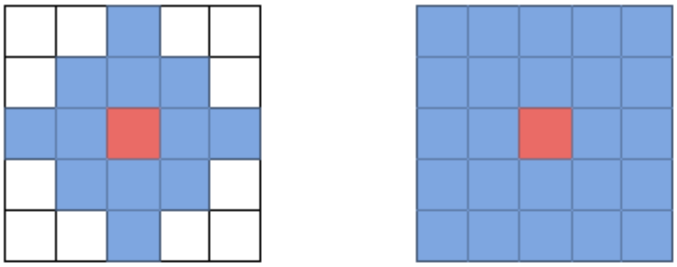
2.3.1 Celulární automat

Jde o speciální typ dvoudimenzionálního konečného automatu, ve kterém při každém kroku dojde ke změně hodnot všech buněk. Následující hodnota buňky závisí na tranzitivní funkci, jejímž vstupem je aktuální hodnota dané buňky a hodnoty buněk v jejím okolí.

Mezi základní typy okolí patří prázdné, Von Neumannovo, Moorovo a jejich rozšířené verze [25]. Prázdné okolí nezahrnuje žádnou buňku, takže následující hodnota závisí pouze na té předchozí, Von Neumannovo bere v potaz čtyři buňky v základních směrech, Moorovo zahrnuje osm buněk – čtyři v základních směrech a čtyři v diagonálních – a rozšířené verze jak Von Neumannova tak Moorova okolí vznikají zvětšením poloměru jejich základních verzí.



(a) Prázdné (b) VonNeumannovo (c) Moorovo



(d) VonNeumannovo (r=2) (e) Moorovo (r=2)

Obrázek 2.3: Nejznámější okolí

Nejznámější příklad celulárního automatu je hra Life, jejíž pravidla vymyslel John Conway a poprvé se na veřejnost dostala roku 1970 ve článku Martin Gardnera pro časopis Scientific American [8]. Všechny články tohoto autora vyšly v knize The Colossal Book of Mathematics: Classic Puzzles, Paradoxes, and Problems [9] roku 2001 a z té je následující definice.

The Game of Life

- *Hrací pole je nekonečná rovina rozdělená na stejně velké čtverce.*
- *V každém kroku je určen nový stav všech buněk na základě jejich stavu před započetím kroku, krok představuje jednu generaci.*
- *Možné stavy jsou plný a prázdný, neboli živý a mrtvý.*
- *Nový stav buňky závisí na osmi sousedních buňkách tvořících její Moorovo okolí.*
- *Pravidla pro zrod (zaplnění) a smrt (vyprázdnění) jsou tři:*
  - *Přežití - Každá živá buňka se dvěma nebo třemi živými buňkami ve svém okolí přežije do další generace (kroku).*
  - *Úmrtí - Každá živá buňka s více než třemi živými sousedními buňkami umírá. Stejně tak umírají buňky s méně než dvěma živými sousedy.*
  - *Narození - V každé mrtvé buňce, v jejímž okolí se vyskytují právě tři živé buňky, dojde ke zrodu.*

Tato pravidla Conway sestavil tak, aby nebylo snadné z počáteční konfigurace odvodit, zda se bude populace rozrůstat donekonečna, nebo po několika krocích vymře, nebo zda se vyvine do stálé, či oscilující fáze.

2.3.2 Čtyřcestný konečný automat

Čtyřcestný automat je typ dvoudimenzionálního automatu specifický tím, že omezuje pohyb čtecí hlavice na čtyři směry a vzdálenost jedné buňky od aktuální pozice. Pokud hlavice dojde k hraničnímu symbolu, může ji automat vrátit opačným pohybem dovnitř pole. V knize The Handbook of Formal Languages [10] je čtyřcestný automat definován jako sedmice $M = (Q, \Sigma, \delta, \Delta, q_0, q_A, q_R)$, jejíž prvky odpovídají definici dvoudimenzionálního automatu v kapitole 2.3 s výjimkou množiny $\Delta = \{R, L, U, D\}$, která obsahuje možné směry pohybu čtecí hlavice, v tomto případě označených podle jejich anglických názvů: Right (doprava), Left (doleva), Up (nahoru), Down (dolů). Výstupem tranzitivní funkce je kromě následujícího stavu také směr pohybu hlavice. Vykonání funkce automatu je ukončeno, ocitne-li se v jednom z akceptačních nebo odmítajících stavů.

Kapitola 3

Návrh dvoudimenzionálního automatu

Nový typ dvoudimenzionálního automatu, představený v této kapitole, vychází z obecné definice dvoudimenzionálního automatu a je inspirován jeho konkrétními příklady. V této kapitole jsou stanoveny prvky šestice, která tvoří dvoudimenzionální konečný automat, s úpravami odpovídajícími cílům této práce. Navržený automat vychází z počátečního stavu určeného vstupním obrazem a na něm se nacházejícími znaky latinské abecedy a číslicemi, které tvoří vstupní abecedu automatu. Přechody mezi stavy jsou řízeny aktuálně zvolenou tranzitivní funkcí a činnost automatu je ukončena při přechodu do koncového stavu. Níže jsou uvedeny konkrétní postupy, které tvoří tělo automatu.

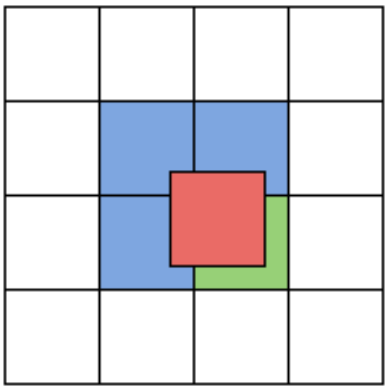
3.1 Počáteční stav

Stav automatu je určen pozicí prvků v poli automatu. Toto pole je rozděleno do stejně velkých obdélníkových buněk, jejichž velikost závisí na velikosti znaků nalezených v obraze a jejichž počet dále závisí na velikosti vstupního obrazu. Pro inicializaci stavu automatu je tedy potřeba obraz a pole znaků se známými pozicemi a velikostmi. Algoritmus získání pole znaků je popsán v kapitole 5. V této části je předveden návrh pro vytvoření vstupního pole automatu a stanovení počáteční příslušnosti prvků k buňkám.

Pole automatu je abstraktní, jeho reprezentace je tvořena čtyřmi hodnotami – výškou a šířkou buněk a jejich počtem, uloženým jako největší možné indexy na osách x a y. Buňky jsou jednoznačně určeny pomocí indexů do tohoto pole, takže jejich výběr je ve skutečnosti výběr dvou celých čísel v rozsahu 0 až maximální možná hodnota. Posun grafické podoby elementu do nové buňky je realizován pomocí výpočtu nové pozice jako násobků indexů buňky s její velikostí. Inicializace pole začíná výpočtem velikosti buněk jako aritmetického průměru velikostí znaků. Největší možné indexy jsou pak dané podílem velikosti vstupního obrazu a vypočtenou velikostí buněk.

Počáteční přiřazení buněk k nalezeným znakům je založeno na určení nejbližší buňky pomocí obsahu plochy, kterou sdílí buňka s obdélníkem obkresleným znaku. Z pole buněk jsou nejdříve vybrány čtyři, jejichž pozice vůči znaku odpovídají obrázku 3.1. Dají se získat například tak, že se nejdříve určí levá horní buňka nalezením bodu s nejvyššími hodnotami souřadnice v mřížce pole automatu, které jsou ale menší než hodnoty souřadnice levého horního rohu znaku. Tento nalezený bod určuje první možnou buňku, k níž jsou přidány

tři okolní buňky přičtením indexů. Z těchto čtyř možností je vybrána buňka, jejíž průnik s plochou obdélníku obkreslenému znaku je největší.



Obrázek 3.1: Prvotní přiřazení buňky - k červeně vyznačenému znaku je přiřazena zelená buňka. Modře jsou označeny zbylé tři možnosti.

3.2 Tranzitivní funkce

Změna stavu automatu je řízena tranzitivní funkcí. Specifikem navrženého automatu je, že nabízí pět možných funkcí, mezi nimiž může uživatel přepínat. Čtyři z těchto možností přesunují prvky a vytvářejí tak efekty popsané v této části. Jedná se o náhodný pohyb, který prvek posune do buňky v jeho okolí, jeho verze omezená podle typu znaku, permutaci všech prvků a nezávislý pohyb přesunující prvek na náhodnou pozici bez závislosti na jeho původní poloze. Všechny jsou složeny z kroků, které jsou elementární, nezávislé a pracují s náhodou, ať už při výběru symbolu, který přemístí, nebo při výběru jeho nové pozice. Pátá definovaná tranzitivní funkce nemění pozice prvků, pouze přepíná automat do koncového stavu a je pojmenována zastavovací.

Náhoda, se kterou funkce pracují, je reprezentována výběrem s rovnoměrným rozdělením pravděpodobnosti, pro který platí: [16]

$$p(x) = \begin{cases} \frac{1}{N} & \text{pro } x_i, i \in (0, N-1) \\ 0 & \text{jinak} \end{cases} \tag{3.1}$$

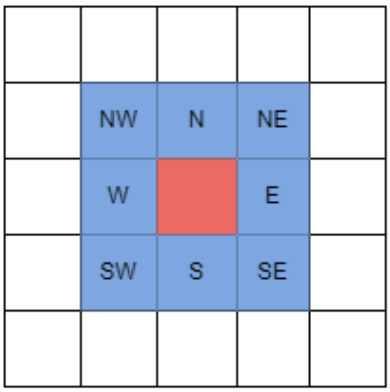
$$\sum_{i=0}^{N-1} p(x_i) = 1 \tag{3.2}$$

kde x je jev a N je počet možných jevů. Rovnoměrné rozdělení pravděpodobnosti znamená, že každý jev má stejnou pravděpodobnost. Typickým příkladem je hod šestihrannou kostkou, při kterém má každá hodnota asi 16,7% šanci padnout.

3.2.1 Náhodný efekt

Náhodný efekt pracuje s buňkami v bezprostředním okolí vybraného prvku. Poloměr okolí je 1 a z možných typů okolí, které jsou představeny v části 2.3.1, jsem vybrala Moorovo, které umožňuje pohyb do osmi okolních buněk. Pro zjednodušení označím možnosti výběru pro každou buňku stejně, a to podle směru pohybu. Všeobecně známé je označení podle

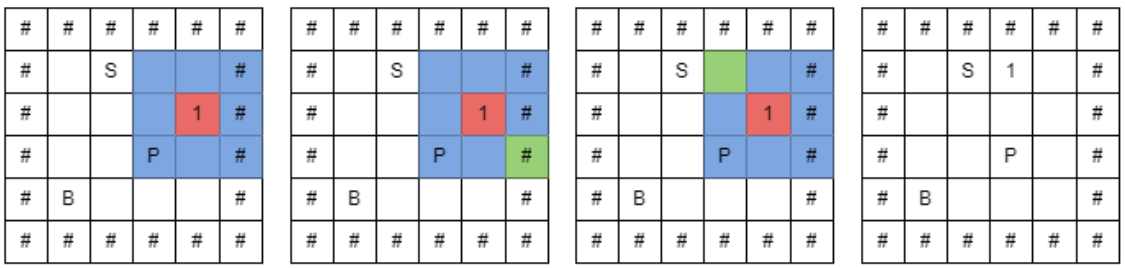
světových stran: North (sever), South (jih), East (východ), West (západ) a jejich kombinace pro diagonální pohyb. Označení je patrné na obrázku 3.2.



Obrázek 3.2: Označení buněk

V každém kroku automat přesune jeden symbol do buňky v jeho okolí. Prvek, který bude přesunut, stejně jako jeho cílovou buňku, určí náhodný výběr s rovnoměrným rozdělením pravděpodobnosti. Pokud ve vybrané cílové buňce již je nějaký symbol, zopakuje se výběr buňky.

1. *Nejprve je zvolen prvek, který se bude přemisťovat. Na obrázku 3.3a je to červeně označená 1, modrá barva znázorňuje okolí vybraného prvku.*
2. *Z okolí byla vybrána buňka, do které by se prvek měl přemístit. Na obrázku 3.3b je označená zeleně. Jde ale o hraniční buňku, je tedy zopakován výběr.*
3. *Na obrázku 3.3c je označená nově vybraná buňka. Je prázdná, takže vyhovuje a je do ní přemístěn prvek 1.*
4. *Obrázek 3.3d ukazuje stav po provedení jednoho kroku náhodného efektu.*



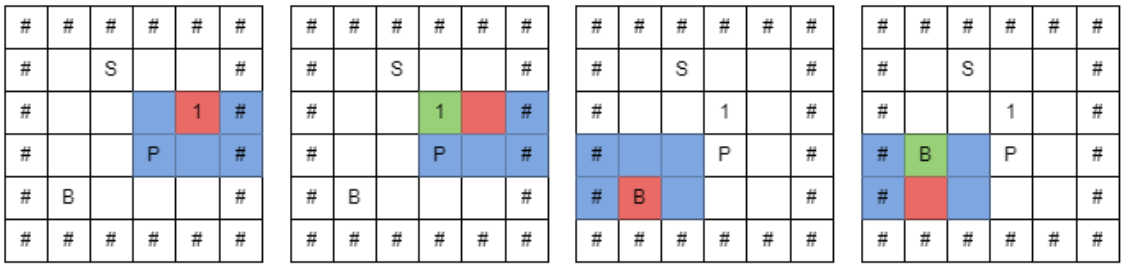
Obrázek 3.3: Náhodný efekt

3.2.2 Náhodný efekt s omezením

Od základní verze náhodného pohybu se liší tím, že omezuje okolí symbolů podle jejich zařazení ve vstupní abecedě. Automat přijímá latinskou abecedu a arabské číslice, což jsou

dvě skupiny, které se budou odlišně chovat. Konkrétně písmena nebudou smět cestovat dolů a číslicím se zakáže pohyb nahoru. Postupně se tak budou tyto skupiny od sebe vzdalovat a symboly třídit.

1. *Obrázek 3.4a představuje automat ve stavu po výběru prvku. Jedná se o červenou 1, jejíž okolí je zobrazeno modrou barvou. Je omezené tím, že se jedná o číslici.*
2. *Na obrázku 3.4b je automat po přesunutí 1 na nově zvolenou pozici.*
3. *Na obrázku 3.4c automat započal další krok, ve kterém zvolil prvek B. Jedná se o písmeno, takže omezení platí na buňky pod ním.*
4. *Na posledním obrázku 3.4d je symbol B přesunut do nově zvolené buňky.*



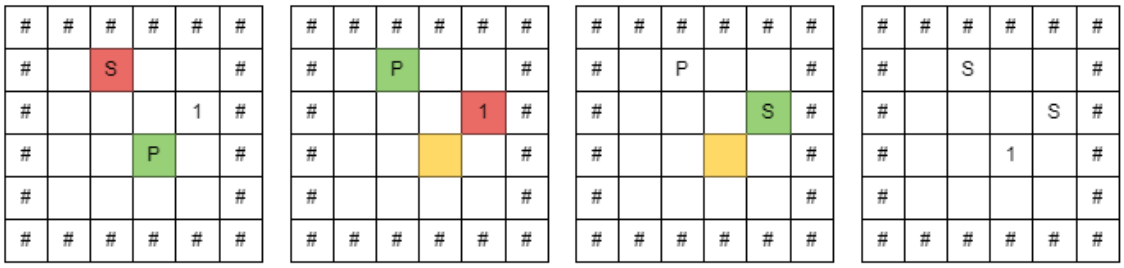
(a) (b) (c) (d)

Obrázek 3.4: Náhodný pohyb s omezením

3.2.3 Permutace

V jednom kroku automat provede permutaci množiny symbolů. Krok začíná výběrem symbolu, který se bude jako první přemisťovat. Nová pozice tohoto symbolu se určí výběrem druhého symbolu, do jehož buňky je umístěn první zvolený prvek. Novou buňku druhého prvku udá výběr třetího a automat tímto způsobem pokračuje, dokud neprojede celou množinu prvků. Poslední prvek umístí do buňky uvolněné prvně zvoleným prvkem.

1. *Na obrázku 3.5a je zeleně označený prvek P, který automat vybral z množiny {1, P, S}. Druhý prvek, vybraný z množiny zbývajících prvků {1, S}, je červeně označené S.*
2. *Prvně vybraný prvek uvolnil svou původní buňku a nahradil v poli prvek S. Pro ten automat vybral další pozici z množiny {1}, která je označená červenou barvou na obrázku 3.5b.*
3. *Na obrázku 3.5c je na pozici prvku 1 umístěný prvek S a jelikož byla vyčerpána množina prvků, nová pozice pro 1 je žlutě vyznačená buňka uvolněná prvně zvoleným P.*
4. *Obrázek 3.5d ukazuje stav po provedení jednoho kroku permutace.*

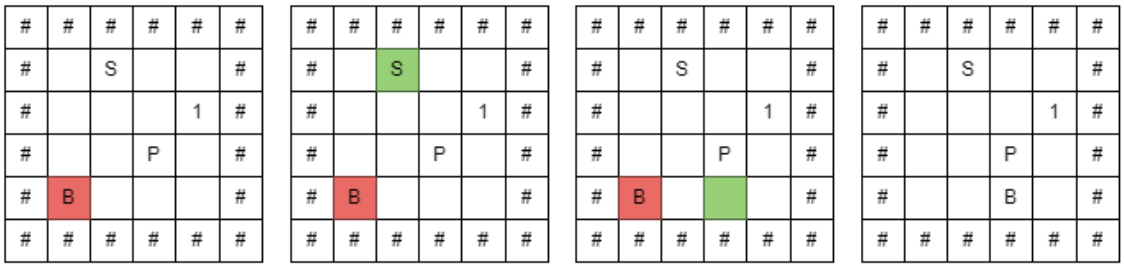


Obrázek 3.5: Permutující efekt

3.2.4 Nezávislý efekt

Pojmenovaný je podle skutečnosti, že automat v jednom kroku provede přesun symbolu na náhodně zvolené souřadnice nezávisle na aktuální pozici symbolu. Tento pohyb je oproštěn od závazku návaznosti, který mají všechny ostatní efekty, řídí se pouze pravidlem náhodného výběru. V jednom kroku automat náhodně zvolí symbol a buňku, do které by se měl přesunout. Pokud se v nové buňce nachází jiný symbol, zopakuje se výběr buňky.

1. *Automat zvolil prvek, který se v tomto kroku přesune. Je jím červeně označené B na obrázku 3.6a.*
2. *Dále byla zvolena buňka, do které se má prvek umístit. Jde o zeleně vyznačenou buňku na obrázku 3.6b, která je ale obsazená prvkem S.*
3. *Kvůli nedostupnosti buňky byl zopakován výběr, který označil zelenou buňku na obrázku 3.6c.*
4. *Zvolená buňka je prázdná, takže je do ní umístěn prvek B vybraný v bodě 1 a výsledný stav je zobrazen na obrázku 3.6d.*



Obrázek 3.6: Nezávislý efekt

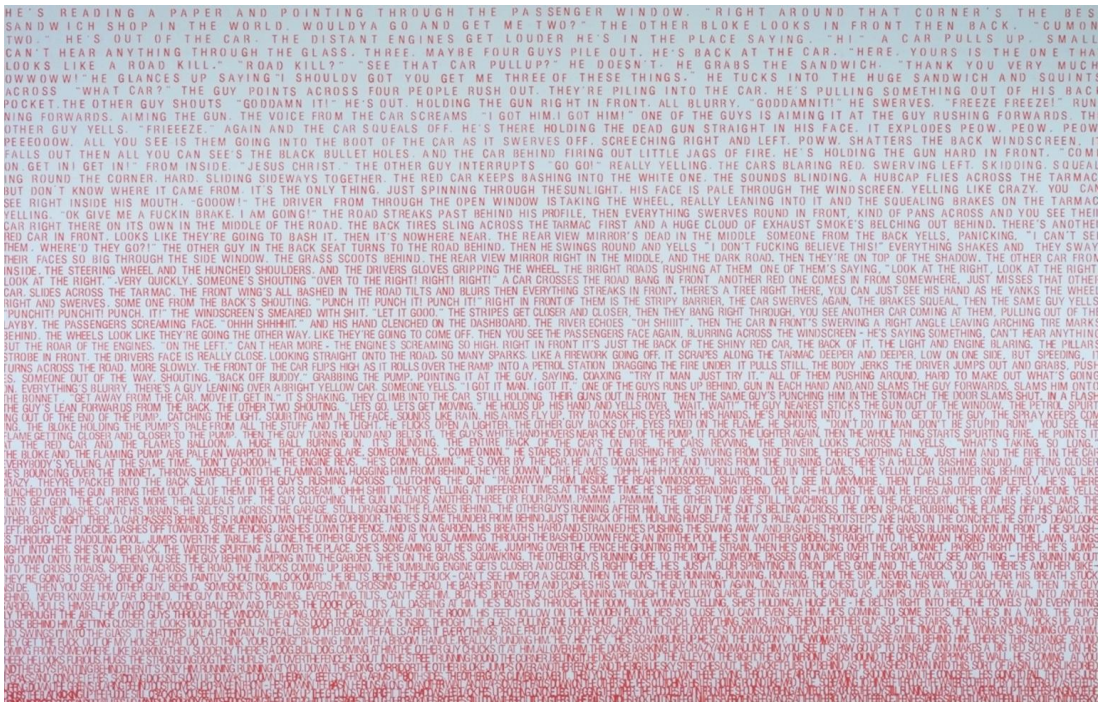
Kapitola 4

Vizuální umění

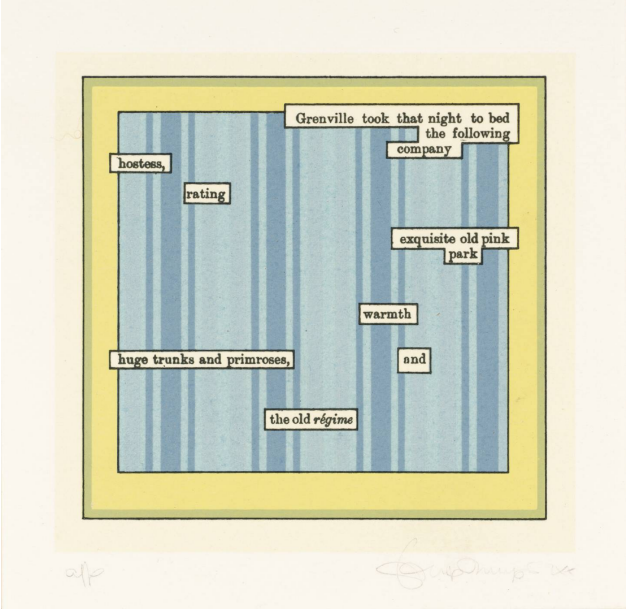
Umění zahrnuje originální díla vytvořená cílenou činností, která mají určitý vliv na diváka. Většina děl je schopná vyvolat v divákovi emoce, ať už je jejich hlavní funkce estetická, nebo například sdělovací. Tato schopnost je silným nástrojem, který umožňuje autorovi ovlivňovat velké množství lidí. Díky tomu je umělecká tvorba ideálním prostředkem pro předání myšlenky a její atraktivnost pro autory je více podtržena skutečností, že nabízí nesčetně možností pro reprezentaci a úpravu myšlenky. Výsledné dílo je obrazem původního sdělení v abstraktní formě, která mu dodává hloubku a dotváří vjem, který z díla čerpá divák. Myšlenku, kterou si z něj odnese, dokreslují zkušenosti pozorovatele, a může se tak velmi lišit od té původní. Tento proces komunikace a variabilita, kterou umožňuje, je to, co dělá umění zajímavým. Stále jsou představovány nové prostředky pro úpravu děl a jedním z nejpoužívanějších je počítač. Digitální tvorba, ať už kreslení pomocí aplikace, renderování definované scény, nebo další, je stále se rozšiřující oblast umění, jejíž nejnovější možnosti přidala umělá inteligence a do níž patří také využití automatů.

Tato kapitola je zaměřená na využití textu ve vizuálním umění. Písmena kromě graficky zajímavého tvaru přidávají do umění konkrétní význam, nesou informaci o hlásce, nebo slovech. Tento zcela přesný výklad textu je ale v dílech často pojat takovým způsobem, aby přiměl pozorovatele se nad ním hlouběji zamyslet.

Mezi umělce pracující s textem v obraze patří Fiona Banner, jejíž tvorba má poukázat na limity verbální komunikace. Jedno z jejích děl, znázorňující přepsaný scénář k akční scéně z filmu Break Point, je na obrázku 4.1. Další autor je například Tom Phillips, který jako základ svých děl používal stránky tištěných knih, jejichž slovům dal pomocí barev nový význam. Jeho tvorbu lze vidět na obrázku 4.2. [3]



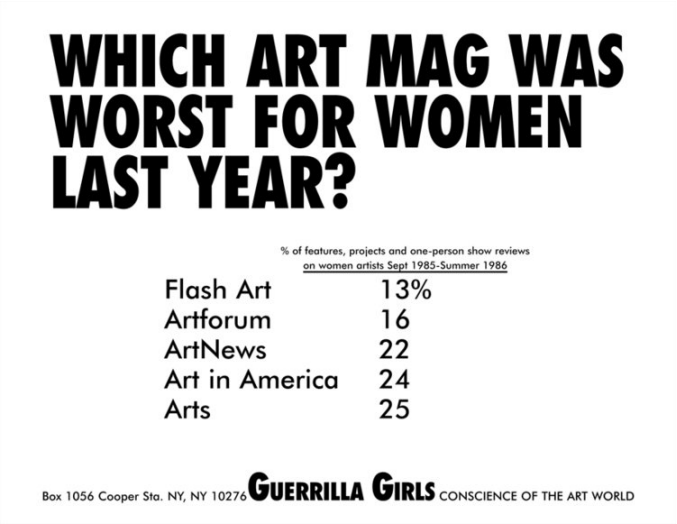
Obrázek 4.1: Fiona Banner: *Break Point* (1998) [5]



Obrázek 4.2: Tom Phillips: *[no title]* (1970) [23]

Nejvíce jsou texty v umění zastoupené v dílech aktivistů, tedy autorů, kteří se snaží svou tvorbou přimět společnost ke změně. Texty objevující se v jejich dílech jsou zaměřené na témata, o kterých se dost nemluví, společnost je nevnímá nebo na ně má zastaralý názor. Taková díla odráží aktuální náladu společnosti a představují zajímavý obrázek do historie.

Jedním z hnutí, které bylo formováno v 80. letech minulého století a které se zaobírá dodnes živým tématem, jsou Guerilla Girls. Jedná se o skupinu anonymních autorek, které nosí gorilí masky nebo tváře známých umělců, aby zdůraznily význam, který mají myšlenky jejich děl pro celou společnost nehledě na jakékoliv rozdělení. Jejich práce mají za cíl upozornit na nerovnoprávnost pohlaví a ras a často k tomu využívají vizuální umění v podobě plakátů, na nichž hraje stěžejní roli text, doplněný o zajímavé grafické zpracování. Příklady jsou na obrázcích 4.3 a 4.4. [4]



Obrázek 4.3: Guerilla Girls: *Which art mag was worst for women last year?* (1986) [11]

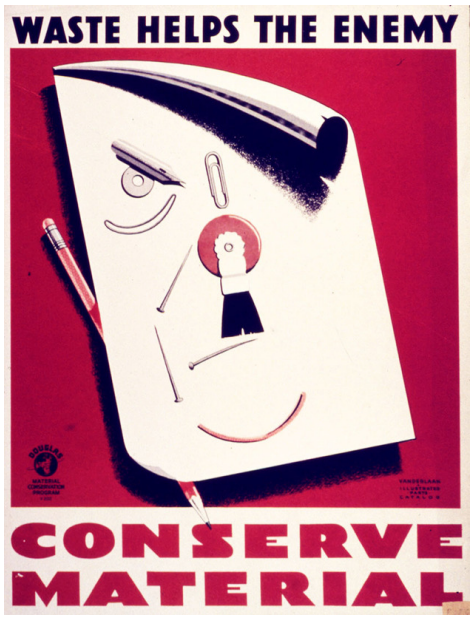
Obrázek 4.4: Guerilla Girls: *It's even worse in Europe* (1989) [12]

Autorka, která využívá plakáty podobným způsobem, je Barbara Kruger. Její tvorba je typická černým textem na červeném pozadí na podkladu tvořeném černobílou fotografií. Slogany, které se v jejích dílech vyskytují, mají pozorovatele vtáhnout do témat zabývajících se mocí a nerovností. Příklad je na obrázku 4.5. [2]

Obrázek 4.5: Barbara Kruger: *Your body is a battleground* (1989) [17]

Další typ plakátů, který má za cíl ovlivnit velké množství lidí, je propagandistický. Jedná se o plakáty, které vytváří formální instituce za účelem kontroly společnosti. Typickým příkladem jsou plakáty na obrázku 4.6, které americká vláda používala jako nástroj pro řízení společnosti za druhé světové války.

(a)



(b)

Obrázek 4.6: Propagandistické plakáty [1]

Kapitola 5

Rozpoznávání textu v obraze

Tato kapitola popisuje proces rozpoznávání znaků v obraze upravený pro graficky netradiční písmena vyskytující se ve vizuálním umění. Základní kroky a postupy jsou dobře popsané v několika publikacích a zaručují skvělé výsledky při převodu tištěného textu do digitální editovatelné podoby. V této práci se ale nedají uplatnit kritéria pro vstupní obraz, jako je 300 dpi, strukturování textu, ostré hrany znaků nebo vysoký kontrast s pozadím, které přispívají k úspěšnosti výsledků [19]. Bude se tedy očekávat horší přesnost v identifikaci písmen a pro zaručení dobrého výsledku je třeba, aby písmena ve vstupním obraze nebyla příliš vychýlená od jejich tradiční podoby a aby co nejlépe kontrastovala s pozadím. Takto omezený vstup lze pak zpracovat a dále předat rozpoznávacímu algoritmu, jehož výsledek se použije pro inicializaci automatu. V této kapitole je popsán proces předzpracování pomocí knihovny opencv a představen rozpoznávací algoritmus tesseractu.

5.1 Postup

Podle Chaudhuri et al. [6] se OCR (Optical Character Recognition) systémy skládají z osmi částí, které jsou vyznačeny na schéma na obrázku 5.1. Jsou to:

Optické skenování - Tato část zahrnuje převedení dokumentu do digitální podoby například pomocí skeneru.

Segmentace oblastí - Část, ve které dochází k detekci oblastí obsahujících text a jejich rozdělení na jednotlivé symboly.

Předzpracování - Cílem předzpracování je převést oblasti identifikované v předešlém kroku do podoby, která usnadní samotné rozpoznávání. Tato fáze nejčastěji zahrnuje tři kroky: redukce šumu, normalizace symbolů a komprese množství dat, které reprezentují daný symbol.

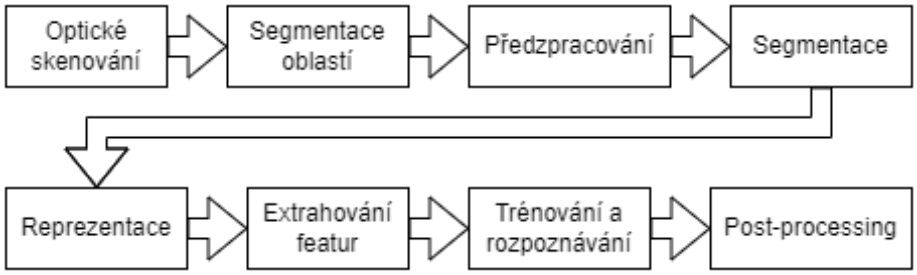
Segmentace - V této fázi dochází k dělení symbolů na jejich dílčí komponenty.

Reprezentace - Tato část převádí symbol z vizuální podoby na sadu dat, které jej popisují.

Extrahování featur - V tomto kroku jsou z předchozí reprezentace symbolu vyňata data, která jednoznačně určují třídu symbolu a zároveň povolují určitou míru variability v rámci třídy.

Trénování a rozpoznávání - Při trénování jsou vytvořeny reprezentace jednotlivých tříd, které při rozpoznávání představují šablony, jejichž porovnání se symbolem generuje skóre určující příslušnost symbolu ke třídě.

Post-processing - Výsledky jednotlivých symbolů jsou v tomto kroku zasazeny do širšího kontextu, který s využitím syntaxe nebo pravděpodobnosti redukuje počet chyb ve výsledném textu.



Obrázek 5.1: Schéma OCR systému [6]

Proces rozpoznávání začíná převodem dokumentu do digitální podoby pomocí optického skeneru. Následně jsou detekovány oblasti, na kterých se vyskytuje text a jednotlivé symboly. Každý symbol je převeden do podoby, ve které ho lze porovnávat s třídami získanými ve fázi učení, a výsledek takového porovnávání je pak zasazen do širšího kontextu, který tvoří slova původního textu. [6]

5.1.1 Předzpracování obrazu

Než se nahraný obrázek pošle do rozpoznávacího algoritmu je vhodné, aby byl upraven tak, aby na něm byla co nejpřesněji vyznačena písmena. Charakteristiky fontů jako patky, tloušťka nebo sklon totiž negativně ovlivňují výsledek, a to zvláště v případě, kdy je dopředu neznáme a algoritmus tak není možné natrénovat. Z toho důvodu se využívají techniky zpracování obrazu, které písmena zvýrazní a zjednoduší, a přiblíží tak výstup dokonalosti.

Nafiz et al. [21] dělí techniky podle jejich účelu na tři kategorie. První z nich je redukce šumu, jejímž cílem je zbavit text nedokonalostí vzniklých při tvorbě textu nebo jeho převodu do elektronické podoby. Tyto nedokonalosti zahrnují přerušované čáry, vyplněné mezery, zaoblení rohů apod. a dají se z velké části odstranit pomocí přístupů, které lze dělit do tří kategorií: filtrování, morfologické operace a modelování šumu.

Filtrování má za úkol redukovat počet drobných nepřesností aplikováním konvolučního filtru, jehož principem je přiřazení hodnoty pixelu na základě okolních pixelů. [21] V opencv má tento efekt metoda dilate, která symboly v obraze vyhladí. [15]

Morfologické operace jsou založeny na logických operacích a jejich účelem je rekonstrukce linií, vyhlazení obrysů a například zúžení fontu. Ve výsledné aplikaci tyto operace ale nejsou zastoupeny kvůli neúměrnosti časové náročnosti s dopadem na přesnost výsledku. [21]

Modelování šumu je technika založená na znalosti šumu, který se v dokumentu vyskytuje, a dá se tedy využít jen v některých případech. Jedním z typů šumu, pro který existuje model, je optické zkreslení, to se ale na vstupu, který očekává výsledná aplikace nevyskytuje, takže tato technika nebude v aplikaci zastoupena. [21]

Druhá kategorie technik je normalizace symbolů a má za cíl sjednocení stylu jednotlivých znaků v delším textu. V praktické části této práce se nepočítá se souvislým textem, takže

zde jen zmíním, že normalizace řeší problémy náklonu písmen a řádků, dále sjednocení velikosti písmen a vyhlazení obrysů za účelem zjednodušení reprezentace znaků pro další zpracování. [21]

Cílem třetí kategorie technik je snížení objemu dat reprezentujících znaky a nejčastěji se za tímto účelem využívá prahování a zúžení.

Prahování znamená převod vstupního obrázku do binární podoby, kdy pixely nabývají jen dvou hodnot. Rozhodování probíhá na základě prahové hodnoty, se kterou je porovnána původní hodnota pixelu a může být globální nebo lokální. Globální práh se určí jednou pro celý obrázek a následně se využije při stanovení hodnot všech pixelů. Nejjednodušší metoda stanovení prahu je jeho přiřazení nezávisle na vstupu, ale existují také algoritmy, které práh vypočítají na základě hodnot vyskytujících se na obrázku. [21] Příkladem může být algoritmus založený na histogramu intenzity, jehož autorem je Nobuyuki Otsu. [22] V opencv jej lze najít jako parametr THRESH_OTSU metody treshold. [7]

Lokální práh se určuje pro každý pixel jinak na základě jeho okolí a je tak schopný reagovat na různorodé světelné podmínky, které hrají roli při digitalizaci vstupu. [21]

Zúžení usiluje o redukci bodů, které tvoří symboly. Algoritmy pro zužování se dělí na pixelové a nepixelové a rozdíl mezi nimi je, že pixelové redukují šířku symbolu pixel po pixelu, až zbude kostra o tloušťce jeden pixel. Tento způsob je velmi citlivý na šum a může negativně ovlivnit výsledný tvar. Nepixelové metody používají globální informace o písmenu, například shluky nebo jedinečné body, k extrakci jeho centrální linie. Oba typy lze implementovat sekvenčně nebo pro větší efektivitu paralelně. [21] Příkladem metody zúžení je erode knihovny opencv. [15]

5.1.2 Princip rozpoznávání

Zatímco fáze předzpracování řeší obrazová data symbolu jako celku, ty následující jeden symbol dělí na části a extrahují z nich jen některá data tak, aby s nimi dobře pracoval rozpoznávací algoritmus. Jelikož jsou tyto fáze provázané, tato sekce nepředstaví různé způsoby, kterými by se daly implementovat, ale zaměří se na principy využité v knihovně tesseract, která je základem rozpoznávání v praktické části této práce.

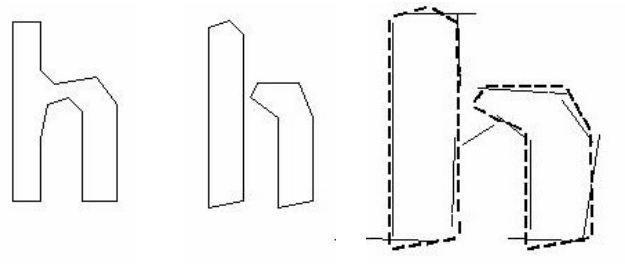
Tesseract je open-source projekt, jehož základem je diplomová práce Ray Smith, z roku 1987 [27] a jehož počáteční vývoj v letech 1984 až 1994 připisuje Smith v článku An Overview of the Tesseract OCR Engine [26] firmě HP a jejímu záměru nahradit nepřesné komerčně dostupné programy. Výsledný software sice zaznamenal úspěch v UNLV Annual Test of OCR Accuracy roku 1995 [24], nestal se z něj ale produkt a roku 2005 byly jeho zdrojové soubory zpřístupněné pro veřejnost. Dnes se nachází na GitHubu[1].

Ve výsledné aplikaci je knihovna tesseract zastoupena pouze metodou image_to_data, jejímž voláním se získá struktura, která obsahuje identifikované písmeno a jistotu, se kterou bylo určeno. Díky parametru --psm 10 [29] tesseract přeskočí kroky hledající spodní linii řádku, slovo a jeho dělení na jednotlivé znaky. Pouze z písmene na obrázku extrahuje featury a porovná je s šablonou.

Šablony reprezentují jednotlivé třídy a jsou tvořeny částmi polygonální aproximace symbolů [26]. Tyto části jsou vyznačené na obrázku 5.2c jako tenké rovné linie a definovány jsou čtveřicí (x, y pozice, úhel, délka). Samotné symboly jsou reprezentovány malými úseky polygonální aproximace s jednotnou délkou, na obrázku 5.2c vyznačenými krátkými silnými čarami, a definovány trojicí (x, y pozice, úhel). Rozdílné povahy featur neznámých symbolů a šablon redukují dopad nepřesností, které mohou vznikat při tvorbě polygonální aproxi-

[1] https://github.com/tesseract-ocr/tesseract

mace. Příkladem je střední linie šablony na obrázku 5.2c, na kterou nesedí ani jedna featura extrahovaná z nepřesného písmene h. Ostatní úseky ale celkem dobře kopírují linie šablony, a ze součtu jejich skóre porovnání by takové písmeno h bylo správně rozpoznáno.



(a) (b) (c)

Obrázek 5.2: (a) neponičené h (b) poškozené h (c) featury z (a) jako tenké linie v porovnání s tučnými úseky featur z (b) [26]

Rozpoznávání probíhá ve dvou krocích. V tom prvním je pro každou featuru podle vyhledávací tabulky sestaven vektor šablon, ke kterým by mohla náležet, a součet vektorů vybraných featur vytváří skóre jednotlivých tříd. Ty s nejvyšším skóre pak postupují do dalšího kroku, ve kterém je hodnota zpřesněna pomocí míry příslušnosti featur k liniím a jejich vzdálenosti. Výsledkem je seznam nejbližších tříd, mezi kterými lze dále vybírat podle kontextu, ve kterém se symbol nachází. [26]

Kapitola 6

Implementace

Výsledným produktem této práce je aplikace s grafickým uživatelským rozhraním, kterou lze využít pro zobrazení efektu pracujícího s textem v obraze. Za vykonáním efektu stojí automat definovaný v kapitole 3, jehož vstupem je pole symbolů nalezených v obraze pomocí algoritmu popsaného v kapitole 5.

6.1 Technologie

Aplikace je napsaná pro interpret Python 3.10[1] s využitím knihoven napsaných v jazyce c++ pro tvorbu grafického uživatelského rozhraní a rozpoznávání textu v obraze. Potřebné balíky jsou:

- PySide6[2] - grafické uživatelské rozhraní založené na frameworku Qt
- PyTesseract[3] - rozpoznávání textu pomocí Tesseract-OCR
- OpenCV[4] - zpracování obrazu
- numpy[5] - práce s daty

Kromě stažení těchto balíku je také potřeba nainstalovat Tesseract-OCR.exe, například z GitHubu[6], a do proměnné PATH přidat cestu k tomuto spustitelnému souboru.

6.2 Spuštění

Aplikace se spouští příkazem s volitelným parametrem --verbose, který povoluje výstup do konzole:

```
python app.py [-v]
```

Po otevření okna aplikace se očekává, že do něj uživatel přetáhne obrázek, na kterém je text. Po jeho načtení a zobrazení je automat připraven a čeká, než uživatel stiskem tlačítka zvolí efekt, který by měl vykonat.

[1] https://www.python.org/downloads/release/python-3103/
[2] https://pypi.org/project/PySide6/
[3] https://github.com/tesseract-ocr/tesseract
[4] https://pypi.org/project/opencv-python/
[5] https://numpy.org/
[6] https://github.com/UB-Mannheim/tesseract/wiki

6.3 Uživatelské rozhraní

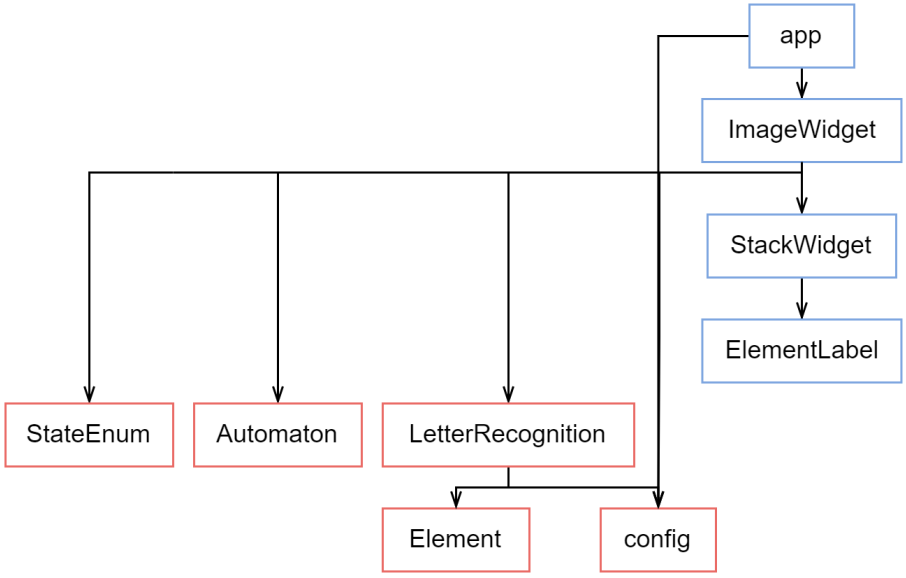
Uživatelské rozhraní lze vidět na obrázku 6.1. Rozložení okna odpovídá QMainWindow[7], jehož hlavním prvkem je CentralWidget, obsahující obraz a elementy. Tlačítka pro ovládání automatu jsou součástí ToolBaru, což je další z prvků QMainWindow.

Obrázek 6.1: Vzhled aplikace

6.4 Struktura programu

Vytvořený program nabízí nástroje pro nalezení písmen latinské abecedy a arabských číslic v obraze, řízení chodu automatu a komunikaci s uživatelem pomocí grafického rozhraní. Hierarchie jeho částí je na obrázku 6.2.

[7] https://doc.qt.io/qt-6/qmainwindow.html



Obrázek 6.2: Struktura programu - modře vyznačené třídy tvoří grafické uživatelské rozhraní

Nejnižší vrstvu tvoří Element, definovaný v souboru element.py. Instance Elementu reprezentuje nalezený znak a uchovává data potřebná pro třídy ve vyšších vrstvách. První z nich, LetterRecognition, je definovaná v souboru letterRecognition.py a představuje příklad implementace OCR schéma pomocí knihoven opencv a pytesseract. Jednotlivé kroky a jejich části jsou v ní rozděleny do metod tak, aby bylo umožněno jejich editací a zřetězením volání jednoduše vyzkoušet různé postupy za účelem získání co nejpřesnějšího výsledku ze vstupního obrázku. Výstupem rozpoznávání je pole Elementů, které zároveň tvoří vstup další třídy, Automatonu, definovaného v souboru automaton.py. Tato třída obsahuje data reprezentující stav automatu a metody potřebné pro jeho činnost.

Hlavní grafický prvek, ImageWidget, je definovaný v souboru imageWidget.py. Tento prvek propojuje třídy nižší vrstvy do fungujícího celku a podle uživatelského vstupu volá jejich metody. Pro přehledné označení vybraných funkcí používá výčtový typ StateEnum, definovaný v souboru stateEnum.py. Kromě řízení chodu programu ImageWidget také zprostředkovává jeho výstup v podobě ElementLabelů zobrazených pomocí StackWidgetu. ElementLabel je třída v souboru elementLabel.py, která představuje grafickou reprezentaci Elementu. StackWidget, definovaný v souboru stackWidget.py, se používá pro zobrazení ElementLabelů na pozadí obrázku.

Nejvyšší vrstvu tvoří soubor app.py, který obsahuje inicializaci grafického rozhraní a hlavní aplikační cyklus. Soubor config.py je součástí nejnižší vrstvy, ale je uveden jako poslední, jelikož pouze obsahuje proměnnou uchovávající informaci o nutnosti textového výstupu.

6.5 Třídy

V této části je popsána struktura programu, jeho rozdělení do tříd a souborů. Pro přehlednost kódu odpovídají názvy tříd názvům souborů, ve kterých jsou definovány, a je zachována nezávislost tříd Automaton a LetterRecognition na ostatních třídách s výjimkou Elementu. U tříd uvádím jejich využití, data, která uchovávají a popis metod, které definují.

6.5.1 Element

Element uchovává data o nalezeném symbolu, potřebná pro vykonání efektu jak v automatu, tak ve třídě zobrazující jeho aktuální stav.

Data

`letter` - písmeno nebo číslice

`isNumerical` - bool značící číselné elementy

`confidence` - míra jistoty rozpoznávacího algoritmu

`rectangle` - čtverec opsaný symbolu v obrázku

`contour` - contour, která přesně ohraničuje symbol na obrázku

`color` - barva symbolu ve formátu RGB

`backgroundColor` - barva vyplňující místo po prvním přesunu symbolu

`index` - index odpovídajícího obrysu v poli contours

`graphic` - grafická reprezentace symbolu

`cell` - buňka, ve které se element právě nachází, reprezentovaná indexy do vstupního pole

Konstruktor

`__init__(letter, confidence, rectangle, contour, color, backgroundColor, index = None, graphic = None, cell = None)`

Konstruktor pro Element má jedinou funkci – přiřazuje parametry do odpovídajících atributů Elementu.

Metody

Element slouží výhradně pro shlukování dat náležících jedné instanci, takže nedefinuje žádné aktivní metody.

`__str__()`

Reprezentace Elementu jako literálového řetězce je zde zjednodušena na atribut letter.

6.5.2 Automaton

Třída reprezentující automat je definována podle návrhu v kapitole 3, její hlavní funkce je řídit efekty a pohyb prvků v poli buněk. Tyto operace jsou prováděny na vlastním vláknu automatu, aby nedocházelo k zastavení hlavního vlákna, které v mnoha případech obsahuje i aplikační cyklus grafického uživatelského rozhraní. Aplikace je tak schopná přijímat uživatelský vstup a měnit chování automatu za jeho běhu.

Data

Instance automatu si pamatuje jednotlivé prvky šestice, která je teoretickým základem jeho definice, a některé další atributy, které určují jeho aktuální stav.

`elements` - pole prvků, se kterými automat pracuje

`usedCells` - pole obsazených buněk

`chosenEffect` - funkce efektu, kterým se má automat řídit

`mutexChosenEffect` - threading.Lock zaručující stabilitu přístupu k právě zvolenému efektu

`onElementMoved` - funkce, která se vykoná při posunu prvku

`onStopped` - funkce, která se vykoná po ukončení chodu automatu

`effectThread` - threading.Thread, na kterém probíhá výběr prvku a buňky podle zvoleného efektu

`cellWidth` - šířka buňky

`cellHeight` - výška buňky

`cellXMax` - největší index, který buňka může mít na ose x

`cellYMax` - největší index, který buňka může mít na ose y

`rng` - np.random.default_rng, generátor náhodných čísel

Konstruktor

```
__init__(elements, imgWidth, imgHeight, onElementMoved = None, onStopped = None)
```

`elements` - pole prvků automatu

`imgWidth` - šířka vstupního obrazu

`imgHeight` - výška vstupního obrazu

`onElementMoved` - funkce, která se zavolá po přesunutí prvku

`onStopped` - funkce, která se volá při dokončení běhu automatu

Konstruktor nejdříve přiřadí argumenty do odpovídajících parametrů a inicializuje effectThread, jehož cílem je metoda run. Následuje výpočet aritmetického průměru velikostí symbolů, který představuje velikost buněk, a použije se pro zjištění maximálního indexu v poli buněk. Dalším krokem je cyklus, v němž se každému prvku přiřadí buňka podle jeho výchozí pozice v obrázku. Použité buňky jsou uloženy do pole obsazených buněk.

Metody

Nejdůležitější metody automatu jsou ty, které posouvají prvky a vykonávají tak kroky efektu. Kromě těchto metod definuje automat některé podpůrné metody a jeden lambda výraz pro zjednodušení zápisu.

`generateRandom(min = 0, max = 1)`
Vrací náhodné celé číslo z intervalu <min, max) vygenerované pomocí rng.random. Tato metoda je základem výběru pro metody chooseElement, chooseAndRemove a chooseCell.

`chooseElement()`
Vrací jeden prvek z pole elements.

`chooseAndRemove(items)`
Odstraní jednu položku z pole items a vrátí její hodnotu.

`chooseCell()`
Vrací tuple (x, y), kde x $\in$ <0, cellXMax) a y $\in$ <0, cellYMax).

`RandomEffectStep()`
Provede jeden krok náhodného efektu, který je popsán v kapitole 3.2.1. Pro přehlednější úpravu kódu využívá lambda výraz getPossibleCells, který inicializuje pole sousedních buněk podle Moorova okolí.

`RandomLimEffectStep()`
Provede jeden krok náhodného efektu s omezením, který je popsán v kapitole 3.2.2. Jako jeho výchozí verze využívá lambda výraz getPossibleCells, jehož výsledek edituje podle toho, zda je přesouváno písmeno, nebo číslice.

`IndependentEffectStep()`
Provede jeden krok nezávislého efektu, podle definice v kapitole 3.2.4.

`PermutateEffect()`
Provede efekt permutace, popsaný v kapitole 3.2.3.

`run()`
Obsahuje hlavní smyčku, která je omezená počtem iterací MAX_EFFECT_STEPS a mezi kroky čeká TIME_TO_SLEEP sekund. V každém cyklu se provede krok efektu uložený v atributu effect.

`runThread()`
Spustí smyčku automatu uloženou ve funkci run na vláknu effectThread.

`moveElement(element, newCell)`
Přiřadí elementu novou buňku a zavolá onElementMoved.

`print()`

Vypíše aktuální stav automatu jako seznam položek tvaru A (x, y), kde A je symbol prvku a x, y jsou souřadnice buňky, ve které se nachází.

6.5.3 LetterRecognition

Třída LetterRecognition je nástroj, který lze využít pro hledání symbolů v obraze, od načtení obrázku přes jeho editaci až k výslednému poli identifikovaných Elementů. Tato funkcionalita stojí na metodách knihoven opencv a pytesseract a jejich využití se dělí v okamžiku, kdy byly z obrazu extrahovány úseky, které obsahují možné symboly.

Data

`imgPath` - cesta ke vstupnímu obrazu

`img` - vstupní obrázek

`imgGray` - obrázek v odstínech šedé

`imgBW` - obrázek v černobílé variantě

`imgText` - obrázek s vyznačenými nalezenými symboly

`contours` - pole obrysů symbolů

`hierarchy` - pole hierarchie symbolů

`elements` - pole nalezených prvků

Metody

Proces rozpoznávání třídou LetterRecognition odpovídá schéma OCR představenému v kapitole 5. Dělí jej na jednotlivé kroky tak, aby se snadno daly nahradit alternativními postupy pro zpřesnění výsledků pouhým řetězením volání. Návratem self z jednotlivých metod je umožněn zápis na jeden řádek.

`readImage(imgPath)`

`imgPath` - cesta ke vstupnímu obrázku

Do pole img načte vtupní obrázek meodou opencv.imread a provede inicializaci argumentů.

`toGray()`
Do imgGray uloží vstupní obraz img převedený do odstínů šedé pomocí metody opencv.cvtColor.

`toBWOtsu()`
Pomocí Otsuova prahování a metody opencv.treshold převede šedý obrázek imgGray na černobílý a uloží jej do pole imgBW.

`extractContours()`

Na černobílém obrázku imgBW najde obrysy symbolů a jejich hierarchii pomocí metody opencv.findContours a uloží je do polí contours a hierarchy.

`getLettersBoundingRect()`

Provádí samotné rozpoznávání a inicializaci Elementů. Pro každý obrys uložený v poli contours najde nejmenší opsaný obdélník a jeho výřez z imgBW pošle do metody pytesseract.image_to_data. Podle navrácených dat dále vyřadí obrysy s nedostatečnou jistotou rozpoznání a takové, které neobkreslují písmena nebo číslice. Přijaté obrysy pokračují k extrakci barev symbolu a jeho pozadí a k samotné inicializaci Elementu. Velikosti symbolů jsou použity při výpočtu průměru, který slouží jako druhý filtr, zaručující konzistentní vzhled efektů.

`getImage(imgPath = None)`

`imgPath` - cesta ke vstupnímu obrázku

Metoda spojuje předešlé metody do řetězce, jehož vstupem je cesta k obrázku a jehož výstupem je cesta k obrázku s vyznačenými prvky, které byly identifikovány.

`getElements(imgPath = None)`

`imgPath` - cesta ke vstupnímu obrázku

Metoda, stejně jako ta předchozí, vytváří řetězec, jehož vstupem je cesta k obrázku. Výstupem je ale pole nalezených elementů a velikost obrázku.

6.5.4 ElementLabel

ElementLabel dědí z QLabel a používá se pro reprezentaci elementu v grafickém uživatelském rozhraní. ElementLabel uchovává tvar symbolu a řeší jeho vykreslení v okně aplikace.

Data

`graphic` - QPainterPath, která obkresluje symbol

`color` - QColor, barva symbolu

Konstruktor

`__init__(element, parent = None)`

`element` - korespondující Element

`parent` - rodičovský prvek

V konstruktoru jsou přiřazeny hodnoty polí graphic a color podle odpovídajícího Elementu. Graphic je posunut na pozici [0,0] a celý ElementLabel je umístěn na původní pozici symbolu. Velikost je nastavena podle minimálního obdélníku obkreslujícího symbol, uloženého v poli Element.rectangle.

Metody

`paintEvent(paintEvent)`
Tato metoda vykreslí graphic vybarvený podle color a volá se při každé aktualizaci ElementLabelu.

6.5.5 StackWidget

StackWidget dědí z QWidgetu a využívá se pro zobrazení vrstev obrazu. Spodní vrstva obsahuje samotný vstupní obraz, nad kterým se vyskytují obdélníky zakrývající původní symboly a v nejvyšší vrstvě grafické reprezentace symbolů.

Data

`imgLabel` - QLabel zobrazující obraz

`elements` - pole Elementů, které se v obraze vyskytují

Konstruktor

`__init__(imagePath, elements, parent = None)`

`imagePath` - cesta k obrázku

`elements` - pole Elementů

`parent` - rodičovský prvek

Konstruktor vytváří imgLabel zobrazující obrázek pomocí QPixmap, obdélníkové QLabely kryjící původní symboly a ElementLabely představující symboly ve formě vhodné k jednoduché manipulaci v grafickém uživatelském rozhraní.

Metody

`paint()`
Vykreslí graphicy všech Elementů.

`dropEvent(event)`
Definován proto, aby bylo možné změnit vstupní obraz a znovu inicializovat automat.

6.5.6 ImageWidget

ImageWidget dědí z QWidget a řeší propojení všech komponent a jejich inicializaci. Ta začíná přetažením obrázku, na kterém jsou nalezeny symboly třídou LetterRecognition a jako pole Elementů předány Automatonu a StackWidgetu. Po zvolení efektu uživatelem zastává ImageWidget funkci prostředníka mezi automatem a grafickou reprezentací jeho stavu.

Data

`imagePath` - cesta k obrázku

`stackWidget` - StackWidget

`progressWidget` - grafický prvek zobrazující QProgressBar při průběhu rozpoznávání textu

`letterRecognition` - LetterRecognition

`automaton` - Automaton

`letterRecognitionThread` - threading.thread, na kterém probíhá rozpoznávání textu

`getLettersFinished` - QtCore.Signal, znamení, které dává letterRecognitionThread těsně před svým ukončením

`automationStoppedEvent` - QtCore.Signal, znamení, které dává automat po ukončení své činnosti

Konstruktor

`__init__(parent=None)`

`parent` - rodičovský prvek

Konstruktor propojuje signály s metodami, nastavuje rodičovský prvek a QStackedLayout.

Metody

`dragEnterEvent(event)`
Akce přesunutí obsahu do pole ImageWidgetu je přijata, pokud se jedná o obrázek.

`dropEvent(event)`
Metoda reaguje na akci puštění levého tlačítka myši v poli ImageWidgetu zobrazením progressWidgetu a spuštěním vlákna letterRecognitionThread.

`getLetters()`
Nejdříve inicializuje letterRecognition, jehož metodu getElements využije pro získání elementů z obrázku uloženého na ceste imagePath, které následně použije pro inicializaci automatu.

`getLettersFinishedEvent()`
Vytvoří a zobrazí stackWidget.

`elementsGraphicInit()`
Všem prvkům v poli elements vytvoří grafickou cestu QPainterPath podle obrysu v jejich poli contour.

`makeSubpath(pp, contour)`

`pp` - QPainterPath

`contour` - opencv contour

K QPainterPath pp přidá uzavřenou cestu podle bodů v contour.

`saveImage()`
Uloží aktuální vzhled obrázku.

`setEffect(id)`

`id` - int hodnota odpovídající StateEnum

Předá zvolený efekt automatu a spustí jej.

`onAutomatonStopped()`
Zvolí Stop QPushButton z nabídky efektů v QToolBaru.

`moveElement(element, newCell)`

`element` - Element

`newCell` - indexy [x,y] nové buňky

Přesune grafickou reprezentaci elementu do nové buňky pomocí indexů a velikosti buňky uložené v letterRecognition.

6.5.7 StateEnum

StateEnum se používá pro sjednocení efektů do jednoduché reprezentace pomocí celých čísel.

Hodnoty

0 `IDLE` - bez efektu

1 `RANDOM` - náhodný

2 `RANDOM_LIM` - náhodný s omezením

3 `INDEPENDENT` - nezávislý

4 `PERMUTATE` - permutace

Kapitola 7

Testování

Účelem testování je zhodnocení výsledků práce, ať už se jedná o samotný program, nebo jeho výstup. Na počítači, ze kterého pochází výsledky představené v této kapitole, byl nainstalován operační systém Windows 10 64bit a program Tesseract-OCR verze 5.0.1. Testování probíhalo pomocí příkazové řádky prostřednictvím interpretu pro Python verze 3.10 s těmito balíky:

- PySide6 verze 6.2.1
- pytesseract verze 0.3.8
- opencv-python verze 4.5.5.64
- numpy verze 1.21.4

Tato kapitola je rozdělena na dvě části, v první je zhodnocena přesnost rozpoznávacího algoritmu a ve druhé chod automatu.

7.1 Rozpoznávání

LetterRecognition je samostatná třída, která má za účel nalézt v obraze písmena a číslice. Výstupem je pole elementů, nebo i obrázek s vyznačenými nalezenými znaky. Výsledkem testování jsou zeleně vyznačené identifikované znaky na obrázku 7.1.

(a) It's even worse in Europe

(b) Bunkr

Obrázek 7.1: Znaky v obrazech

Z obrázku je patrné, že přesnost LetterRecognition je asi 79 %, jelikož z 48 jasně definovaných písmen tvořících nápisy IT'S EVEN WORSE IN EUROPE, GUERILLA GIRLS a BYLI JSME A BUDEM bylo správně identifikováno 38, další správně označené písmeno je W na obrázku 7.1a, ale ve výsledku jsou také dvě falešná pozitiva (a, n) na obrázku 7.1b.

Mezi příčiny nepřesností mohou spadat například parametr --psm 10, kterým je omezena funkce tesseractu na rozpoznávání jednoho písmena a který je nutný vzhledem k povaze vstupu. Bez kontextu zbytku slova ale dochází k poklesu úspěšnosti identifikace. Další faktor, který hraje roli při rozpoznávání, je úprava vstupního obrazu, jejímž výstupem je vyříznuté písmeno v černobílé podobě.

Samotné výsledky tesseractu jsou v některých případech velmi přesné, na druhou stranu občas dochází k falešné identifikaci znaku s vysokou mírou jistoty v obraze, na kterém se vyskytuje tvar pro člověka zřetelně nepodobný jakémukoliv znaku. Tyto chyby jsou do jisté míry redukovány při kontrole velikosti nalezených písmen, kterou jsou přijaty pouze znaky s odchylkou 0.1 od průměru. Pro další zpřesnění výsledku by bylo nutné více omezit vstup.

7.2 Automat

Jednotlivé efekty automatu byly demonstrovány na různých obrázcích. U každého jsou uvedeny základní údaje o počátečním stavu automatu a zobrazeny stavy po dvou kolech iterací.

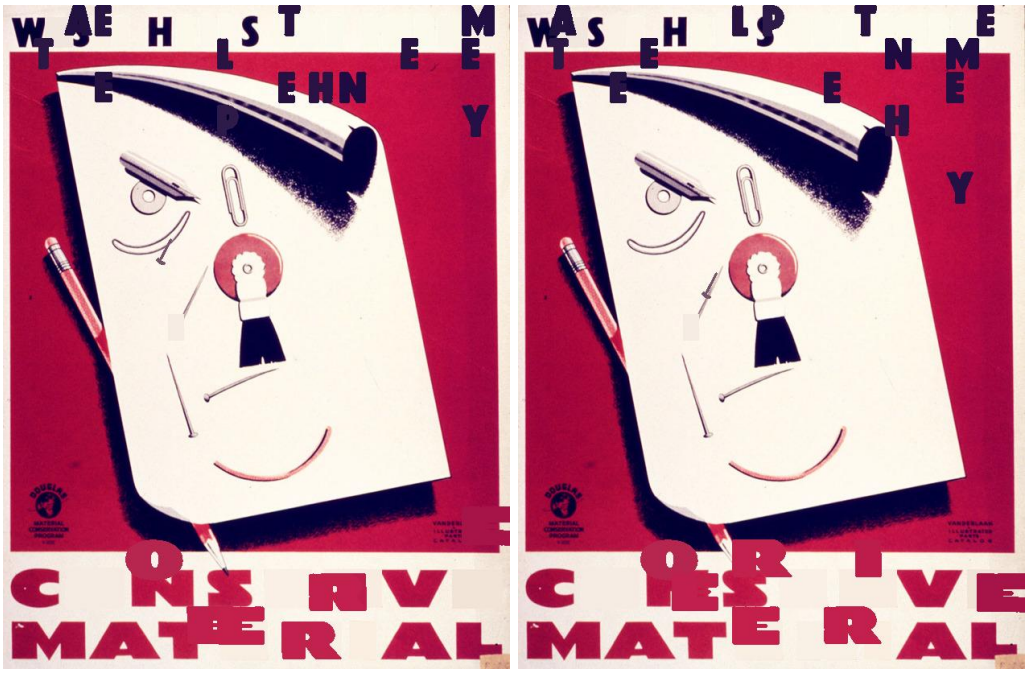
Náhodný

Obraz: Waste Helps The Enemy (4.6b)

Velikost: 630 x 825

Velikost buněk: 38 x 41.5

Počet buněk: 16 x 19



(a) po 50 iteracích (b) po 100 iteracích

Obrázek 7.2: Náhodný efekt

Náhodný efekt v jedné iteraci vybere znak a buňku z jeho okolí, do níž jej přesune. Na obrázku 7.2 je vidět posun písmen do nových buněk po 50 a 100 iteracích. Například černé písmeno A v levém horním rohu bylo nejdříve přesunuto z výchozí pozice do severo-východní a z té následně do západní.

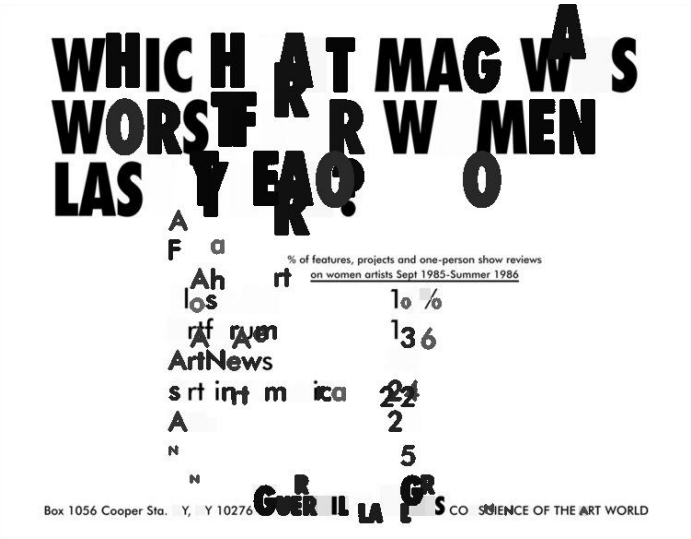
Náhodný s omezením

Obraz: Which art mag was worst for women last year? (4.3)

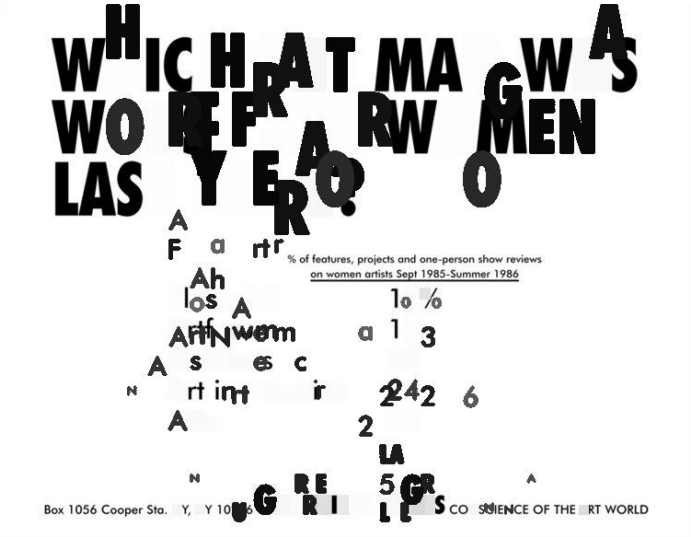
Velikost: 750 x 580

Velikost buněk: 22.8 x 31.9

Počet buněk: 32 x 18



(a) po 50 iteracích



(b) po 100 iteracích

Obrázek 7.3: Náhodný efekt s omezením

Omezení náhodného efektu má za důsledek postupné třídění písmen a číslic. Písmena, například H v levé spodní části obrázku 7.3, jsou přesouvána do horní poloviny obrázku, zatímco číslice, například 6 v pravé střední části, směřují ke spodní hranici obrázku. Posun některých písmen dolů – například G v pravé horní části a O pod ním – se dá vysvětlit nepřesností rozpoznávání, kvůli které jsou klasifikovány jako číslice.

Nezávislý

Obraz: Pvt. Joe Louis (4.6a)

Velikost: 572 x 825

Velikost buněk: 19.4 x 25.2

Počet buněk: 29 x 32

(a) po 50 iteracích (b) po 100 iteracích

Obrázek 7.4: Nezávislý efekt

Nezávislý efekt v jedné iteraci vybere znak a libovolnou buňku z pole, do níž jej přesune. Obrázek 7.4 ilustruje stav automatu po 50 a 100 takových posunech.

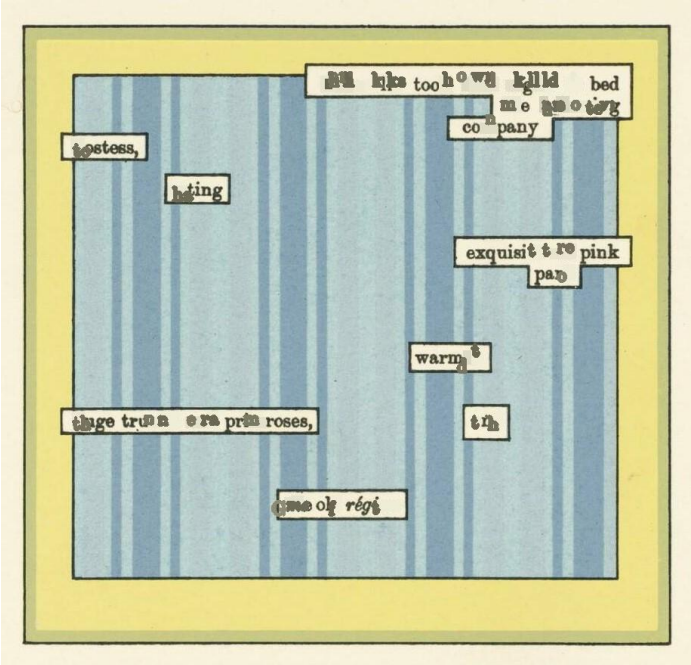
Permutace

Obraz: [no title] (4.2)

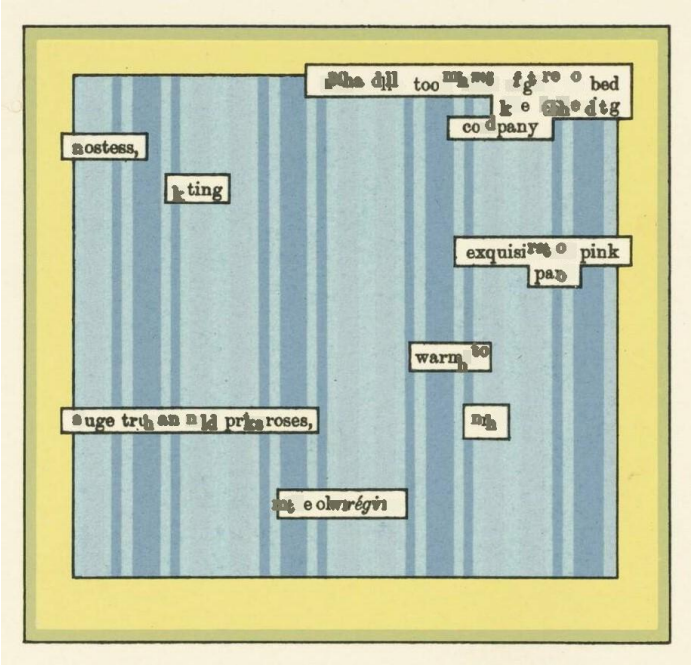
Velikost: 914 x 883

Velikost buněk: 18.9 x 18.9

Počet buněk: 48 x 46



(a) po prvním kole



(b) po druhém kole

Obrázek 7.5: Permutace

Permutace zaměňuje pozice nalezených znaků a vytváří tak nová slova. Na obrázku 7.5 lze vidět příklady takto vzniklých slov, například z původního hostess v levém horním rohu bylo vytvořeno tostess a následně nostess.

Kapitola 8

Závěr

Cílem práce bylo demonstrovat využití formálních modelů, konkrétně dvoudimenzionálních automatů, při tvorbě vizuálního umění. K tomu bylo zapotřebí navrhnout automat, jeho propojení se vstupem a vybrat vhodná umělecká díla.

Nejdříve jsem nastudovala pojmy spjaté s návrhem formálních modelů, dále jsem se zaměřila na obecné definice konečných automatů, jednodimenzionálních a dvoudimenzionálních, a našla konkrétní typy dvoudimenzionálního automatu, které sloužily jako inspirace pro výsledný návrh.

Nový typ automatu jsem založila na definici obecného dvoudimenzionálního automatu. Jako vstupní abecedu jsem zvolila latinskou s arabskými číslicemi, navrhla jsem převod pole znaků nalezených v obraze na vstupní pole automatu, které vytváří počáteční stav. Definovala jsem sadu tranzitivních funkcí, které automat nabízí, a pojmenovala jsem efekty, které vytváří, jako náhodný, náhodný s omezením, permutace a nezávislý. Každý přesouvá prvky jiným způsobem, ale všechny je spojuje využití výběru s náhodným rozdělením pravděpodobnosti při určení přesouvaného prvku a cílové buňky. Poslední tranzitivní funkce přepíná automat do koncového stavu.

Dále jsem nalezla možné vstupní obrazy podle toho, jakou roli v nich hraje text tvořený vstupní abecedou automatu. Seznámila jsem se s různým využitím textu ve vizuálním umění, prostudovala jsem tvorbu a základní myšlenky některých umělců, kteří ve svých dílech pracují s textem. Nakonec jsem zjistila, že nejvýraznější roli zastává text na plakátech, protože ty mají za cíl ovlivnit velké množství lidí. Vybrala jsem konkrétní příklady vhodné pro demonstraci vlivu automatu na vizuální umění.

Propojení obrazu s automatem pomocí algoritmu pro rozpoznávání znaků jsem založila na obecném schéma OCR systému. Seznámila jsem se s kroky, které jej tvoří, a nalezla možnosti předzpracování obrazu vedoucí ke zpřesnění výsledku. Získané poznatky jsem využila při návrhu procesu rozpoznávání založenému na metodách knihoven opencv a tesseract.

Návrhy automatu a rozpoznávacího algoritmu jsem poté převedla do modulů, přidala jsem prvky grafického uživatelského rozhraní, a vytvořila tak sadu skriptů, které při spuštění pomocí interpretu pro Python 3.10 zobrazí okno aplikace, kterou lze využít pro praktickou ukázku navrženého řešení. Vstupem aplikace je obraz, na kterém jsou nalezeny znaky tvořící počáteční stav automatu. Pomocí tlačítek pro výběr tranzitivní funkce lze dále řídit jeho činnost, v okně aplikace je možné sledovat jeho stav a výsledné dílo se dá stiskem tlačítka uložit.

Aplikaci jsem testovala na různých vstupních obrazech. V hodnocení jsem se nejdříve zaměřila na výsledky rozpoznávacího modulu, podle kterých jsem poukázala na možné nedostatky a jejich řešení. Následně jsem na různých uměleckých dílech demonstrovala

činnost automatu při zvolení jednotlivých efektů. Na výsledných obrazech znázorňujících stav automatu bylo patrné, že jeho činnost probíhá podle očekávání. Největší prostor pro vylepšení vidím v oblasti rozpoznávacího modulu, ale i automat nabízí možnosti pro budoucí rozšíření.

Navržené úpravy stávajícího řešení se dělí na dvě kategorie, první z nich je rozpoznávání. Cílem by bylo zvýšit jeho přesnost nebo uvolnit omezení vstupu. Mohl by být navržen nový algoritmus předzpracování obrazu, případně využita jiná knihovna pro samotné rozpoznávání.

Další rozšíření se týká samotného automatu. Dalo by se uvažovat o jiném tvaru buněk nebo jejich velikosti, která by se mohla přizpůsobovat podle obsaženého symbolu, nebo by se dalo pracovat s pozicí symbolů uvnitř buněk.

Co se týče činnosti automatu, mohly by být definovány nové efekty nebo efekty založené na již existujících. Příkladem by bylo přidání různých skupin, do nichž by symboly patřily a které by ovlivnily jejich pohyb. Mohly by být založené třeba na abecedě, nebo i počtu uzavřených kruhů a podobných grafických vlastnostech. Další modifikace by mohly přidat vliv kontextu. Takové efekty by upravovaly chování podle okolních symbolů například tak, že by se čísla vzdalovala od písmen a shlukovala s ostatními čísly, nebo že by automat kontroloval význam slov, která písmena při přesunu tvoří.

Další oblast vhodná pro rozšíření je vliv uživatele na výsledek. Dala by se přidat možnost editovat prvky, se kterými automat pracuje. Uživatel by tak mohl přidávat a odebírat znaky nebo měnit jejich postavení ve vstupním poli.

Grafická rozšíření by zahrnovala animace nebo tvorbu videa.

Dosud zmiňované možnosti mohou posloužit jako inspirace pro další vývoj nebo nové pojetí zadání. Příklad konkrétního projektu, který zde uvádím, je založený na přidání dimenze automatu. Efekty by mohly pracovat s obrazem a časem, aby vytvořily video jako ucelené nové dílo. Prvkům by tak bylo umožněno v některém časovém úseku úplně chybět a v jiném se objevit vícekrát. To by ale vyžadovalo možnost cestování zpět v čase, čehož by se dalo dosáhnout odložením zobrazení grafické reprezentace stavů automatu na dobu, ve které automat dokončil činnost. V takovém případě by bylo třeba nějakým způsobem uložit posloupnost vykonaných kroků nebo stavů automatu a až podle její konečné verze vygenerovat grafický výstup.

Literatura

[1] Powers of Persuasion. *National Archives* [online]. College Park, Maryland, US: The U.S. National Archives and Records Administration, 06. června 2019 [cit. 2022-03-28]. Dostupné z: https://www.archives.gov/exhibits/powers-of-persuasion.

[2] *Biography* [online]. BarbaraKruger.com, 2022 [cit. 2022-03-26]. Dostupné z: http://www.barbarakruger.com/biography.shtml.

[3] Letters and Words Coursework Guide. *TATE* [online]. London: Tate, 2022 [cit. 2022-03-26]. Dostupné z: https://www.tate.org.uk/art/student-resource/exam-help/letters-and-words.

[4] Guerilla Girls. *TATE* [online]. London: Tate, 2022 [cit. 2022-03-26]. Dostupné z: https://www.tate.org.uk/art/artists/guerrilla-girls-6858.

[5] Banner, F. *Break Point* [online]. 1998 [cit. 2022-03-26]. Dostupné z: https://www.tate.org.uk/art/artworks/banner-break-point-t07501.

[6] Chaudhuri, A., Mandaviya, K., Badelia, P. a Ghosh, S. Optical Character Recognition Systems for Different Languages with Soft Computing. *Studies in Fuzziness and Soft Computing*. Cham, Germany: Springer. Leden 2017, sv. 352. ISSN 1434-9922.

[7] eastWillow. *Open-CV Python Tutorials: Image Thresholding* [online]. červenec 2016 [cit. 2022-02-25]. Dostupné z: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html.

[8] Gardner, M. MATHEMATICAL GAMES. *Scientific American*. Scientific American, a division of Nature America, Inc. 1970, sv. 223, č. 4, s. 120–123. ISSN 0036-8733.

[9] Gardner, M. *The Colossal Book of Mathematics: Classic Puzzles, Paradoxes, and Problems*. 1. vyd. New York: W. W. Norton Company, 2001. ISBN 978-0393020236.

[10] Giammarresi, D. a Restivo, A. Two-Dimensional Languages. In: Rozenberg, G. a Salomaa, A., ed. *Handbook of Formal Languages, Vol. 3: Beyond Words*. Berlin: Springer, 1997. ISBN 978-3-642-63859-6.

[11] Guerilla Girls. *Which art mag was worst for women last year?* [online]. 1986 [cit. 2022-03-26]. Dostupné z: https://www.guerrillagirls.com/projects.

[12] Guerilla Girls. *It's even worse in Europe* [online]. 1989 [cit. 2022-03-26]. Dostupné z: https://www.guerrillagirls.com/projects.

[13] Hopcroft, J. E., Motwani, R. a Ullman, J. D. *Introduction to automata theory, languages, and computation*. 2. vyd. Addison-Wesley, 2001. ISBN 0-201-44124-1.

[14] Horáček, P., Meduna, A. a Tomko, M. *Handbook of Mathematical Models for Languages and Computation*. Stevenage, United Kingdom, GB: The Institution of Engineering and Technology, 2020. 750 s. ISBN 978-1-78561-659-4.

[15] Huamán, A. Eroding and Dilating. *OpenCV* [online]. 2022 [cit. 2022-02-25]. Dostupné z: https://docs.opencv.org/4.x/db/df6/tutorial_erosion_dilatation.html.

[16] Ibe, O. *Fundamentals of Applied Probability and Random Processes*. 2. vyd. Boston: Academic Press, 2014. ISBN 978-0128008522.

[17] Kruger, B. *Your body is a battleground* [online]. 1989 [cit. 2022-03-26]. Dostupné z: http://www.barbarakruger.com/art.shtml.

[18] Linz, P. *An Introduction to Formal Languages and Automata*. 6. vyd. Jones Bartlett Learning, 2016. ISBN 978-1-284-07724-7.

[19] Mageshwaran R. Survey on Image Preprocessing Techniques to Improve OCR Accuracy. *Medium* [online]. San Francisco: A Medium Corporation, 06. ledna 2021 [cit. 2022-02-27]. Dostupné z: https://medium.com/technovators/survey-on-image-preprocessing-techniques-to-improve-ocr-accuracy-616ddb931b76.

[20] Meduna, A. a Zemek, P. *Regulated Grammars and Automata*. 1. vyd. New York: Springer, 2014. ISBN 978-1-4939-0368-9.

[21] Nafiz, A. a Yarman Vural, F. T. An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. IEEE. Květen 2001, sv. 31, č. 2, s. 216–233. ISSN 1094-6977.

[22] Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. IEEE. 1979, sv. 9, č. 1, s. 62–66. ISSN 0018-9472.

[23] Phillips, T. *[no title]* [online]. 1970 [cit. 2022-03-26]. Dostupné z: https://www.tate.org.uk/art/artworks/phillips-no-title-p04952.

[24] Rice, S., Jenkins, F. a Nartker, T. The Fourth Annual Test of OCR Accuracy. [online]. Las Vegas: University of Nevada. Červenec 1995, [cit. 2022-02-27]. Dostupné z: http://www.expervision.com/testimonial-world-leading-and-champion-ocr/annual-test-of-ocr-accuracy-by-us-department-of-energy-doe-university-of-nevada-las-vegas-unlv.

[25] Schiff, J. L. *Cellular Automata: A Discrete View of the World*. 1. vyd. Hoboken, New Jersey: John Wiley & Sons, Ltd, 2007. ISBN 978-0470168790.

[26] Smith, R. An Overview of the Tesseract OCR Engine. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. IEEE, 2007, sv. 2, s. 629–633. ISSN 1520-5363.

[27] Smith, R. W. *The extraction and recognition of text from multimedia document images* [online]. Bristol, 1987. [cit. 2022-02-27]. Dissertation. University of Bristol. Faculty of Science. Vedoucí práce Lewis, E. Dostupné z: https://research-information.bris.ac.uk/en/studentTheses/the-extraction-and-recognition-of-text-from-multimedia-document-i.

[28] Smith, T. J. *Two-Dimensional Automata* [online]. Kingston, Ontario, Canada: School of Computing Queen's University, 2019 [cit. 2021-10-22]. ISSN 0836-0227-2019-637. Dostupné z: https://research.cs.queensu.ca/TechReports/Reports/2019-637.pdf.

[29] tesseract-ocr. *Tesseract documentation: Improving the quality of the output* [online]. Prosinec 2021 [cit. 2022-02-26]. Dostupné z: https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html.

[30] The Qt Company. *Qt Documentation: Coordinate System* [online]. The Qt Company Ltd., 2021 [cit. 2021-12-05]. Dostupné z: https://doc.qt.io/qt-6/coordsys.html.

Příloha A

Obsah přiloženého paměťového média

doc/ soubory tvořící dokumentaci

images/ použité obrázky

README.md soubor s postupem spuštění

requirements.txt seznam potřebných balíků

src/ zdrojové Python skripty

xzlevo00.pdf text bakalářské práce