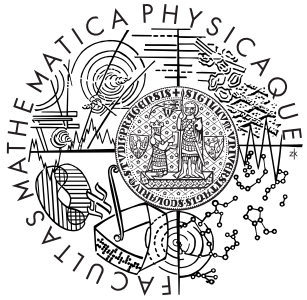
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Daniel Sipták

## Grafiti pro Google Android

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Ondřej Bojar, Ph.D.
Studijní obor: Správa počítačových systémů

2010

Na tomto mieste sa chcem poďakovať predovšetkým svojmu vedúcemu práce, doktorovi Ondřejovi Bojarovi za jeho cenné rady, pripomienky a zaujímavé návrhy, ktoré mi pomohli pri vypracovaní bakalárskej práce.
Ďakujem aj svojej rodine a priateľom, ktorí ma podporovali nielen počas písania bakalárskej práce, ale aj v celom doterajšom štúdiu.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne, výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 6. augusta 2010 Daniel Sipták

# Obsah

Názov práce: Grafiti pro Google Android
Autor: Daniel Sipták
Katedra: Ústav formální a aplikované lingvistiky
Vedúci bakalárskej práce: RNDr. Ondřej Bojar, Ph.D.
e-mail vedúceho: bojar@ufal.mff.cuni.cz

Abstrakt: V predloženej práci sa sa venujeme návrhu, rozboru a implementácii dotykovej klávesnice pre OS Android. Táto klávesnica bude používať jednoťahové grafity na zadávanie vstupných znakov. V práci rozoberáme možné riešenia a návrhy rôznych algoritmov na rozpoznávanie ťahu, ktoré taktiež testujeme na ich presnosť.

Kľúčové slová: Android, Grafiti, Rozpoznávanie znakov, Klávesnica

Title: Graffiti for Google Android
Author: Daniel Sipták
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Ondřej Bojar, Ph.D.
Supervisor's e-mail address: bojar@ufal.mff.cuni.cz

Abstract: The present study is dedicated to the design, analysis and implementation of software keyboard on Android OS. The keyboard will use customizable single-stroke letters to represnet input characters. In this paper we analyze possible solutions and propose various algorithms to recognize the stroke and we also test them for their accuracy.

Keywords: Android, Grafiti, Pattern matching, Keyboard

# Kapitola 1

# Úvod

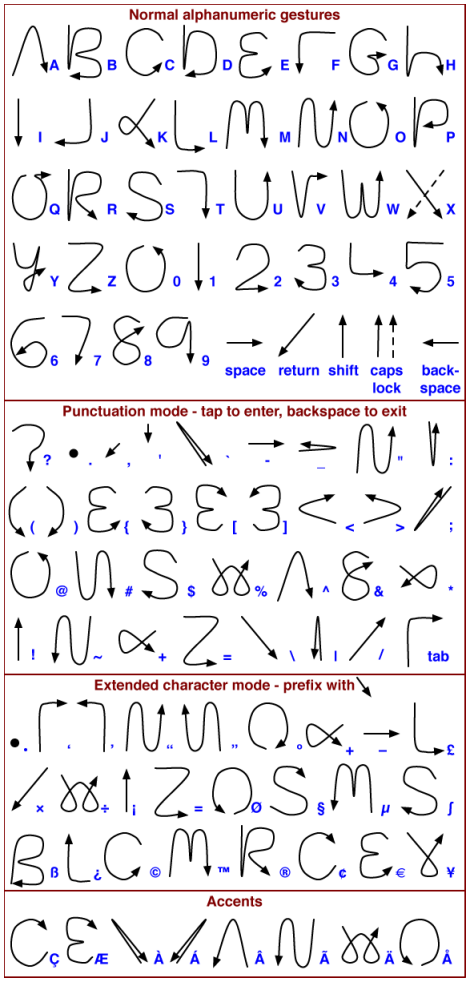
Používanie grafiti ako spôsob zadávania znakov na mobilnej platforme je účinná metóda ako optimalizovať rozpoznávanie na výkon a presnosť. Ich samotný tvar je jednoduchý na zapamätanie, aj keď sa niekedy líši od zaužívaných tvarov pre znaky. Naším cieľom je navrhnúť a vytvoriť túto metódu zadávania textu aj pre softverovú platformu Android, a to z dôvodu neexistencie voľne dostupnej aplikácie s podobnou funkcionalitou. Podobným aplikáciám a ich rozboru sa budeme venovať v prvej kapitole ako aj samotnej platforme Android a používanému vývojovému prostrediu. V druhej kapitole si upresníme, čo chceme vytvorit, vlastnosti a aké by mala aplikácia mať. Ďalej budeme analyzovať samotný problém rozpoznávania grafitu a navrhneme niekoľko algoritmov aj s možnosťami na ich zlepšenie. V štvrtej kapitole rozoberieme, ako sa nám podarilo vyvynúť samotný softvér a na aké problémy sme pri jeho realizácii narazili. Obsahom bude aj rozobranie detailov ohľadom konfigurácie celej aplikácie. V piatej a poslednej kapitole zhodnotíme ladenie a hlavné testovanie rozpoznávacieho algoritmu na množine testovacích ťahov, kde budeme sledovať správnosť rozpoznávania. V závere zhodnotíme celkové dielo, a to, ako sa nám podarilo naplniť naše ciele.

## 1.1 Grafity

Grafit je spôsob zadávania znaku – krátkym jednoduchým ťahom. Vznikol ako odozva na podobné systémy, ktoré mali nižšiu úspešnosť rozpoznávania písaného textu a veľkú časovú zložitosť. Grafity boli navrhnuté s ohľadom na jednoduchosť, rýchlosť a presnosť zadávania pre použitie na vreckových počítačoch s kapacitnýmm displejom. Dnes sú používané na rôznorodých zariadeniach s dotykovým ovládaním a taktiež na rôznych technológiách zachytávania dotyku.

Zakladná idea je zjednodušenie - takmer všetky znaky sú tvorené iba jedným ťahom (viz 1.1). Tým je zapezpečená ich dobrá odlíšiteľnosť a tiež je eliminovaný problém s rozpoznávaním začiatku a konca znaku. Toto napomohlo k zníženiu zložitosti identifikovania zadaného znaku, čím bola dosiahnutá väčšia rýchlosť samotného zadávania a komfort pre užívateľa. Nevýhodou grafitov je nutnosť oboznámiť sa so samotnými znakmi. Prvotný návrh počítal hlavne s použitím anglickej znakovej sady.

Preto až ich prispôsobením bolo možné zadávať znaky iných sád – napríklad znaky s diakritikou a podobne.




Obr. 1.1: Ukažka ťahovej sady Grafity

## 1.2 Analýza existujúcich riešení

V tejto časti sa budeme venovať už existujúcim podobným riešeniam, ktoré zhodnotíme z hladiska ich kladov a záporov. Taktiež si rozpíšeme vlastnosti, ktoré by sme chceli prevziať do našej aplikácie a ktorým sa naopak chceme vyhnuť.

### Graffiti (Palm OS)

*Grafiti* (viz 1.2) je spôsob zadávania jednotlivých písmen pomocou krátkych neprerušovaných ťahov jedným prstom.

Operačný systém Palm bol vývjaný spoločnosťou Palm pre systémy PDA (Personal Digital Assistant – vreckový počítač). Pre tento OS vyvynuli spôsob zadávania textu pomocou znakov písaných dotykom. Tento spôsob sa podobal klasickému písaniu textu, čo však spôsobovalo veľkú chybovosť pri rozpoznávaní znaku. Preto vyvynuli nový systém, ktorý bol postavený na zjednodušení zadávaných znakov – jednotlivé znaky boli zadávané osobitne jedným ťahom.

Toto zjednodušenie sa používa dodnes vo viacerých variantách a operačných systémoch.

Vzhľadom na to, že graffiti sú rozdielne od klasických písmen, pretože sú jednoťahové, užívateľ sa musí tieto znaky naučiť, čo môže niektorých užívateľov odradiť od ich používania.

**Výhody:**

- vysoká úspešnosť pri rozpoznávaní znakov
- rýchlejšie písanie textu oproti klasickému zadávaniu znakov
- menšia výpočtová a časová zložitosť – z toho plynúce menšie nároky na výkon zariadenia

**Nevýhody:**

- nutnosť naučiť sa používanú sadu znakov
- len súčasť OS Palm
- nekonfigurovateľnosť systému
- zastaralosť programu – softvér existuje len pre OS Palm, ktorý sa už nenasadzuje


Obr. 1.2: Palm Graffiti

# MobileWrite

MobileWrite je program na rozpoznávanie znakov vložených dotykom. Slúži ako alternatíva k obrazovkovej alebo klasickej klávesnici.

V programe MobileWrite (viz 1.3) je možnosť používať štandardné sady znakov (vyzerajúce ako tlačené písmo) alebo Graffiti sady. Pomocou štandardnej sady je možné písať kapitálky a malé písmená rovnako ako na papier, čo zjednodušuje prácu s aplikáciou. Použitím Graffiti sady môžu byť dáta rozpoznané rýchlejšie a s väčšou presnosťou.

Podporované jazyky: francúzština, nemčina, taliančina, španielčina, holandština a portugalčina. Najväčšou nevýhodou tohto softvéru je nutnosť zakúpenia a taktiež nepodporovanie slovenských a českých znakov.

**Výhody:**

- multiplatformovosť (Android, Palm OS, Windows Mobile)
- možnosť výberu zadávania typu znakov (štandard, Graffiti)

**Nevýhody:**

- platená aplikácia
- nekonfigurovateľnosť systému
- jazykovo obmedzené sady znakov


Obr. 1.3: Mobile Write

## CellWriter

CellWriter (viz 1.4) je malý, jednoduchý nástroj na rozpoznávanie rukopisu, ktorý sa ľahko integruje do moderného prostredia Linuxu. Rozhranie tvorí mriežka, do ktorej sa píše klasickým rukopisom. Preto aj najlepšie funguje práve pri tenkých dotykoch – teda napríklad s tabletovým perom, ale nie je to nevyhnutné. Veľkou výhodou tejto aplikácie je jej schopnosť samoučenia – dokáže sa naučiť štýl písania jednotlivých užívateľov. Taktiež si môže užívateľ prispôsobiť štýl vstupného panelu, či vybrať z niekoľkých znakových sád – latinka, cyrilika, thajská, medzinárodná fonetická rozšírená abeceda a iné. Projekt CellWriter bol vyvýjaný na Minesottskej Univerzite študentom počítačovej vedy a umelej inteligencie.

**Výhody:**

- schopnosť učenia sa
- podpora špeciálnych znakov
- open source – voľne šíriteľné zdrojové kódy

**Nevýhody:**

- prispôsobené len na platformu linux
- nutnosť písať v blokoch


Obr. 1.4: CellWriter

## 1.3 Android

Android je softvérová platforma obsahujúca operačný systém, prostredie a základné aplikácie. Primárne je určený pre mobilné zariadenia. Pre viac informaci [1]

Vyvýjaný je ako Open Source pod záštitou spoločnosti Google Corporation. Vývoj začal v roku 2007 a od tej doby prebieha neustále pridávanie nových vlastností systému, ktorý čiastočne spôsobuje problém s vyšším počtom verzií systému. Tento problém je však dobre eliminovaný spätnou kompatibilitou a distribučným kanálom Android Market.

# Android architektúra

Táto softvérová platforma je založena na rôznych technológiách

1. jadro operačného systému

   GNU linux vo verzi 2.6 .

2. aplikačná vrstva

   dovoľuje zdieľanie a znovupoužitie súčastí programov.

3. dalvik virtuálny stroj

   je optimalizovaný pre mobilné zariadenia a vytvára virtualizované prostrenie pre každú aplikáciu užívateľa.

4. SQLite

   jednoduché databázové úložisko prístupné pre každú aplikáciu

5. vývojové prostredie

   bohaté vývojové prostrenie obsahujúce emulátor zariadenia, nástroje na vývoj a ukážkové aplikácie.

Užívateľská aplikácia beží vo virtualizovanom prostredí, čím sa zabezpečuje odlúčenie od systému a bezpečnosť platformy. Preto aplikácia môže komunikovať so systémom iba cez navrhnutú aplikačnú vrstvu. Pri inštalovaní sa aplikácii udelia presne špecifikované práva, čím je riadený prístup do citlivých dát alebo súčastí systému.

Obrázok x.: Android platforma

# Inštalácia android aplikácie

Aplikácie na platformu Android su dodávané v podobe balíčku s koncoukou *.apk* . Tento balíček v sebe obsahuje všetky potrebné súčasti aplikácie od konfigurácie cez zdrojový kód až po multimediálny obsah (obrázky, zvuky, atď...). Takýto balíček je nainštalovateľný systémom do zariadenia.

V našom prípade budeme používať priamu inštaláciu cez vývojové prostredie, ktoré automaticky vytvorí a aj nainštaluje tento balík na zariadenie. Ďalšou možnosťou je takto vytvorený balík inštalovať manuálne priamo na zariadenie. Pri nasadení

aplikácie do ostrého používania sa počíta s použitím Android marketu. Táto aplikácia poslúži ako kompletný distribučný kanál so zabezpečením, doručením aj inštaláciou aplikácie. Pre takúto distribúciu je nutné zaregistrovať sa do systému Android Market a podpísať aplikačný baliček digitálnym podpisom.

## Eclipse – vývojové prostredie

Eclipse je integrované vývojové prostredie určené pre programovanie v jazykoch Java a C++ (viac v zdroji [2]). Po nainštalovaní samotného prostredia stiahnuteľného zo stránok
*http://www.eclipse.org/*, je ešte nutné doinštalovať SDK (software development kit – softvérový balík pre vývoj) od Google Corporation, ktorý sa nachádza na stránkach Android development *http://developer.android.čom/sdk/*. Na tejto anglickej stránke sa nachádza podrobný návod ako toto prostredie kompletne nainstalovať a spúšťať. Nainštalované SDK v sebe integruje všetky potrebné knižnice a dostupné zdroje multimediálnych dát ako sú obrázky, ikony, grafické témy, zvuky a iné. S týmto SDK sa doinštaluje aj virtualizované prostredie na testovanie vyvýjanej aplikácie, ktoré takmer v plnom rozsahu napodobňuje samotné zariadenie, na ktorom bude aplikácia používaná. Nástroj na ovládanie tohto virtuálneho stroja funguje aj na ovládanie zariadenia pripojeného na vývoj aplikácie cez USB. Týmto spôsobom je možné počas vývoja používať nielen virtuálne prostredie, ale aj reálne zariadenie. Výhodou virtualizovaného stroja je možnosť vytvoriť zariadenia s rôznymi konfiguráciami a v rôznych verziách používaného OS Android. Takto je možné aplikáciu dôkladne otestovať ešte pred samotným umiestnením na Android Market. Toto do značnej miery uľahčuje vývoj aplikácie, nakoľko nie je nutné priamo vlastniť zariadenie pre vyvýjanú aplikáciu. Pri použití oboch možností pre vývoj aplikácie je možné sledovať proces ovládajúci aplikáciu ako aj logovacie hlásenia pochádzajúce zo zariadenia.

## Výber verzie Android API

Výber používanej verzie je veľmi dôležitý pre prístupnosť istých funkcií v systéme. Dopredná kompatibilia je takmer bezproblémova. Takže pokiaľ nie je potrebné používať novšie funkcie Android API, nie je nutné vývjať pre vyššiu verziu.

Náš výber verzie bol silne ovplyvnený aj dostupným zariadením pre vývoj, a tým bol *HTC HERO* s verziou systému *Android 1.5*.

Preto je naša aplikácia stavaná tak, aby bola plne kompatibilná so zariadeniami, ktoré maju aspoň túto verziu.

## Mierené zariadenia

Aplikácia je mierená na dotykové zariadenia, ale nie je priamo obmedzená metódou vstupu. Vyvýjaná bola na zariadeni s kapacitným displejom o rozlíšení 480x320. Predpokadané zariadenie je mobilný telefón s rôznym rozlíšením obrazovky a nie je špecifikované, či obsahuje aj fyzickú klávesnicu.

# Kapitola 2

# Špecifikácia projektu

## 2.1 Návrh programu

**Program z pohľadu užívateľa bude rozdelený na dve časti:**

1. Uživateľské prostrednie s dotykovou plochou slúžiacou ako klávesnica.
2. Uživateľské menu s možnosťami základných nastavení a získania informácii o programe, prípadne jeho použití.

V oboch častiach aplikácie dbáme na jednoduchú štruktúru a na možnosť použitia aplikácie nie len vo vertikalnej, ale aj v horizontálnej polohe zariadenia. Držíme sa štýlu celej platformy použitím interných súčastí na zobrazenie prvkov.

## Užívateľské prostredie pre dotykovú plochu

V tejto časti aplikácie chceme používateľovi poskytnúť čo najväčšiu plochu na samotné zadávanie ťahou a pritom mu dodať všetky potrebné informácie a možnosti. Medzi potrebené informácie patrí aktuálny stav dopredného modifikátora (shift, capslock) a rozčlenie dotykovej plochy na definované časti pre rozpoznávanie rôznych druhov znakov (písmená, diakritika, číslice...).

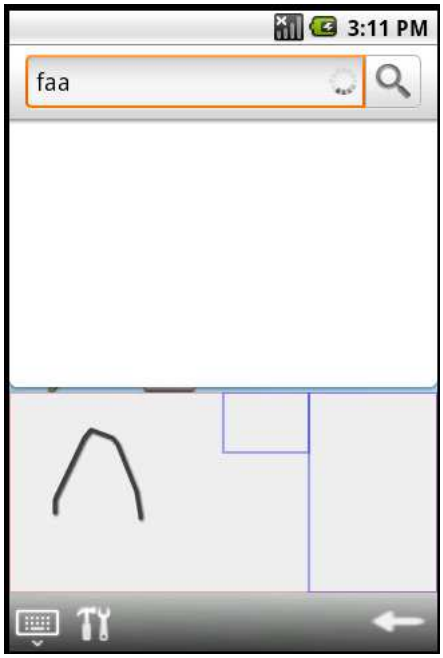
Rozhranie nemá pevne stanovenú plochu ani vyhradenú časť displeja. Preto si aplikácia musí zvoliť túto výšku sama. Túto časť rozdelíme na dve časti a to na nástrojovú lištu umiestnenú v dolnej časti aplikácie a dotykovú plochu umiesntnenú v ostávajúcom priestore.

**Na nástrojovej lište sa nachádzajú:**

- odkaz na zasunutie virtuálnej klávesnice
- odkaz na prístup do nastavení aplikácie
- stlačiteľné zobrazenie stavu dopredného modifikátora, ktorý po kliknutí zmení stav na nasledujúci
- tlačidlo pre mazanie posledného zadaného znaku

**Na dotykovej ploche sa nachádza:**

- graficky rozdelené plochy rozpoznávania. Definované plochy su rozlíšené farebným okrajom zafarbeným definovanou farbou. (viz 2.1)




Obr. 2.1: Dotyková plocha

## Uživateľské menu

Tu chceme dať užívateľovi niekoľko možností ako získať informácie alebo ako nastaviť niektoré voľby.

**Potrebné súčasti:**

- vybrať a načítať novú konfiguráciu aplikácie.
- uložiť aktuálnu konfiguráciu aplikácie.
- krátke zoznámenie sa s aplikáciou a s jej použitím.
- zobrazenie informácií o aplikáci (autor, podmienky použitia, atď'...).
- zobrazenie rozpoznávaných ťahov s ich významom.
- možnosť úpravy ťahu.

## Hlavne súčasti programu

Program bude zložený z viacerých oddelených súčastí, ktoré budú vzájomne komunikovať. Niektoré tieto súčasti budú môcť byť zamenené podľa potreby.

**Plánovane súčasti:**

- Hlavný konfiguračný objekt

  Tento objekt bude vytváraný pri načítavaní konfigurácie ktorá bude do neho uložená. Táto ústredná časť bude držať všetky informácie potrebné pre ďalšie súčasti programu. Ďalej bude obsahovať odkazy na ostatné súčasti programu, ktoré aj sám inicializuje.

- Trieda na komukáciu s klavesnicovým subsystémom.

  Táto trieda bude takzvaný IME (Input Method). Slúžiť bude na komukáciu so systémom ohľadom naviazania plochy klávesnice na obrazovku a na oznamovanie rozpoznaných znakov späť do systému. V tejto triede možu byt neskôr implementované aj pomocné funkcie pre písanie.

- Dotyková plocha

  Trieda určená na vykreslenie dotykovej plochy a zachytávanie dotykov prichádzajúcich od vyšších súčastí systému. Pri každom dotyku vytvára nový objekt, ktorý posiela na spracovanie odchytovej triede.

- Odchytová trieda

  Trieda bude preberať príchodzí dotykový objekt od dotykovej plochy. Spracuje ho a predpripraví na spracovanie vyhľadávacím algoritmom. Dostane výsledný dotykový objekt rozpoznaný vyhľadávacou triedou a podľa rozpoznaného spojenia vykoná zodpovedajúcu akciu.

- Vyhľadávacia trieda

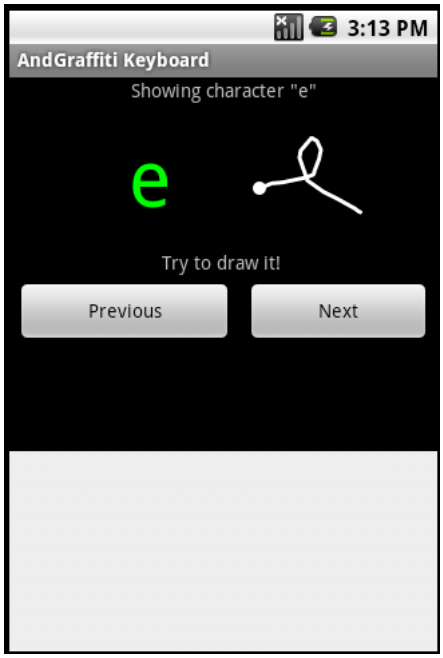
  Trieda určená na implementáciu samotného vyhľadávacieho algoritmu. Ako vstup dostáva predspracovaný ťah a zoznam použiteľných vzorov. Výstup bude dotykový objekt, kde už bude poznačená hodnota spojenia, ktoré bolo priradené k rozpoznanému dotyku. Táto trieda bude mať viac implementácií, ktoré budú dediť od základného algoritmu. Takto bude možne v programe určiť, aký algoritmus sa aktuálne používa.

- Dotykový objekt

  Tento objekt bude vytváraný pri začiatku dotyku na ploche displeja a bude zrušený po vykonaní príslušnej akcie plynúcej z jeho rozpoznaného spojenia. Bude obsahovať niekoľko položiek, napr. čisté dáta o dotyku, plochu, pre ktorú vznikol, predspracované informácie o dotyku, výsledné spojenie, ktoré bolo identifikovateľné.

- Učiaci režim

  V tejto časti pôjde o spoluprácu niekoľkých tried, ktoré spoločne vytvárajú možnosť ukázať a otestovať správne zadanie ťahu. Tiež budú po prepnuti schopné učenia nového vzoru pre vybrané spojenie. A to zadaním nového tvaru a jeho odsúhlasením. (viz 2.2)




Obr. 2.2: Dotyková plocha

## 2.2 Konfigurácia programu

# Analýza konfigurácie

Ako cieľ práce sme zadali aj vytvorenie konfigurácie v externe editovateľnom textovom súbore.

**Požiadavky na konfiguráciu sú:**

- možnosť definovať viac plôch s rôznou množinou vzorov.
- možnosť definovať viac stavov, ktoré slúžia ako modifikátory obsahu množín stavov pre danú plochu – sú to tzv. dopredné modifikátory ako (Shift, CapsLock, ...).
- možnosť k zadanej ploche pri zadanom stave a vstupnom vzore určiť výstupnú akciu.

- možnosť používať všetky znaky abecedy.
- možnosť spätných modifikátorov znaku ako napr ´ ˇ
- možnosť cez konfiguráciu nastaviť správanie programu
- možnosť definovať pre ten isty výstupný reťazec viac vzorov

Konfigurácia by mala byť ľahko pochopiteľná a editovateľná priamo v textovom editore. Editácia cez prostredie programu nie je žiadaná pre jeho zbytočné skomplikovanie prostredia.

**Označenie súčastí konfigurácie:**

- nastavenia

  Konfiguračná časť určená pre globáne nastavenia týkajúce sa celého programu. Prvky tejto časti obsahujú názov premenej, čize kľúč a jej hodnotu, čiže obsah.

- plochy

  Časť pre definovanie plôch a ich vlastností. Medzi ich vlastnosti by mala partriť ich identifikácia, názov, pozícia a ďalšie atribúty.

- stavy

  Časť pre definovanie stavov dopredných modifikátorov textu. Ich vlastnosťami by mali byť identifikácia a názov.

- spojenia

  Časť pre definovanie spojení. Spojenie je záznam, ktorý spája danú plochu s daným stavom a vzorom a určuje novú akciu a jej parameter.

- vzory

  Časť konfigurácie, kde sa ku zadanému identifikátoru vzoru priradí samotný znak používaný aktuálnym algoritmom. Mal by obsahovať aj pre človeka prijateľnú podobnu vzoru, ktorý znak reprezentuje. Napríklad v podobe ťahu.

Pre lepšie pochopenie je tu obrázok 2.3.

**Ďalšie pojmy:**

- znak

  hodnota vzoru v konfigurácii, ktorá je určená na rozpoznávanie v rozpoznávacom algoritme

- akcia

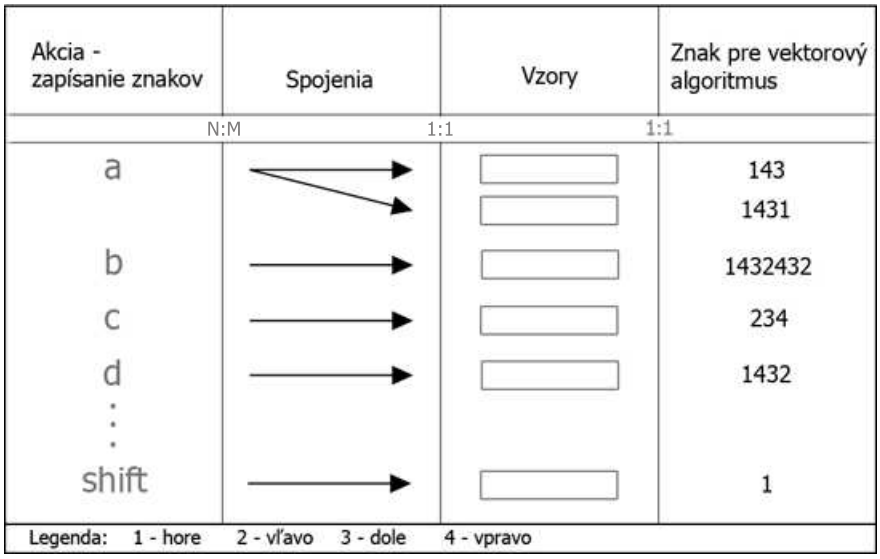
  rôzne akcie, ktoré program vykoná po úspešnom rozpoznaní ťahu.

- editovaný text

  aktuálny text, ktorý obsluhuje klávesnica.

**Typy akcií:**

- vypísanie znaku do editovaného textu
- zmena stavu
- špeciálne modifikátory (Backspace, Enter)
- spätné modifikovanie posledného znaku editovaného textu.




Obr. 2.3: Ilustracia opisujuca vzťahy medzi súčasťami

# Kapitola 3

# Algoritmy

## 3.1 Analýza algoritmu rozpoznávania

V tejto časti práce analyzujeme problém identifikovania zadaného ťahu. Zavedieme niekoľko pojmov, ktoré budeme ďalej v analýzach a návrhoch používať. Pri riešení tohto problému vytvoríme základné podmienky a vlastnosti, ktoré by mal vytvorený algoritmus spĺňať.

## Vstup algoritmu

Vstup algoritmu je generovaný na vstupnej dvojrozmernej ploche. Vstupom algoritmu je lubovoľne dlhá postupnosť dvojíc súradníc. Takúto postupnosť nazveme ťah. Ťah nemá pevne stanovený rozptyl vzdialeností susedných pozícií. Každú dvojcu súradníc označujúcu miesto no vstupnej ploche, nazveme ako pozíciu ťahu. Pozície budeme označovať malými písmenami napr. $a, b, c...$ a celý ťah veľkými písmenami napr. $A, B, C...$, kde identifikátor ťahu $A$ je jednoznačný. Pre určenie danej pozície a súradnice v ťahu budeme používať konvenciu $A.b.X$, kde v ťahu $A$ zobrieme pozíciu $b$ a súradnicu $X$. Množín všetkých vstupnych ťahov budeme nazývať $T$. Ďalším vstupom algoritmu je množina vzorov $V$. Celú množinu vzorov nazývame $M$, kde vzor je tvorený znakom a jednoznačným číselným identifikátorom. Pod znakom vzoru rozumieme štrukturu, ktorá bude závislá na použitom algoritme. Vzor budeme označovať malými písmenami napr $x, y, z...$ a ich znak ako $x.s$. Pre značenie identifikátora budeme využívať zápis $x.i$, ktory bude jednotný pre každý použitý algoritmus.

# Úloha algoritmu

Algoritmus má za úlohu čo najefektívnejsie, najpresnejšie a s čo najmenším počtom chýb priradiť ťah $A$ ku vzoru z domény vzorov. Pre tieto prvky platí, že ťahy sú ku vzorom v pomere $N : 1$. To znamená, že pre každý ťah existuje maximálne jeden správny vzor z domény. Vstupný ťah nemusí zodpovedať žiadnemu vzoru. Naopak, každému vzoru by mala zodpovedať neprázdna množina ťahov, ktoré by mali byť pre každý vzor navzájom disjunktné. Toto môžeme popísať funkciou $f(A, V) = x$ . Ide o idealnu funkciu priradenia ťahu ku vzoru. Táto funkcia ku každému ťahu $A$ priradí práve jeden správny prvok z množiny vzoru $V$ alebo nepriradí žiaden prvok, pokiaľ ťah nezodpovedá žiadnemu vzoru z množiny $V$.

Algoritmus pomenujeme ako funkciu $g(A, V) = x$, čo je čo najlepšia aproximácia funkcie $f$.

Problém môžeme definovať ako nájdenie spoločného znaku pre prvky jednej disjunktnej podmnožiny $P$ množiny $T$. Tento znak priradzuje všetkým ťahom z tejto podmnožiny $P$ rovnaký vzor $x$. Takúto transformačnú funkciu nazveme funkcia $t$, definujeme ju ako $AeM, t(A) = x.s$ . Táto transformačná funkcia zmení vstupný ťah na znak vo vzore. Ďalšia funkcia $V \in M, y \in V, p(x.s, V) = y$. Budeme ju nazývat priradzovacia funkcia. Táto funkcia priradí ku znaku odpovedajuci vzor, pričom nemusí platiť, že $x.s = y.s$.

Pod transformačnou funkciuou si môžeme predsatviť funkciu, ktorá ku vstupnému ťahu priradí jej charakteristiku ako napriklad dĺžku, tvar, smer, rozdelenie ... Pod pridradzovaciou funckiou si môžeme predstaviť funkciu, ktorá porovná charakteristiky ťahu z transformačnej funkcie a z množiny vzorov, priradí im prevdepodobnostné hodnoty zhody a na výstup pošle tú najpravepodobnejšiu.

Týmto máme hotovú základnú analýzu problému, z ktorej sme pôvodný problém rozložili na potrebu dvoch funkci: $t(A) = x.s$ a $p(x.s, V) = y$ . Tieto nám spoločne dajú vyslednú hľadanú aproximačnú funkciu

$$g(A, V) = p(t(A), V) = x$$

.

## Vlastnosti transformačnej funkcie

$$A \in M, t(A) = x.s.$$

Úlohou tejto funckie je zjednodušit vstupné ťahy. Vytvorený znak by mal byť čo najjednoznačnejší a najpodobnejší pre každý prvok disjuktnej podmnožiny $P$ množiny $T$, pre ktorú platí, že

$$\exists x \in M, A \in P, f(A, M) = x$$

. Pre funkciu t platí, že samotný znak $x.s$ nemusí byť pre každý prvok $P$ rovnaký, ale mal by čo najpresnejšie popisovať charakteristické vlastnosti ťahu $A$.

## Vlastnosti priradzovacej funckie

$V \in M, y \in V, p(x.s, V) = y$

Táto funkcia vyberie najpodobnejši prvok z v ku znaku $x.s$. Funkcia môže prvok aj nepriradiť; to znamená, že sa jej nepodarilo nájsť správne priradenie. Funkcia môze spôsobovať chybovosť - priradenie zlého vzoru $y$ ku znaku $x.s$. Preto táto funckia musí byt navrhunta s ohľadom nielen na maximalizáciu správnych priradení, ale aj na minimalizáciu chybných priradení.

## Úspešnosť rozpoznávajúcej funckie

Tu definujeme funkciu, ktorá bude vyhodnocovať úspešnost algoritmu. Túto funkciu využijeme neskôr pri testovaní implementovaných algoritmov.

Funkciu h nazveme úspešnosť algoritmu a definujeme ju pre vzory $x, y$.

$$A \in T, V \in M, \exists x = f(A, V), g(A, V) = y$$

- Ak $h(x, y) = 1$ algoritmus uspel
- Ak $h(x, y) = 0$ algoritmus spôsobil chybu
- $y$ neexistuje algoritmus nerozpoznal ťah

## 3.2 Charakteristické vlastnosti ťahu

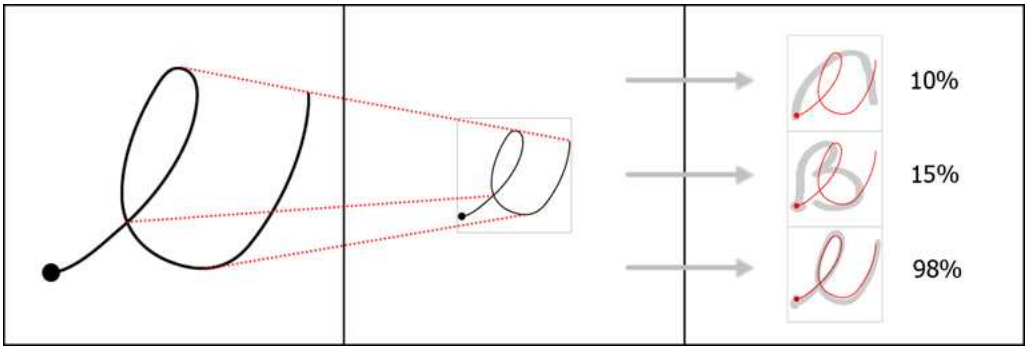
Sú to vlastnosti, ktoré nám pomôzu odlíšit rôzne ťahy a určiť im ich správny vzor. Týchto vlastností je viac, ale všeobecne ide o tvar, veľkosť, orientáciu, smer, krivosť a podobne. Tú si rozoberieme niekoľko takýchto vlastností a pokúsime sa vybrať najprespektívnejšie z nich na použitie v neskôr navrhovaných algoritmoch.

### Smer vstupného ťahu

Pod smerom ťahu si predstavíme rozdelenie ťahu na rôzne smery, ktorými sa ťah uberá. Takáto vlastnosť by hodnotila postupne celý ťah a sledovala by smer, v ktorom je ťah kreslený. Pri výraznej zmene smeru by zapísala nový smer, a tým by vytvárala postupnosť smerov, ktorými je daný ťah tvorený. Takáto postupnosť by mala byť, pokiaľ sa dokáže rozpoznať dostatočne presne, takmer jednoznačná pre každý typ ťahu. Preto túto vlastnosť budeme ďalej skúmať a využijeme ju pri návrhu vektorového algoritmu rozpoznávania. Navrhované pomocné vlastnosti ako orientácia alebo krivosť by v prípade tohto algoritmu pravdepodobne moc neuplatnili, kedže algoritmus veľmi prísne priradzuje znak vzoru ku ťahu, preto zmenšenie domény by nemalo príliš veľký efekt. Pri neskoršom porovnávaní rozpoznaného znaku a znakov z množiny vzoru by sa mohla uplatniť určitá tolerancia, ktorý by pri správnom použití mohla zvýšiť množstvo rozpoznáných znakov.

### Tvar vstupného ťahu

Táto vlastnosť by brala do úvahy celkový tvar ťahu. Ten by vyhodnocovala za pomoci transformácie do definovaného rozmeru a za použitia kritérii na veľkosť a deformáciu. Pre takto pretransformovaný ťah by sme pri vykreslovaní do predpripravenej bitovej mapy znaku zo vzoru počítali percentuálne prekrytie zadaného a porovnávaného ťahu podľa obrázku 3.1. Takýto algoritmus by sa dal použiť ako základ celkovej metódy rozpoznávania, pretože má možnosť rozpoznať všetky navrhované ťahy aspoň v dostatočnej presnosti. Preto tento algoritmus zdokonalíme a pridáme mu ďalšie zlepšenia v rastrovom algoritme rozpracovanom nižšie. Ako navrhované zlepšenia by mohlo byť použitie niektorých metód na zmenšenie množiny porovnávaných vzorov za použitia hore uvedených vlastností. Vyhodnocovanie tejto vlastnosti može byť o dosť výpočtovo náročnejšie ako hore uvedené vlastnosti, pretože pracuje s rastrom nakresného ťahu.




Obr. 3.1: Postup rozpoznávania tvaru

## Veľkosť vstupného ťahu

Pod veľkostou ťahu budeme rozumieť pomer jeho výšky a šírky k veľkosti celej plochy pre zadávanie.

Takéto rozlíšenie by sme mohli použit na určenie veľkosti zadávaného znaku viz 3.2. Napríklad malé alebo velké písmená a podobne. Určenie veľkosti ťahu by nebolo zložité - jednalo by sa iba o spočítanie pomeru. Ale takúto funkcionalitu nepožadujeme, chceme ju robiť iným spôsobom pomocou doprednych modifikátorov. Tu by mohol nastať problém so schopnosťou zaznamenávat ťahy dostatočne malé, aby užívateľ dával jasne najavo, že chce písat malými písmenami.

## Orientácia vstupného ťahu

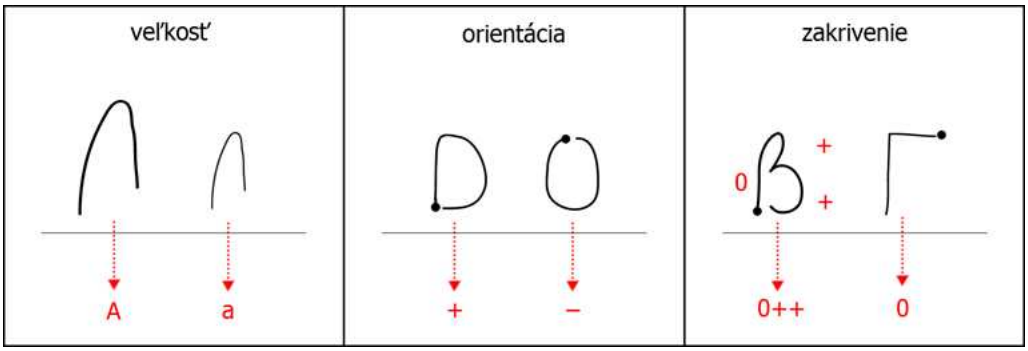
Pod orientáciou ťahu si predstavíme vlastnosť dotyku, ktorá bude popisovať, či bol vytváraný v smere hodinových ručičiek alebo v proti smere viz 3.2.

Pre ťahy, ktoré sú bez takejto orientácie (napríklad I), budeme túto hodnotu vyhodnocovať ako neurčenú. Výpočet takejto vlastnosti by bol založený na zistovaní počtu bodov nad alebo pod spojnicami sesedných bodov alebo spojnicou počiatočného a koncového bodu. V prípade viditeľného rozdielu jedným smerom im bude priradená hodnota orientácie. Týmto spôsobom budeme zmenšovať množinu testovaných vzorov ku rozpoznaným znakom. Použiť sa bude dať s algoritmom, ktorý bude používať percentuálne hodnotenie zhody ťahu a znaku vzoru. Takýmto algoritmom bude napríklad rastrový algoritmus.

## Krivosť vstupného ťahu

Táto vlastnosť je určitým rozšírením hore uvedenej vlastnosti. Ide o počítanie celkovej krivosti ťahu.

Taktiež by sa podobala orientácii, ale nepočítali by sme smer zakrivenia, ale hodnotu, akou je ťah zakrivený viz 3.2. Toto by platilo pre jednu aj druhú orientáciu - vhodné hlavne na odlišovanie znakov ako B,C,D,G. Táto vlastnosť by zmenšovala množinu testovaných vzorov voči ťahu. Takže by mala podobné využitie ako predošlý algoritmus.




Obr. 3.2: Ukážka vlastností ťahov

## Vzdialenosť počiatočnej a konečnej pozície

Táto vlastnosť môže mať viac podôb, ale vždy je zameraná na zhodnotenie vzájomnej pozície počiatočnej a konečnej pozície. Jedna možnosť je iba detekovať relatívnu vzdialenosť medzi danými bodmi, kde tento výpočet je výpočtom euklidovskej vzdialenosti. Ďalej môže mať rozširenie v podobe určenia, či sú body vo vzájomnej polohe horizontálne alebo vertikálne. Prípadne to rozšíriť na väčší počet smerov a ich presnosť. Limit takéhoto delenia by sa odvádzal od rozmerov ťahu.

## 3.3 Vektorový algoritmus

V tejto časti sa budeme venovať návrhu prvého samotného rozpoznávacieho algoritmu. Tento algoritmus bude založený na vlastnostiach smeru vstupného ťahu uvedeného vyššie.

Algoritmus bude návrhom ako transformačnej funkcie,

$A \in M, t(A) = x.s$ ,

tak aj príslušnej priraďovacej funkcie

$V \in M, y \in V, p(x.s, V) = y$

**Transformačná funkcia**

Transformačná funkcia tohto algoritmu vytvára k vstupnému ťahu výstupný znak podľa pevne daného postupu, ktorý dá pri tom istom vstupe ten istý výstup.

Funkcia bude priradzovať určitým častiam ťahu ich smer a postupnosť smerov, ktoré takto vzniknú, budú výstupný znak ťahu. Priradzovanie smerov ku častiam ťahu sa bude diať v lineárnom čase vzhľadom na množstvo pozícií v ťahu. Funkcia bude postupne prechádzať všetky pozície a aplikovať na ne pomocnú funkciu, ktorá priradzuje symbol smeru.

Počet smerov, na ktoré sa bude rozlišovať, bude pevný, ale bude určený až pri testovaní. Odhadovaný počet smerov sa pohybuje od 4 do 8. Pre naše potreby zoberme len 4 smery, v prípade väčšieho počtu bude algoritmus fungovať analogicky. Smer ohodnocuje relatívnu zmenu súradníc medzi pozíciami.

Ako hlavná časť funkcie prechádza nové pozície ťahu a predáva starú a novú pozíciu pomocnej funkcii, tak si postupne vytvára znak ťahu reťazením symbolov smeru, ktoré dostane od pomocnej funkcie. Pre toto reťazenie bude platiť, že do znaku sa pridá nová hodnota iba ak došlo ku zmene hodnoty symbolu pre smer. Preto vo výstupe nebude sekvencia rovnakých symbolov za sebou viz 3.3.

Hlavnú funkciu zapíšeme ako $r(A) -> a, b...$ ,kde a<>b
Zápis pomocnej funkcie $q(A.a, A.b) = s$ , kde s je zástupný symbol znaku (h,d,p,l)

# Implementácia vektorovej funkcie

Hlavná funkcia priradzujúca ku vstupnému ťahu $A$ znak $x.s$

**Postup hlavnej funkcie:**

inicializácia - určenie minimálnych hraníc

1. načítanie nasledujúcej pozície
2. zavolanie pomocnej funkcie s predošlou a novou pozíciou
3. rozhodnutie o pridaní symbolu do vytváraného reťazca symbolov podľa posledného pridaného symbolu

ukončenie - vrátenie vytvoreného reťazca

Pomocná funckia dostáva na vstupe starú $A.a$ a novú $A.b$ pozíciu. K tejto pozíci sa snaží priradiť novú hodnotu smeru.

**Postup pomocnej funkcie:**

1. spocíta rozdiely súradníc medzi starým a novým ťahom a pripočíta preklenovacie hodnoty, horizontálne aj vertikálne zvlášť
2. rozhodnutie podľa veľkosti zistených rozdielov

   Rozdiel pozícii je príliš malý v nejakom rozmere. Algoritmus vráti staré hodnoty smeru a zapamätá si preklenovacie hodnoty, ktoré sa rovnajú aktuálnym rozdielom.

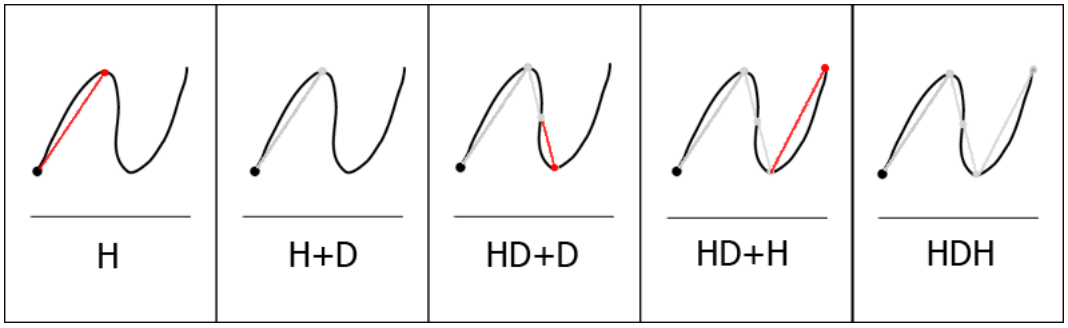
   Rozdiel spĺňa stanovené minimálne hranice. Tak podľa rozdielov zistí smer zmeny súradníc.
3. zistený smer vráti hlavnej funkcii

Minimálne stanovené hranice prichystá ešte hlavný algoritmus, ktorý ich určí ako dané percento výšky a šírky ťahu. Pre tieto hodnoty bude platiť absolútne minimum, ktoré bude nezávislé na veľkosti ťahu. Tieto hodnoty napomáhajú vyhladeniu vstupu, kedže sa musí počítať s určitou nepresnosťou súradnic priradzovaných do pozícii ťahov. A tak sa bráni opakovaným zmenám smeru na základe malého šumu. Takéto zmeny by silne ovplyvňovali vznikajúci reťazec a museli by sa filtrovať. Hodnoty

obidvoch premenných budú určené testovaním.

### Priradzovacia funkcia

Funkcia nechá spracovať prijatý ťah transformačnou funkciou, a znak, ktorý po transformácii dostane, sa bude snažiť priradiť ku správnemu vzoru z dodanej množiny vzorov $V$. Tak priradí k nájdenému znaku $x.s$ vhodný vzor $y$, kde $x.s$ sa nemusí rovnať $x.y$. Funkcia postupne porovnáva všetky znaky vzorov z množiny so vzorom ťahu a vyberie najvhodnejší vzor podľa daných kritérií.




Obr. 3.3: Ilustrácia postupného rozpoznávania znaku

### Kritéria priradenia

1. presná zhoda

   ukončí prehľadávanie a vráti nájdený vzor

2. zhoda zmenou nejakých symbolov v znaku

   vráti prvok s najmenším počtom zmenených znakov

3. zhoda odstranením opakujúcich sa párov

   vráti hodnotu vykonaných zmien

Pri hľadaní vhodného vzoru za pomoci zmeny v rozpoznanom znaku by sa mohla uplatniť editačná vzdialenosť, ale jej výpočet je zbytočne zložitý a k tomu chceme obmedziť typy prevadzaných zmien.

## Časová zložitosť

Časová zložitosť bude lineárna k veľkosti vstupu. A multiplikačná konštanta nebude veľká, lebo pre každú pozíciu v ťahu nie je veľký počet operácii. A ani prevádzané operácie nie sú moc náročné na čas procesora. Transformačná funkcia má lineárnu zložitosť ku počtu pozícii vo vstupnom ťahu. Priradzovacia funkcia má lineárnu zložitosť ku množine $V$ platných vzorov pre danú plochu. Najnáročnejšia operácia bude hľadanie podobných znakov ku rozpoznanému znaku. Hľadanie smeru má konštantnú zložitosť.

## Učenie vzorov

Znak uložený vo vzore je postupnosť symbolov označujúcich smery. Učiaci režim pri tomto algoritme bude založený na nahraní jedného vzoru, ktorého hodnoty smerov v znaku budú presne porovnávané k rozpoznavaným ťahom. Prípadne može byť dorobený učiaci režim, kde sa budú nazbierané hodnoty priemerovať spôsobom že sa vyberie najpočetnejší výskyt daného znaku v nazbieraných vzoroch.

## Problémy vektorového algoritmu

### Problémy s podobnými ťahmi

Algoritmus môže mať problémy pri pismenách typu G,C,O a 0,6, kde smerový rozdiel týchto znakov je veľmi malý, pretože sa líšia iba vo svojom zakončení. Užívateľ bude musieť presne zadávať znaky tak, aby boli dobre odlíšiteľné a nedochádzalo k veľkému počtu zlých rozlíšení. Možné zlepšenie je malými zmenami ťahov tak, aby bol konečný smerový rozdiel väčší, a to napríklad núteným predĺžením ťahu G o smer nadol. Tým sa zaručí lepšie odlíšenie a menšia chybovosť. Prípadne by mohol byť použitý na rozpoznávanie prídavný jav vzdialenosti začiatku a konca ťahu.

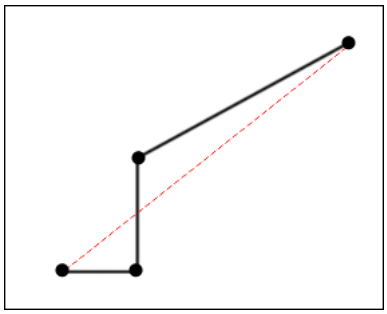
### Problémy s roztrepanim

Ide o problém vstupu, kde sú pozície od seba nepravideľne vzdialené viz obrazok 3.4. To spôsobuje, že aj veľmi blízke pozície vyzerajú pre algoritmus ako zmeny smeru, čo by malo nepriaznivý dopad na schopnosť priradiť ťahu správny stav. Prídavné editácie rozpoznaného znaku môžu zvyšovať počet chybných rozpoznaní. Preto sa snažíme takéto hrany vyhladzovať už v transformačnom algoritme tým,

že priliš blízske body ignorujeme.

**Problémy s jednoduchými ťahmi**

Jednoduché dotyky ako zvislá, horizontálna alebo zošikmená čiara majú problém obmedzeného počtu typov, a to iba na počet smerov, ktorý algoritmus rozpoznáva. To znamená, že ak algoritmus pracuje len so štyrmi smermi, tak sú možné len štyri takéto jednoduché ťahy. Pre ich rozpoznanie by sme sa ku takýmto ťahom museli správať špeciálne, naprklad za pomoci vlastnosti vdialenosti počiatočného a konečného bodu.



Obr. 3.4: Ukážka roztrepania vstupného ťahu

## Zhodnotenie vektorového algoritmu

Algoritmus nebude príliš výpočtovo zložitý. Bude si však vyžadovať dobré testovanie a nastavenie minimálnych medzí pre rozpoznánie a pre určenie zmeny ťahu. Ďalej si pravdepodobne bude vyžadovať dodatočné zmeny v sade znakov, ktoré sa používajú tak, aby ich dokázal lepšie spoznávať. Pravdapodobne by bolo dobré použiť dalšiu vlastnosť, a to vzdialenosť začiatočnej a konecnej pozície, pre zlepšenie rozlišovacej schopnoti v nejednoznačných prípadoch.

## 3.4 Rastrový algoritmus

Rastrový algoritmus je založený na priamom provnávaní tvarov tahu, a to za pomoci rastrovej mapy a percentuálného prekrytia vzorov.

Algoritmus obsahuje obe funkcie potrebné pre rozpoznávanie tak, ako sme si definovali vyššie transformačnú funkciu a priradzovaciu funkciu

Oproti rozboru uvedeného pri vlastnosti tvaru vstupného ťahu, bude mať tento algoritmus jednu zmenu, a to, že zo vstupného ťahu sa bude vytvárať bitová mapa, a tá bude porovnávaná so znakmi testovaných vzorov tým, že tieto vzory budú fiktívne vykreslované na bitovú mapu aktuálneho ťahu. Takto bude znížená pamätová náročnost algoritmu, pretože bude potrebovať iba jednu bitovú mapu pevných rozmerov.

**Transformačná funkcia**

Transformačná funkcia bude mať za úlohu zobrať príjmaný ťah a ten správne pretrasformovať do jednotného formátu. Pod jednotným formátom rozumieme používanie súradnicového systému s obmedzeným maximom rovnakým pre vzorové znaky ako aj pre príjmaný znak. Táto transformácia by mala strácať, co najemnej informácii o ťahu, ale pritom sa musí snažit o urcitú generalizáciu. Preto je možná určitá deformácia ťahu - natiahnutie, sploštenie a podobne.

Spoločný súradnicový systém budú vlastne súradnice bodu v bitovej mape, ktorá ma pevne dané rozlíšenie. Od tohto rozlišenia bude závisieť nielen rozlišovacia schopnosť, ale aj pamäťová a časová náročnosť.

**Priradzovacia funkcia**

Ďalšia časť algoritmu je priradzovacia funkcia, ktorá ma dve časti: samotné porovnávamie vzorov a následné vyhodnocovanie najúspešnejšieho vzoru. Prvá casť algoritmu zoberie transformovaný ťah a ten vykreslí do predpripravenej bitovej mapy, ktorá bude mať pevné rozmery udané spoločným súradnicovým systémom. Táto mapa sa použie pri testovaní vzorov a tie budú vyhodnocované na úspešnosť. Najzhodnejší vzor, ktorý bude spĺňať dané percentuálne minimum, bude vyhlásený za rozpoznaný vzor pre tento ťah.

Zhodnosť definujeme ako pomer medzi bodmi v mape, ktoré boli pri fiktívnom vykresľovaní označené ako tvar ťahu a celkovému počtu vykreslených bodov.

počet prekrývajúcich / celkový počet = zhodnosť

Tu priblížime určité detaily práce rástrového algoritmu.

Transofmačná funkcia pracuje v lineárnom čase a má malé pamäťové nároky. Pri jej vytváraní je najdôležitejšia správna voľba presunu ťahu do spoločnej súradnicovej sústavy. Na samotnú transformáciu slúžia transformačné pomery, ktoré sa prepočítavajú na základe opísu ťahu a rozmerov výslednej sústavy.

**Postup práce tranformačnej funkcie:**

1. inicializácia - načítanie ťahu a zistenie jeho rozmerov
2. určnie transformačných pomerov
3. prejdenie všetkých pozícií v ťahu a transformovanie ich súradníc do novej súradnicovej sústavy

Určenie transforamačných pomerov má dve časti. V prvej vytvárame ťahu jeho opis. Pod opisom rozumieme obdlžnik, ktorému je ťah vpísaný. Určenie polohy a stanovenie rozmeru tohto obdlžnika bude vecou testovania. Pravdepodobne sa použie priamo opísaný obdĺžnik ťahu, ktorý sa určí na základe najväčších a najmenších hodnôt rozmerov v horizontalnom a verikalnom smere. V druhej časti urcovania pomerov zoberieme rozmer opisu ťahu a rozmer spolocnej sústavy a vytvoríme pomery pre zmenu súradníc v oboch smeroch. Táto transformácia súradníc može, ale nemusí zachovat pomer strán. Pri tejto zmene súradníc sa najprv uplatní posunutie ťahu z jeho súradnic do súradnic (0,0). Pre lepšiu predstavu máme obrázok 3.5

Po tejto transformácii súradnic je ťah pripravený na vykreslenie do bitovej mapy, ktoré prevadza príslušná priradzovacia funkcia.

**Postup práce priradzovacej funkcie:**

1. inicializácia - vykreslenie transformovaného ťahu do bitovej mapy
2. postupné vyhodnotenie zhodnosti pre každý testovaný vzor
3. vybratie najlepšieho vzoru s dostatocnou zhodnosťou

Samotné vykreslenie ťahu do bitovej mapy by malo mať určitú mieru tolerantnosti. To môže byt dosiehnuté vykreslením čiar s určitou hrúbkou. Prípadne môžeme pre zlepšenie presnosti vykreslovať ťah v rôznych hrúbkach a s rôznym ohodnotením bodov, ktoré znamenajú prekrytie so vzorom. Takýmto spôsobom by mohol byt algoritmus spresnený.

Obr. 3.5: Ukážka transformácie suradníc

Vyhodnotenie zhodnosti znamená vykonat fiktívne vykreslenie do bitovej mapy, načo bude potrebné mať implementovaný vlastný, dostatočne rýchly algoritmus na kreslenie čiar. Tento algoritmus namiesto zakreslenia zistí aktuálnu hodnotu v mape a pokiaľ bude označovať vzor, zvýši sa počet spoločných bodov. Inak sa zvýši iba celkový počet bodov vo vykreslenom znaku.

Vybratie najlepšieho vzoru sa bude riadit jednoduchým výberom vzoru s najvyššou zhodou. Ak tento vzor bude spĺňať minimálnu hranicu zhodnosti, bude prehlásený za rozpoznaný vzor ku vstupnému ťahu.

## Učenie rastrového algoritmu

Algoritmus by mal mať uložené znaky vo vzoroch ako už transformované súradnice do spoločnej sústavy. Preto by na získavanie znaku pre vzor potreboval iba jeden ťah, ktorý by slúžil ako predloha pre transformáciu.

## Problémy rastrového algoritmu

### Určenie transformačnej funkcie

Pri tejto funkcii budú najväčšie problémy s určením správneho opisu a určením transformačných pomerov, pretože samotný opis a vkladanie do novej súradnicovej sústavy môže deformovat vstupný ťah. Dalším problemom je správne napasovanie bitovej mapy zo vstupného ťahu a všetkých znakov v testovaných vzoroch.

**Vykreslovanie ťahu do bitovej mapy**

Tu môže nastať problém pri implementácii pre zložitosť použitia štandartných funkcíi na vykreslovanie. Preto bude možno nutné, kvôli vykresleniu hrubých čiar tento algoritmus implementovať.

**Nerozpoznávanie smeru ťahu**

Algoritmus vôbec nebere do úvahý v akom smere bol vstupný ťah kreslený. Preto nebude môct rozlíšit tahy líšiace sa iba začiatočným a konečným bodom a smerom ich kreslenia. Preto by bolo vhodné dorobiť podporu aj na túto vlastnosť, čím by sa zväčšila doména možných rozpoznavaných znakov.

**Problém histórie**

Algoritmus nedokáže rozpoznať ťahy, ktoré majú po vykreslení rovnaký obrys. Pod takýmito ťahmi si môžeme predstaviť ťahy, ktoré sa vracajú po svojej stope.

# Zhodnotenie rástrového algoritmu

Takto navrhuntý algoritmus by mal byt schopný identifikovať väčšinu rozpoznávaných ťahov, až na hore uvedený problém s ťahmi rovnakého tvaru, ktoré majú iba iný smer. Dôležité bude dobré nastavenie konštánt, ktoré bude potrebovať dostatočné testovanie. Časová zložitosť algoritmu bude väčšia kvôli použitiu a vykreslovaniu do bitovej mapy, pre ktorú zatial nemáme určenú jej veľkosť. Pamätové nároky nebudú velké aj vdaka zmeny ktorý znak je vykreslený a ktorý je porovnávaný. Problém može nastat pri implementácii s dostatocne rýchlym vykreslvovaním a s fiktívnym vykreslovaním určeným na testovanie zhodnosti.

# Kapitola 4

# Implementácia

V tejto kapitole opisujeme ako sa nám podarilo naimplementovať vektorový algoritmus, z dôvodu že rastrový algoritmus nebol dokočený.

## Hlavné súčasti

Samotná implementácia sa v niekoľkých bodoch líši od špecifikácie, a to hlavne kvôli problémom s rozhraním android a nepotrebnosti nektorých informácií v určitých častiach spracovania ťahu. Z tohto dôvodu sme neimplementovali triedu dotykový objekt. Ostatné súčasti sú implementované podľa návrhu až na malé úpravy.

Zmena v predávaných objektoch medzi súčasťami je v takmer v každom kroku rozpoznávania. Pre vysvetlenie uvedieme celú cestu ťahu súčasťami systému. Ťah začína dotykom užívateľa na dotykovej ploche. Vtedy sa vytvára objekt dotykový pohyb (touch motion).

## 4.1 Užívateľské rozhranie

**Implementácia dotykovej plochy**

Pre implementáciu dotykovej plochy sme použili upravený štandardný prvok prostredia nazývaný pohľad (view). Jeho úpravou sme dosiahli zobrazovanie rôznych plôch ohraničených farebným okrajmi a možnosť zaznamenávať prichádzajúce správy o dotyku. Tieto správy pochádzajú od nadradenej súčasti systému, na ktorú nemáme vplyv, a preto nemôžeme ovplyvniť to, aký vstup dostávame. Správy o dotyku chodia

v nepravidelných rozostupoch a intervaloch. Obsahujú postupné súradnice nachádzajúce sa v tomto dotyku (ťahu). Tieto informácie si zapamätávame a s pomocou nich vytvárame stopu za dotykom. Ďalej posielame informácie o polohe dotyku ďalšej súčasti aplikácie, ktorá ich pripraví na rozpoznanie. Implementácia spodnej nástrojovej lišty je kompletne vytvorená zo štandartných súčasti systému s použitím malých úprav.

**Implementácia menu**

- načítavanie a ukladanie konfigurácie
- ukázanie dostupnych ťahov
- testovací a trenovaci režim dotykov
- prístup k ďalším informáciám

Zobrazenie položiek je riešené jednoduchým výpisom, kde každú položka má svoj vlastný riadkový prvok v menu.

**Načítavanie konfigurácie**

Pri načítavaní konfigurácie zobrazíme užívateľovi dostupné konfigurácie nachádzajúce sa v hlavnom adresári aplikácie (./andgrafiti). Tu zobrazíme a označíme aj aktuálne používanú konfiguráciu. Po tom, čo užívateľ označí novú konfiguráciu, ju spracujeme a vytvoríme nový hlavný konfiguračný objekt, ktorý nanovo inicializuje všetky potrebné podobjekty.

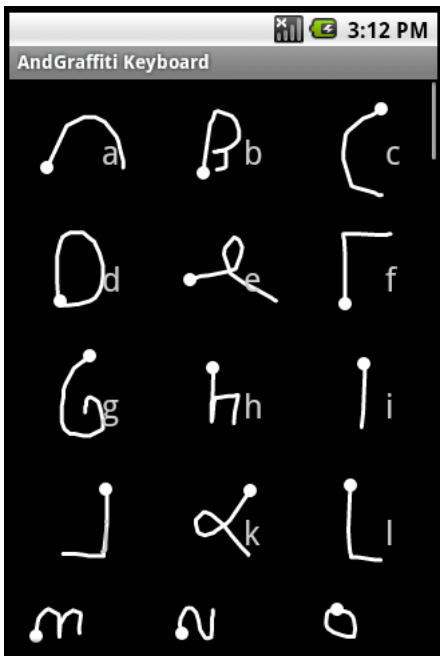
**Ukladanie konfigurácie**

Pri ukladaní konfigurácie zobrazíme používateľovi dialógové okno s editovateľným názvom súboru. Do tohto súboru umiestneného v hlavnom adresári sa konfigurácia uloží, pričom sa použie práve používaný hlavný konfiguračný objekt.

**Ukazovanie dostupných ťahov**

Pre túto funkcionalitu je vyhradené nové podmenu, ktoré najprv ukazuje v stĺpcovom zobrazeni všetky dostupné ťahy (viz 4.1). Presnejšie ukazuje všetky spojenia aj s ich vlsatnostami ako je tvar ťahu, hodnota výstupného reťazca a stav dopredného modifikátora (shift...). Po označeni vybraného spojenia sa zobrazí nová stránka

s možnosťou otestovať si ako musí vyzerať ťah, aby bol rozpoznaný. V tejto časti sa zobrazuje samotné písmeno, ktoré sa ma rozpoznať a grafické znázornenie vzoru ťahu pre jeho rozpoznanie. Po zadaní správne rozpoznaného ťahu sa oblasť s ukazaným písmenom rozsvieti na zeleno, inak je červená.




Obr. 4.1: Zobrazenie spojeni a ich vlasťností

Táto časť má vlastnosť ukladať všetky ťahy, ktoré boli vykonané, a to aj s ich spojeniami. Táto vlastnosť je zapínaná premennou v hlavnej konfigurácii, ktorá je určená pred inštálaciou aplikácie na zariadenie. Ťahy sú ukladané do adresára aplikácie v osobitnom súbore (debugtouches.txt), ktorý neskôr slúži aj na získavanie testovaných ťahov.

**Trénovací režim**

Beží v rovnakom okne ako testovací režim. Rozdiel je v tom, že v okne, kde sa zbrazoval požadovaný tvar ťahu sa teraz zobrazuje posledný ťah a ten je možné uložiť ako nový vzor pre toto spojenie.
**Pomocné funkcie pre vývojárov:** V implementácii menu sú ešte pridané skryté po-

ložky zobrazené iba pri spustených ladiacich premenných v kóde.

- Učenie z ťahov

  Spustí naučenie znakov vo všetkých vzoroch podľa zapamätaného ťahu určeného na vykresľovanie užívateľovi.

- Testovací režim

  Spustí testovaciu úlohu, kde sa načítajú testované ťahy zo súboru. Tie sa otestujú a výsledky sa uložia do nového súboru.

## 4.2 Konfigurácia

V programe sme použili konfiguráciu zapísanú pomocou jazyka XML. Táto konfigurácia spĺňa všetky predpoklady popísané v zadaní v sekcii 2.2. Je tvorená časťami, ktoré sú vyhradené začiatočnými a koncovými značkami. Pre príklad uvádzame krátku ukážku a popisok použitého spôsobu zapísania konfigurácie:

```
<resources>
<settings>
<!--Nastavenia - ulozeny nazov premenej a jej hodnota -->
<setting name="ConfigFileName" value="Default_Config.xml" />
</settings>
<spaces>
<!-- Plocha - ulozeny nazov plochy, jej identifikator,
vyska sirka a pozcia uvedena v percentach a farba
urcena na farebne vyznacenie plochy -->
<space name="Alphabet" id="1" height="100" width="100"
positionx="0" positiony="0" color="#55FF0000" />
</spaces>
<cases>
<!-- Stavy - ulozeny nazov stavu a jeho identifikator -->
<case name="Shift" id="2" />
</cases>
<bindings>
<!-- Spojenia - ulozena plocha, stav, identifikator vzoru,
vstupna znakova hodnota a novy znak -->
<binding space="1" case="1" id="1" string="a" new_case="0" />
</bindings>
<patterns>
<!-- Vzor - identifikacia, znak vzoru a t'ah zobrazujuci
vzoru v ludskej podobe -->
<pattern id="0" pattern="0" bitmap="" />
<pattern id="15" pattern="234124"
bitmap="[117,52, 111,53, 101,55, 97,60, 95,78]" />
</patterns>
</resources>
```

**Špecifikácia spojení a akcií, ktoré sa vykonávajú**

Pre zjednodušenie boli prijaté špeciálne hodnoty v spojeniach, aby sme tým zabezpečili určitú funkcionalitu. Naproti štandartnym pripadom typu

```
<binding space="1" case="1" id="2" string="b" new_case="0" />
```

**Špeciálne prípady**

- `case="0"`

  znamená, že toto spojenie je platné pre všetky stavy. Takto sme dosiahli, že nie je nutné opakovať spojenia pre každy stav osobitne, aj keď tieto spojenia budú zaradené do všetkých stavov.

- `new_case="0"`

  znamená, že aktuálny stav sa nemení, eliminuje to nutnosť pri case="0"zadávať spojenia len kvoli tomu, aby sa neprechádzalo na iny stav, tj. napríklad pre zachovanie CapsLocku. Text v string= sa iba teraz spracúvava ako výstupný text do editovacieho poľa. Preto nie je povolené vytvoriť modifikátor stavu, ktorý pridá aj text do editovaného textu. Tento prípad je akcia zapísania znaku.

- `new_case="-1"`

  špeciálny režim, ktorý spúšťa spätnú modifikáciu znaku v editovanom texte. Pri tomto režime sa výstupný text bere ako trasnformačná tabuľka premien koncového znaku na nový znak v editovanom texte. Akcia zmeny stavu.

- `string="$backspace"`

  špeciálna hodnota výstupného reťazca určená na možnosť priradiť funkciu zmazania posledného znaku v editovanom texte. Akcia mazania posledného znaku.

Týmyto špeciálnymi vlastnostami spojení zmenšujeme ich počet, čím zjednodušujeme ich konfigurovanie, načítavanie aj pamäťovú náročnosť pri používaní. Celkové zmenšenie počtu je aj niekoľkonásobne viac závislé od počtu stavov.

Možnosť definovať viac vzorov pre ten istý vstupný reťazec je umožnená definovaním viacerých spojení s tými istými vlastnosťami až na identifikátor vzoru.

V konfiguraci je tiež možné špecifikovať algoritmus, ktorý ma byť použitý pri vyhodnocovaní. Preto je program pisaný s modularitou na typ rozpoznavajúcej funkcie,

ta má pevne stanovené typy vstupov a vystupov. Pre zohľadnenie potrieb algoritmov na rôzne znaky vo vzoroch, sú znaky implementované vo viacerých variantách.

**Varianty znakov vo vzoroch**

1. číselná - hodnota pripravená pre rozširenie
2. reťazec - používaná pre vektorový algoritmus
3. zoznam súradnic ťahu - použitý pri nedokončenej implementáci rástrového algoritmu

## 4.3 Vektorový algoritmus

Pri implementácii sme sa rozhodli pre vektorový algoritmus, ktorý mal v analýze veľmi dobré vyhliadky a pritom mal nízku časovú a pamätovú náročnosť pre procesor, ktorá je dôležitá pri použití na mobilnom zariadení.

Pri vývoji sme postupovali presne podľa návrhu algoritmu, pričom boli niektoré detaily upresnené. Použitý počet smerov je 4. Pri tomto počte je najnižšia možnosť zlých identifikácií smeru, a tým sa zvyšuje celková úspešnosť algoritmu. Algoritmus využíva variantu znakov uložených ako reťazec, kde každá pozícia v reťazci znamená nový smer v reprezentovanom ťahu.

Systém rozpoznáva znaky okamžite po skončení zadávania, co znamená hneď, ako dostane od operačného systému správu o jeho ukončení.

Rôzne varianty rozpoznávacích algoritmov sú implementované pomocou dedenia od základného algoritmu. Týmto spôsobom je zaručená dostatočná kontrola nad priebehom vyhodnocovania a pritom je dodržané rozhranie s ostatkom systému. Preto doprogramovanie nového algoritmu je obmedzené iba na zdedenia od hlavnej triedy (searcher) a úpravu jej funkcií.

# Problémy implemetácie

Pri implementácí sme narazili na pár problémov, ktoré nie sú priamo riešiteľné.

**Ukončovanie ťahou**

Tu ide o problém rozhrania našej aplikácie a operacného systému, pri ktorom ocakávame, že ak sa dokončí zadávanie ťahu na dotykovej ploche, budeme o tom informovaní štandartným spôsobom - pomocou objektu dotyku, ktorý má v sebe nastavený príznak o ukoncení ťahu.

Problém nastáva v tom, že systém nie vždy takýto dotykový objekt našej aplikácii pošle. Preto nie sme schopní správne zacat rozpoznávanie po ukoncení zadávania a takýto ťah aplikácia vôbec nespracuje. Riešenie v používanom rozhraní neexistuje, čo spôsobuje, že nie sme schopní takémuto neželanému správaniu zabrániť.

Možné riešenia vzniknutej situácie sú aspoň dve, ale ani jedno sa príliš nehodí do kontextu našej aplikácie. Jednou možnostou je neustále rozpoznávat aj nedokoncený ťah, co by zvýšilo zataženie a musel by sa výrazne prispôsobiť celý spôsob spracovania aj rozpoznávania algoritmu. Preto tento spôsob nápravy nie je vhodný.

Druhá možnost nápravy by bolo definovať pevný čas, ktorý by slúžil ako limit na príjem dalšieho dotykového objektu. To by napomohlo prijímat všetky zadávané ťahy, ale za to by spôsobilo možné skracovanie ťahov a následné nerozpoznanie, pokial by tento limit nestačil užívateľovi. Ďalej pri tomto prístupe sa zavádza určitá forma aktívneho čakania, čo práve pri mobilnej platforme je veľmi dobre.

**Nepresnosť sledovania dotyku**

Ďalší problém, ktorý bol počas vývoja zistený je, že pri posielaní správ o dotyku je vysoká nepresnosť vznikajúceho ťahu. Je to spôsobené prílišnými odstupmi medzi jednotlivými pozíciami dotyku, ktoré sú pri rýchlejších pohyboch až 30% veľkosti celej zadávanej plochy. V tomto prípade dochádza k vynechávaniu dôležitých častí dotyku, a preto je získaný ťah velmi nepresný a vedie k problémom v rozpoznávaní. V rozhraní operačného systému nie je možné definovať žiadnu hodnotu na zvýšenie frekvencie správ o dotyku, a preto nie je možná priama náprava tohto problému. Samotný problém je v tom, že platforma nie je stavaná na typ aplikácie podobný našej aplikácii. Je to spôsobené tým, že pocíta skôr s jednoduchými gestami na ovládanie a nie zložitejšími ťahmi, ktoré chceme implementovať. Pri snahe zistiť viac o možnom riešení tohto problému bolo zistené, že interval zasielania správ o dotyku je zavislý od aktuálnej zátaže zariadenia, čo znamená, že čím je zariadenie vytaženejšie, tým sú intervaly dlhšie. Výsledkokm tohto nedostatku je, že iba pomalšie zadávané dotyky sú dostatočne presne spracované na ťahy, a preto majú väcšiu pravdepodobnosť na správne rozpoznanie.

# Kapitola 5

# Testovanie

V tejto časti uvedieme postup testovania nášho algoritmu a zadefinujeme požiadavky na neho kladené. Uvedieme aj výsledky tohto testovania. Pri testovaní prebiehalo aj ladenie rôznych konštánt pre rozpoznávanie - ako počet smerov, minimálne hranice a podobne.

## Testovacie dáta

Pre testovanie boli vytvorené dve nezávislé množiny, a to množina ťahov spracovaných na vzory a množina ťahov, ktoré sú testované na správne rozpoznanie. Obidve množiny boli nazbierané pri reálnom zadávaní ťahov v prostredí aplikácie. K týmto tahom sú ukladané aj identifikacné čísla (id) vzorov, ktoré ich reprezentujú. Samotné zbieranie ťahov sa odohráva v ukážkovom režime učiacej časti užívatelského menu.

### Množina získaných ťahov

Celkovo sme nazbierali 462 ťahov tak, aby sme pre každý vzor mali 11 ťahov. Jeden z týchto ťahov sme ručne vybrali ako predlohu pre znak vzoru. Ostatných 10 používame na testovanie daného vzoru.

### Testovacie množina

Testovacia množina obsahuje desať ťahov pre každé spojenie, ktoré je definované v konfigurácii. Tieto ťahy sú nezávislé od ťahov určených na spracovanie na vzory. Tieto ťahy tiež neboli upravované a preto simulujú rovnaký vstup ako vstup z dotykovej plochy. Táto množina ťahov je uložená v hlavnom adresári aplikácie.

# Postup testovania

Testovanie ťahov prebieha priamo v aplikácii, pri spustenom móde v hlavnom konfiguracnom súbore.(TEST_TOUCHES). Po jeho spustení je prístupná položka menu na testovanie.

**Postup testovania:**

inicializácia

naučenie ťahov postupným pretvorením vstupných ťahou na znaky vo vzoroch

1. načítanie ťahu zo vstupného súboru

   pri načítaní sa prečíta aj indentifikacné číslo pre spojenie, ktore ťah reprezentuje

2. testovanie ťahu

   testovanie prebieha pomocou odchytového objektu, ktorý simuluje vstup pre rozpoznávací alogoritmus tak, aby bol vstup rovnaký, akoby prichádzal dotykom na plochu. Jedna zmena - ťah je postupne testovaný na všetky definované plochy.

3. vyhodnotenie výsledkov

   pri vyhodnocovaní výsledkov vytvárame aj štatistiky

**Vyhodnocovanie výsledkov**

Pri vyhodnocovaní overujeme, či algoritmus našiel správne id spojenia, ku ktorému ťah patrí. Toto vyhodnocovanie prebieha nad všetkými nájdenými id pre daný ťah, ktorý pochádzajú z rôznych testovaných plôch.

Testovanie sa riadi podľa vyššie definovanej funckie úspešnosti algoritmu a definujeme ju pre vzory x,y. $A \in T, V \in M, \exists x = f(A,V), g(A,V) = y$

- Ak našiel aspoň v jednej ploche správne id, tak algoritmus ťah rozpoznal správne.
- Ak nenašiel ani v jednej ploche správne id, tak algoritmus rozpoznal nesprávne.
- Ak ani v jednej ploche nerozpoznal id, tak algoritmus ťah nerozpoznal.

Pri testovaní sa získava štatistika nielen pre celkový počet testovaných ťahov, ale aj pre ťahy s rovnakým id spojenia. Táto štatistika nesie údaje o počte testovaných ťahov, počte správne rozpoznaných, počte nesprávne rozpoznaných a počte nerozpoznaných.

**Pri testovaní boli skúmané tieto hodnoty**

- celkový počet správnych rozpoznaní
- celkový počet nesprávnych rozpoznaní
- celkovy počet nerozpoznaných ťahovu
- počet správnych rozpoznaní pre každý ťah
- počet nesprávnych rozpoznaní pre každý ťah
- počet nerozpoznaných ťahou pre každy ťah

# Vyhodnotenie

Vyhodnotenie testovania prebieha na vektorovom algoritme. Počas testovania sme dospeli k záveru, že definovanie iba štyroch rozlišovacích smerov je optimálna hodnota pre nízku úroveň mylne identifikovaných smerov. Ďalším testovaním nad získanou množinou ťahov sme sa snažili nastaviť parametre rozpoznávania na čo najlepšiu hodnotu. Samotná optimalizácia parametrov bola prevádzaná metodicky ručnou úpravou, preto dané hodnoty nemusia byť optimálne a môžu byt výhodné iba pre určitú skupinu ťahov.

Na zlepšenie celkových výsledkov algoritmu by sme potrebovali väčšiu množinu testovaných tahov a hlavne ich mať od rôznych osôb pre elimináciu opakovania chýb. Najväčším limitom testovania je čas potrebný na otestovanie množiny ťahov. Pri našej množine 420 ťahov je čas cez jednu minútu. To veľmi predlžuje dobu ladenia algoritmu. Preto zväčšenie množiny by bolo možné iba ak by sa testovanie algoritmu odohrávalo mimo samotné zariadenie alebo priamo na desktopovom počítači.

Vo výsledku máme uvedených niekoľko konfigurícii najvýznamnejších atribútov, ktoré sa nastavujú pre náš vektorový rozpoznávací algoritmus.

**Tieto atributy su:**

- minimálna vzdialenosť dvoch pozícii

- počet prístupných rozdielov na stále platnú identifikáciu

Výsledky testovania sú uvedené v tabuľke 5.1. Z týchto výsledkov je jasné, že ak optimalizujeme parametre na minimálny počet zle rozpoznaných dotykou, tak nastavíme atribúty na prvú variatnu. Zato ak chceme naväčší počet dobre rozpoznaných znakov bez ohľadu na počet zle rozpoznaných, tak našou voľbou by bola šiesta varianta. V konečnej úprave aplikácie používame nastavenie z prvého riadku, takže minimálna vzdialenosť je tri a nulový počet možných zmien pre porovnanie znakov.

Prevdepodbne výrazne zlepšenie rozpoznávania by priniesla dobre vyladená konfigurácia, ktorá by mala viac spojení pre každú akciu tak, aby sa pokryli všetky unikátne a časte prípady rozpoznania ťahov. Tým by bola dosiahnutá veľká pravdepodobnosť správneho rozpoznania. Na prevedenie takejto úpravy konfigurácie by sme potrebovali viac času a presunúť testovanie algoritmu mimo cieľové zariadenie.

| Verzia | Minimum px. | Rozdielov | Testovaných | Správne | % | Nesprávne | % | Nerozpoznaných | % |
|---|---|---|---|---|---|---|---|---|---|
| I | 3 | 0 | 420 | 304 | 72 | 26 | 6 | 90 | 21 |
| II | 7 | 0 | 420 | 303 | 72 | 33 | 8 | 84 | 20 |
| III | 10 | 0 | 420 | 281 | 67 | 38 | 9 | 101 | 24 |
| IV | 3 | 1 | 420 | 321 | 76 | 55 | 13 | 44 | 10 |
| V | 3 | 2 | 420 | 325 | 77 | 81 | 19 | 14 | 3 |
| VI | 3 | 3 | 420 | 327 | 78 | 87 | 21 | 6 | 1 |
| VII | 7 | 1 | 420 | 315 | 75 | 67 | 16 | 38 | 9 |
| VIII | 7 | 2 | 420 | 317 | 75 | 96 | 23 | 7 | 2 |

Obr. 5.1: Tabuľka výsledkov testovania

# Kapitola 6

# Záver

V tejto práci sme s úspechom až na hore uvedené problémy navrhli a implementovali dotykovú klávesnicu s použitím jednoťahových gest pre zadávanie znakov. Pri uvedených problémoch sa uvádzajú ich navrhované riešenia, ktoré by trebalo otestovať a vybrať z nich tie najúčinnejšie. Samotný algoritmus rozpoznávania až na jeho malé nedostatky v možnostiach rozpoznávania má veľmi vyváženú schopnosť rozpoznať ťah pri nízkych výpočtových nárokoch. Podarilo sa nám vytvoriť aj prehľadný a použiteľný systém na rozmanitú konfiguráciu schopnú zvládať aj viacznakové varianty ťahov. V sumáre bola práca úspešná.

# Literatúra

[1] Android Development Guide 2010
*http://developer.android.com/index.html*

[2] Eclipse Fundation. 2010
*http://www.eclipse.org/org/*

.