ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE, FAKULTA ELEKTROTECHNICKÁ, KATEDRA KYBERNETIKY

DIPLOMOVÁ PRÁCE

Hodnocení EEG signálu v reálném čase

04. 01. 2011

RADEK NOVÁK

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Radek N o v á k

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Kybernetika a měření, blok KM2 – Umělá inteligence

Název tématu: Hodnocení EEG signálu v reálném čase

Pokyny pro vypracování:

Tématem diplomové práce je návrh a realizace nástroje pro zpracování EEG signálu v reálném čase. Tento přístup je požadován ve všech aplikacích, kde potřebujeme znát obsah užitečné informace obsažené v EEG signálu již při samotném měření (EEG biofeedback, spánková apnoe, monitorování novorozenců apod.).

Student se zaměří především na následujici části:

1. rozbor problematiky on-line vyhodnocování EEG,
2. datová analýza, možnost načítaní dat přes TCP/IP protokol,
3. výběr a implementace následujících metod: segmentace signálu, výpočet příznaků, klasifikace do tříd,
4. přehledná visualizace vypočtených příznaků a výsledné klasifikace,
5. časová optimalizace sestavených algoritmů,
6. testování na reálných datech (spánkové EEG, novorozenecké EEG),
7. dokumentace.

Poznámka k implementaci:
- Java / C++ ,
- možnost hodnotit dlouhodobé záznamy,
- modularita.

Většina podobných přístupů je v současnosti schopná pracovat pouze v off-line režimu, což přináší řadu omezení. Důležitým úkolem této práce bude navrhnout takové řešení, aby bylo dosaženo maximální efektivnosti použitých metod při jejich minimální výpočetní náročnosti.

Seznam odborné literatury:

[1] Malmivuo, J.; Plonsey, R.: Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields. New York: Oxford University Press. 1995, 512 p., ISBN 0-19-505823-2.
[2] Goldensohn, E. S., Legatt, A. D.; Koszer, S.; Wolf, S. M. : Goldensohn's EEG interpretation: Problems of Overreading and Underreading. 2nd Completely Revised Edition, Armong, 1999.
[3] Liu, H.; Motoda, H.: Feature Extraction, Construction and Selection: a Data Mining Perspective. In National University of Singapore, Singapore, 1998, ISBN 0-7923-8196-3.

Vedoucí diplomové práce: Ing. Václav Gerla

Platnost zadání: do konce letního semestru 2009/2010

L.S.

prof. Ing. Vladimír Mařík, CSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 23. 1. 2009

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 4.1.2011 .....

.....

Poděkování

Děkuji Ing. Václavu Gerlovi za jeho věcné připomínky a obětavost při řešení funkčního návrhu aplikace. Dále bych chtěl poděkovat Fakultní nemocnici Na Bulovce v Praze za poskytnutí spánkových dat a Ústavu pro péči o matku a dítě za poskytnutí novorozeneckých záznamů. Poděkování patří také grantovým organizacím, díky kterým vznikl projekt č. 1ET101210512 "Inteligentní metody pro vyhodnocování dlouhodobých EEG záznamů" (projekt byl ukončen v roce 2009) a projekt č. MSM6840770012 "Transdisciplinární výzkum v oblasti biomedicínského inženýrství II", oba projekty stály za vznikem této diplomové práce.

Anotace

Práce se zabývá návrhem aplikace, která je schopna v reálném čase přijímat a analyzovat EEG signály a monitorovat stav pacienta po celou dobu jeho spánku. Výsledkem této práce je vlastní implementace aplikace v programovacím jazyce JAVA, která je schopna z přijatých dat zobrazit spektrogram, rozdělit signály na segmenty pomocí adaptivní segmentace, analyzovat tyto segmenty a metodou nejbližšího souseda je zařadit do klasifikační třídy. Cílem aplikace není vytvořit plnohodnotnou klasifikaci, ale předložit lékaři podrobnější informaci o aktuálním stavu pacienta. Při návrhu řešení se snažíme najít vhodný kompromis mezi rychlostí algoritmu a jejím výsledkem z hlediska přesnosti metody. Součástí návrhu aplikace bylo také vytvoření trénovací množiny pro klasifikaci EEG záznamů. Aplikace s vytvořenou trénovací množinou byla odzkoušena na reálném EEG záznamu. Výsledky práce včetně zdrojových kódů v jazyce Java jsou přiloženy na CD.

Abstract

This work describes the design of an application that will be able to receive and analyze the EEG signals in real-time and monitor the patient throughout his sleep. The result of this work is own implementation of application in Java programming language, which from received data is able to display spectrogram, divide signal into segments by adaptive segmentation, analyze these segments and by the nearest neighbor method classify them into the classification classes. The function is not to create a full classification, but provide to the doctors detailed information about the current state of the patient. When designing a solution we tried to find a suitable compromise between the speed of the algorithm and its results in terms of accuracy of the method. Part of the application design was also to create a training set for classification of EEG recordings. Application with the training set was tested on real EEG recording. Results of this work with source codes in Java are included on the CD.

Obsah

1 Úvod

Současný způsob měření a analýzy EEG záznamů vyžaduje nejprve měření EEG a po skončení měření jeho následnou analýzu. Naším úkolem bylo tyto dva postupy spojit a poskytnout lékaři podrobnější informaci o stavu pacienta.

Cílem této práce nebylo navrhnout nový nebo zlepšit stávající algoritmy pro analýzu EEG, ale využít již známé metody zpracování EEG signálu a upravit je tak, aby byly schopny zpracovávat data v reálném čase. Výsledkem implementace je pak aplikace, která je schopna v reálném čase poskytovat ošetřujícímu lékaři informace o aktuálním stavu pacienta.

Hlavním výsledkem této práce je aplikace napsaná v jazyce JAVA, která je schopna v reálném čase zpracovávat data až z 24 kanálů. Aplikace je navržena modulárně tak, aby byla schopna komunikovat jak se záznamovou stanicí pomocí TCP/IP protokolu, tak i načítat data přímo z již uložených záznamů z D-file souboru. Ve druhé kapitole jsou popsány základní pojmy a metody pro snímání EEG signálu. Další kapitola se zabývá zpracováním EEG signálu, především je zde popsána frekvenční analýza, segmentace a klasifikace signálu. Následující čtvrtá kapitola se věnuje programovacímu jazyku JAVA, který byl zvolen pro implementaci aplikace především kvůli jeho multiplatformnímu použití. V páté kapitole jsou pak stručně popsány vstupní zdroje dat a samotnou implementaci programu popisuje šestá kapitola, v sedmé jsou pak shrnuty výsledky této práce.

V průběhu implementace programu jsem poměrně často narážel na performační problémy, především na nedostatečnou rychlost použitých algoritmů a nedostatek paměti. Z tohoto důvodu jsem značnou část času při vývoji aplikace věnoval optimalizaci použitých algoritmů a detailně sledoval výkon jednotlivých částí aplikace. Pro ladění paměťových nároků a sledování výkonnosti aplikace byl použit program JVisualVM [37].

2 Snímání a popis EEG signálu

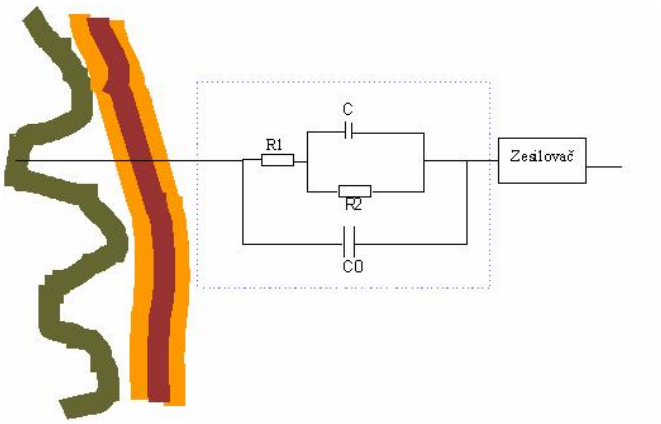
2.1 EEG (ElektroEncefaloGrafie)

2.1.1 Elektroencefalograf

Elektroencefalograf je přístroj skládající se ze snímajících elektrod a procesoru. Pracuje na principu snímání elektrické aktivity mozku pomocí elektrod připevněných na povrchu hlavy. Elektroencefalograf získané signály zesílí a pak je vypíše na papír nebo zobrazí na obrazovce. Vzniklé EEG-křivky mají charakteristický průběh a typickou frekvenci vln. Jinou křivku zobrazí přístroj ve spánku, odlišnou při denní aktivitě. [27]

2.1.2 Technika snímání EEG

Snímání EEG signálu je zajištěno elektrodami. Elektrody jsou důležitým elementem při převodu bioelektrických potenciálů, protože mohou signál přiváděný na vstup zesilovače do značné míry zkreslit. [27]



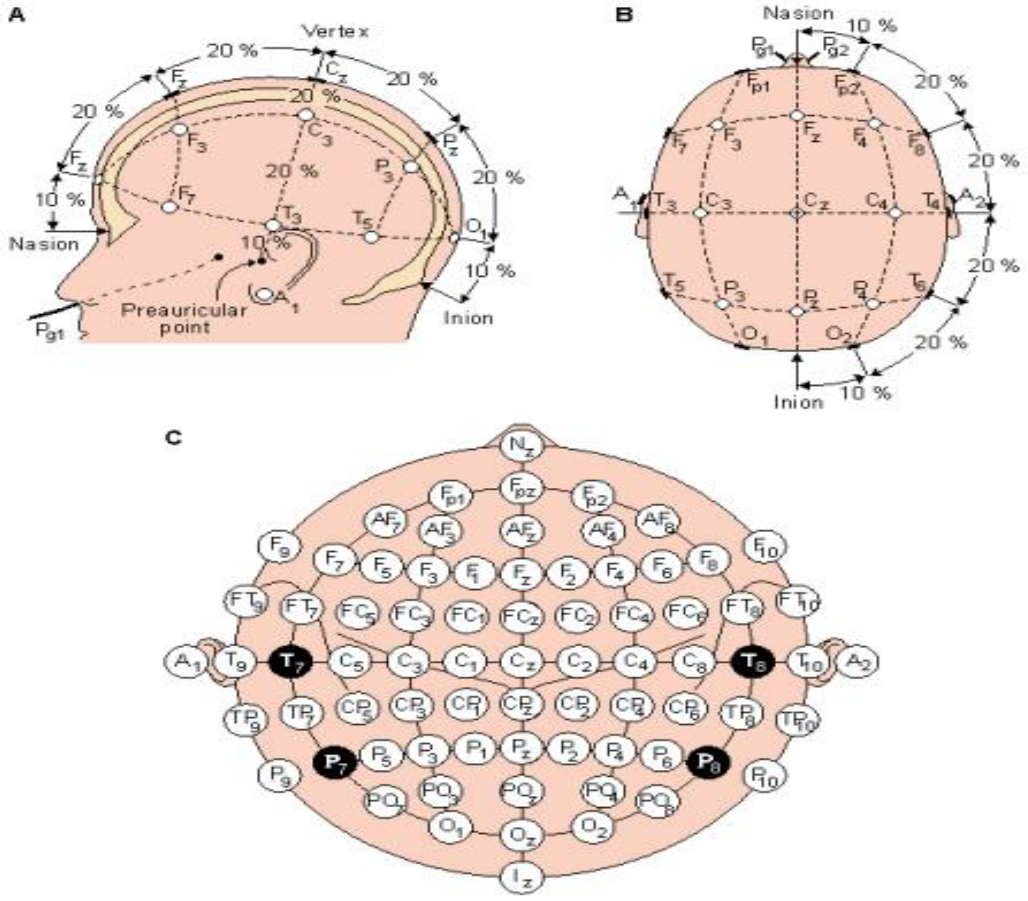
Obrázek 2.1 Impedanční model elektrody [27]

2.1.2.1 Zapojení elektrod

Rozložení elektrod na hlavě není náhodné, řídí se podle jednoduchého antropometrického měření, které navrhl H. Jasper. Jde o tzv. systém **10-20**, ve kterém je vzdálenost od dvou krajních poloh jak v sagitální (podélné) tak v transversální (příčné) linii rozdělena na úseky po 10% nebo 20%. Vychází ze 4 bodů: nasion (kořen nosu), inion (výstupek na týlní kosti), body před ušním boltcem vpravo a vlevo. Lichá čísla elektrod jsou v levé, sudá v pravé polokouli. Indexem Z (zero) jsou označena místa elektrod ve středu.

Označení elektrod v systému 10-20 se skládá z písmene udávající přibližnou polohu elektrody (F – frontal, P – parietal, T – temporal, O – occipital, C – central) a čísla nebo písmene určující přesnější umístění elektrody na hlavě (např. pro senzomotorickou oblast mají elektrody označení C3, C4).

Kombinací ze svodového systému „10-20“ lze definovat velmi mnoho. Např. u ambulantního záznamu se velmi často používá 24kanálový záznam (23 EEG signálů a 1 kanál je určen pro EKG). Více informací o zapojení elektrod lze nalézt např. v [4].

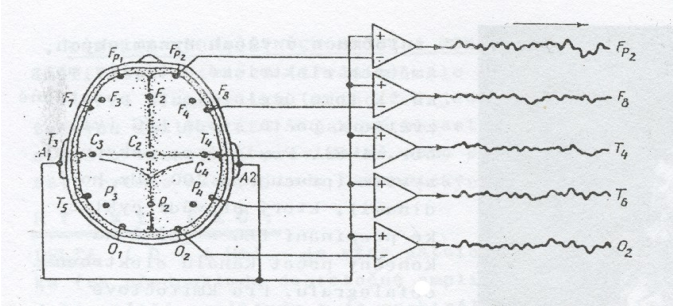


Obrázek 2.2 A) a B) Rozložení EEG elektrod v systému 10-20 C) Vícekanálový systém [27]

2.1.2.2 Režimy snímání EEG

Unipolární režim

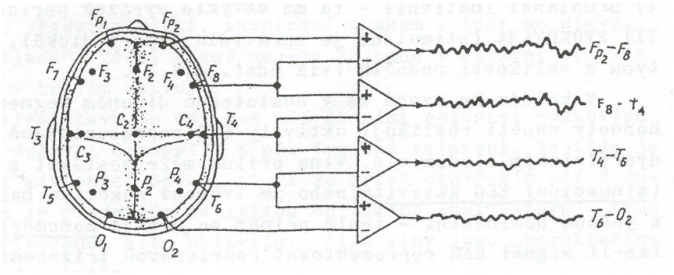
Používá společnou referenční elektrodu (nejčastěji je tato elektroda na ušním lalůčku, někdy se přitom spojují oba ušní lalůčky). Tento systém umožňuje lépe pozorovat velikost a tvar vln, získáme vyšší úroveň EEG, ale lokalizační výsledky mají větší chyby než u velmi často používaného bipolárního režimu. [27]



Obrázek 2.3 Unipolární režim měření EEG [27]

Bipolární režim

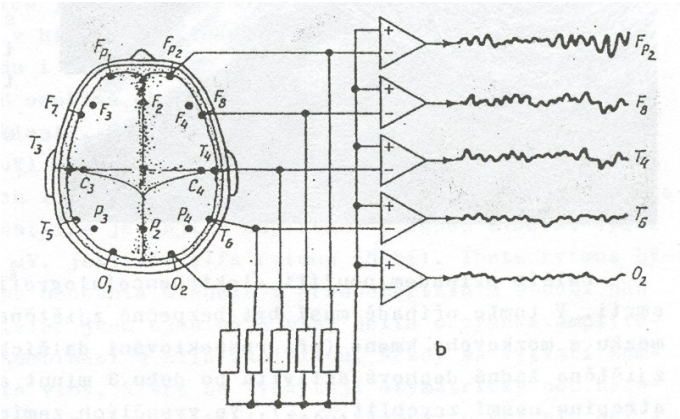
Používá se množina svodů zapojených bez společné referenční elektrody. Označíme-li jednotlivé OZ na obrázku čísly shora dolů, pak invertující vstup k-tého OZ je vždy spojen s neinvertujícím vstupem (k+1)-ního OZ. [27]



Obrázek 2.4 Bipolární režim měření EEG [27]

Zprůměrovaný režim

V tomto systému se používá společná indiferentní elektroda, obvykle vytvořená podle Goldmana. [27]



Obrázek 2.5 Zprůměrovaný režim měření EEG [27]

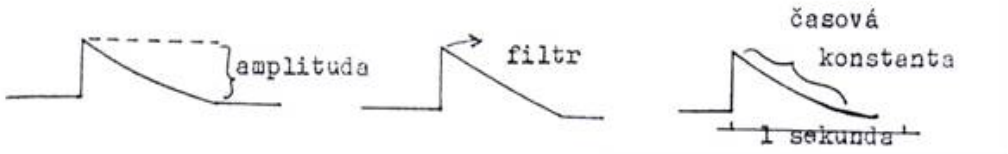
Věncový bipolární režim – systém používá elektrody umístěné v příčných řadách.

Uzavřený bipolární režim – v tomto systému páry elektrod vytváří uzavřený kruh.

Referenční bipolární režim – používá systém referenčních svodů.

Trojúhelníkový režim – představuje speciální variantu vždy tří párů bipolárních svodů.

Před registrací vlastního EEG záznamu musíme znát fyzikální parametry zesilovačů, které prověřujeme před "natočením" vlastní EEG křivky pacienta. Jde o tzv. kalibraci. Na registračním papíře pak můžeme sledovat tyto křivky:



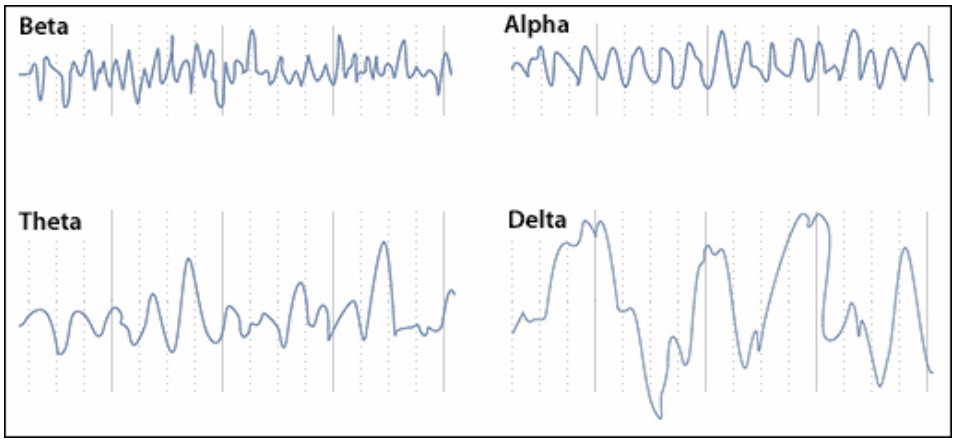
Obrázek 2.6 Křivky zesilovačů [27]

které označují zesílení (amplitudu artefaktu), filtr zesilovače ("ostrost" špičky artefaktu) a konečně časovou osovou konstantu (rychlost klesání z maxima na nulu).

Zesílení i velikost výchylky lze měnit. Běžný filtr je 70Hz, tzn. že většina frekvencí rychlejších než 70Hz nepronikne do výsledné křivky. Čím je filtr nižší (30 - 50Hz), tím je "ostrost špičky" více oblá. Časovou konstantu někdy označujeme jako dolní filtr - omezení pomalých frekvencí. Běžný parametr časové konstanty je 0,3s. To znamená, že za uvedený čas klesne vychýlené napětí o dvě třetiny k základní linii.

Chceme-li registrovat pomalé rytmy, např. dechovou aktivitu, dáme co nejmenší filtr (15 nebo 30Hz) a co nejdelší časovou konstantu (0,3 nebo 0,5s). [27]

2.1.3 Frekvenční pásma



Obrázek 2.7 Klinicky významné frekvenční pásma používané v EEG [18]

2.1.3.1 Delta rytmus (0 – 4Hz)

Je vždy patologickým projevem v EEG dospělého bdělého člověka. Vlny delta se vyskytují hlavně v hlubokém spánku (non REM III, IV), ale i v transu a hypnóze. Objevuje se také při bezesném spánku, případně během bezvědomí způsobeného nemocí či úrazem.

2.1.3.2 Theta rytmus (4 – 8Hz)

U zdravých lidí se objevuje v centrální, temporální (spánkové) a parietální (temenní) oblasti. Patologický stav indikuje theta vlny, jestliže je jejich amplituda alespoň dvakrát vyšší než aktivita alfa. Theta a delta aktivita stoupá během psychotestů. Vlny theta se objevují v EEG signálu též v určitých spánkových fázích a při meditaci – hlubokém uvolnění. Theta vlny se často pojí se živými vzpomínkami, fantazií, obraznou představivostí, inspirací a snem – tudíž se stavy, kdy je vědomé myšlení „odpojeno“.

2.1.3.3 Alfa rytmus (8 – 13Hz)

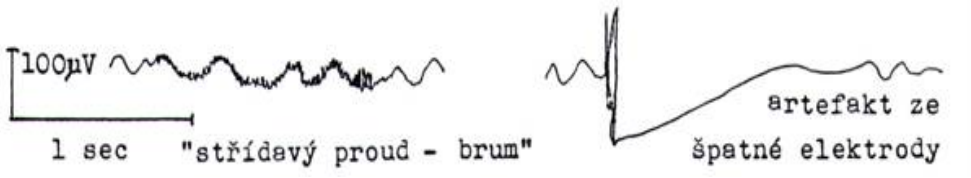
Alfa vlny jsou charakteristické pro stav relaxace a objevují se také těsně před usnutím. Zvýšenou alfa aktivitu můžeme v EEG nalézt při zavřených očích, tlumí se právě otevřením a duševní činností. Alfa rytmus je především aktivitou optického analyzátoru – u lidí, kteří jsou od narození slepí, se alfa aktivita zpravidla neobjevuje.

2.1.3.4 Beta rytmus (13 – 30Hz)

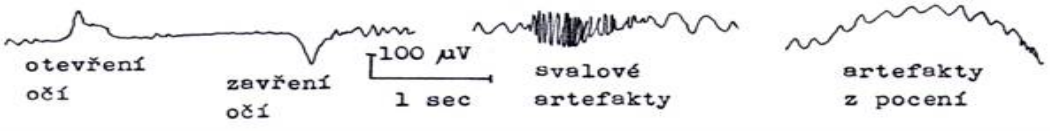
Z hlediska lokalizace je tento rytmus symetrický. Maximum je nejčastěji nad předními částmi lebky, hlavně frontálně. Směrem dozadu ubývá. Beta vlny jsou typické pro soustředění na vnější podněty, pro logicko-analytické myšlení. Pásma vyšší (kolem 30Hz) se pak objevují při podráždění, trémě, úzkosti, nebo vysoce náročných životních situacích.

2.1.4 Artefakty

Elektrický potenciál mozku má na povrchu lebky napětí jen několik desítek mikrovoltů (μV), tudíž elektronické zesilovače v elektroencefalografu musí být velmi výkonné. Velká výkonnost zesilovačů se nám odráží v podobě mnoha nepravých potenciálů, které nazýváme artefakty. Artefakty jsou dvojího druhu: technické a biologické. Typickými technickými artefakty jsou: síťové rušení 50Hz, artefakt ze špatné elektrody (nedostatečně vlhké elektrody, porušené nebo oxidované elektrody, polámané drátky v kabelech elektrod). Příklad několika takových artefaktů je na Obrázek 2.8. Mezi biologické artefakty pak můžeme řadit např. tyto: pohyb očí a očních víček, pocení a svalovou aktivitu (viz. Obrázek 2.9).



Obrázek 2.8 Ukázka technických artefaktů [27]



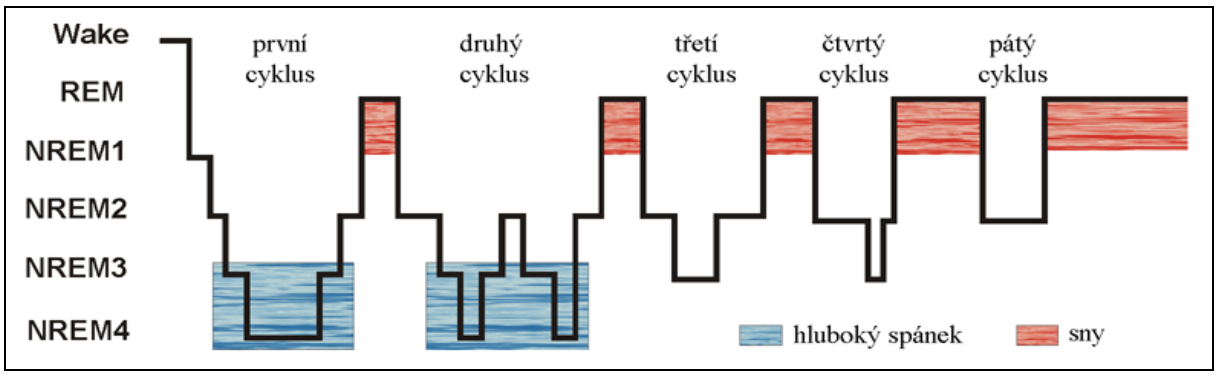
Obrázek 2.9 Ukázka biologických artefaktů [27]

2.2 Spánek

2.2.1 Fáze spánku

Během noci dochází zhruba ke čtyřem až sedmi spánkovým cyklům, každý z nich trvá přibližně devadesát minut. U zdravého člověka se během noci střídá šest spánkových fází:

- Wake (bdělost)
- REM (Rapid Eye Movements)
- NREM1 (usínání)
- NREM2 (lehký spánek)
- NREM3 (hluboký spánek)
- NREM4 (nejhlubší spánek)



Obrázek 2.10 Hypnogram [18]

REM

Tato fáze zabírá 20-25% celkového času spánku a je často označována jako paradoxní spánek, protože je EEG během REM fáze podobné, jako když je člověk vzbuzený. Průběh

EEG je v této fázi vyznačován nízkými napěťovými úrovněmi a přítomností alfa a theta vln. Alfa je zde o 1-2 Hz pomalejší než když je člověk vzbuzený. V této fázi spánku se nám zdá nejvíce snů.

NREM - fáze I

Tato fáze je ve spánku přítomna z 3-8%. V této fázi postupně ubývá alfa aktivita a začínají se objevovat vlny smíšených frekvencí s menší amplitudou. Největší EEG aktivita je na frekvencích v pásmu theta (4–8 Hz). Oči se začínají pohybovat dokola po očnici, dýchání je čím dál hlubší a pomalejší.

NREM - fáze II

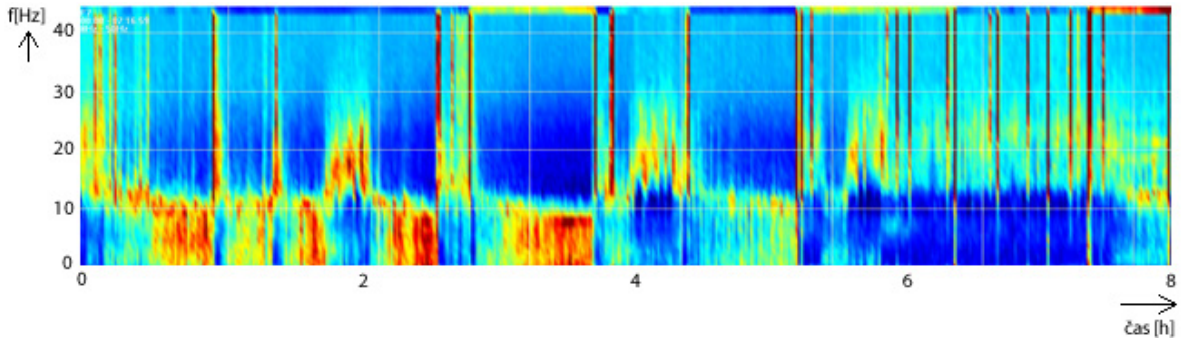
Tato fáze nastává přibližně 10-12 minut po první fázi a je ve spánku zastoupena ze 45-55%. Charakteristickým rysem EEG v této fázi je přítomnost spánkových vřetének, která jsou definována jako krátké úseky vln s frekvencí 12-14Hz a trvají minimálně 0,5s. Jejich amplituda v průběhu celého EEG náhle klesá a narůstá (K-komplex).

NREM - fáze III a IV

Tato fáze je ve spánku zastoupena z 15-20%. Ve třetí fázi se začínají objevovat pomalé vlny o frekvenci 1-2Hz (delta vlny), které mají velikou amplitudu. Třetí fáze trvá přibližně deset minut a čtvrtá pak mezi dvaceti a třiceti minutami. Tyto dvě fáze se někdy u člověka nazývají hluboký NREM spánek, který je ze všech nejdůležitější, neboť v jeho průběhu tělo nabývá nových sil, a je proto nejdůležitější pro celkové posouzení kvality spánku.

2.2.2 Spektrogram

Spektrogram je trojrozměrný graf s časovou osou, frekvenční osou a osou energie spektrálních složek. Lze jej použít například ke sledování spánkových fází během noci. Obrázek 2.11 ilustruje spánkový spektrogram (8 hodinový záznam). Na vodorovné ose je čas, svislá osa představuje frekvence (0-45Hz). Barva zastupuje četnost frekvence a to tak, že červená odpovídá největší četnosti a modrá naopak minimální. Spektrogram slouží lékařům pro stanovení kvality spánku a lze z něj určit zastoupení jednotlivých spánkových fází.



Obrázek 2.11 Spektrogram (obrázek byl vygenerován v programu EEG Lab [34])

3 Zpracování a analýza EEG signálu

3.1 Vzorkování signálu

Jelikož je EEG signál analogový, musí se v měřicím přístroji vzorkovat pomocí analogově-číslicového převodníku. Vzorkovací věta nám pak říká, jak správně tento spojitý signál vzorkovat, abychom z něj neztratili užitečnou informaci.

Vzorkovací věta

Jako autoři vzorkovací věty jsou nejčastěji uváděni americký matematik Claude Elwood Shannon a ruský radiotechnik Vladimír Alexandrovič Kotělnikov, a stanovuje podmínku, za které se při vzorkování signálu neztratí užitečná informace.

Formulace vzorkovací věty: Pokud signál x(t) spojitý v čase obsahuje pouze frekvenční složky s frekvencemi menšími než f_{max}, pak x(t) může být jednoznačně rekonstruován z posloupnosti ekvidistantních vzorků x(nT), pokud vzorkovací frekvence $f_{vz}=1/T$ je větší než $2*f_{max}$, čili

$$f_{vz} > 2f_{\max} \tag{3.1}$$

Signál obsahující pouze frekvenční složky s frekvencemi menšími než konečné číslo f_{max} se nazývá frekvenčně omezený signál, případně signál s omezeným frekvenčním spektrem. Frekvence $f_{vz}/2$ je Nyquistova frekvence. [21]

3.2 Fourierova transformace

Fourierova transformace patří mezi integrální transformace a vyjadřuje obraz signálu pomocí ortogonálních bázových funkcí. Vztah (3.2) přiřazuje k funkci času x(t) její obraz v komplexní rovině X(jω).

$$X(j\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt \tag{3.2}$$

, kde ω je kruhová frekvence e je definována následovně:

$$\omega = 2\pi f \tag{3.3}$$

Více informací o Fourierově transformaci lze nalézt v [24] nebo [21].

3.2.1 Fourierova transformace diskrétních signálů

Fourierova transformace je nejpoužívanější z transformací, vyjadřujících obraz signálu pomocí ortogonálních bázových funkcí. K popisu diskrétních signálů ve frekvenční oblasti se používají její modifikace Fourierova transformace diskrétních signálů (Fourier transform of discrete signals, FTD), označovaná též zkratkou DTFT. Vztah pro přímou transformaci se získá ze vztahu pro přímou Fourierovu transformaci náhradou integrálu sumou. Protože je Fourierův obraz (frekvenční spektrum) vzorkovaného signálu periodická funkce v ω s periodou $\omega_{vz} = 2\pi / T$, zapisuje se argument ve tvaru $e^{j\omega T}$. [21]

Přímá FTD je dána vztahem:

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x(nT) \cdot e^{-jn\omega T} \tag{3.4}$$

V teorii diskrétních signálů se často místo s kruhovou frekvencí ω pracuje s normovanou kruhovou frekvencí θ. Pomocí θ vyjádřené spektrum má periodu $\theta = 2\pi$ a platí pro ně:

$$X(e^{j\theta}) = \sum_{n=-\infty}^{\infty} x(n) \cdot e^{-jn\theta} \tag{3.5}$$

3.2.2 Diskrétní Fourierova transformace

Frekvenční spektrum vzorkovaného signálu $X(e^{j\theta})$ je spojitá funkce normované kruhové frekvence θ počítaná z nekonečného počtu hodnot x(n). Při zpracování signálů pomocí číslicových obvodů se pracuje s konečnými počty hodnot. Pracuje se i s konečnými posloupnostmi i ve frekvenční oblasti, tedy s konečnými počty vzorků frekvenčního spektra. Signály v oblasti času i frekvence mají stejný počet vzorků N a při výpočtech přímé i zpětné transformace se považují za periodické (pracujeme s periodickými prodlouženími signálů ze základního intervalu). Přechodem mezi signály x(n) v časové oblasti a signály ve frekvenční oblasti, které budeme značit X(k), je v tomto případě tzv. finitní Fourierova transformace. V případě diskrétních signálů se nazývá diskrétní Fourierova transformace a označuje se DFT. Tato transformace je definována vztahy:

$$X(k/NT) = T \cdot \sum_{n=0}^{N-1} x(nT) \cdot e^{-j\frac{2\pi}{N}nk} \text{ , k=0,1,2,…,N-1} \tag{3.6}$$

Při výpočtech DFT se v praxi používá zjednodušený vztah:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}nk} \text{ , k=0,1,2,…,N-1} \tag{3.7}$$

Frekvence odpovídající dané hodnotě k:

$$f_k = k/NT \tag{3.8}$$

Čas odpovídající hodnotě n:

$$t_n = nT \tag{3.9}$$

DFT tedy vypočte N hodnot spektra X(k) z N hodnot signálu x(n). Hodnoty spektra dostaneme pro diskrétní ekvidistantní hodnoty frekvencí začínající v f=0 a vzdálené od sebe o hodnoty:

$$\Delta f = \frac{1}{NT} = \frac{f_{vz}}{N} \tag{3.10}$$

Množině těchto hodnot se říká DFT mřížka. Hodnota Δf se často v literatuře nazývá frekvenční bin. N*T je celková doba odebírání signálu. [21]

3.2.3 Rychlá Fourierova transformace (FFT)

Výpočet pomocí DFT je pomalý a pro velký počet dat trvá nepřijatelně dlouho dobu. Pro výpočet N hodnot podle (3.5) je potřeba N^2 komplexních násobení a N*(N-1) komplexních sčítání, takže doba potřebná pro výpočet je přibližně dána časem potřebným pro provedení N^2 operací.

FFT čili rychlá Fourierova transformace (angl. Fast Fourier Transform) jsou velmi efektivní algoritmy výpočtu DFT. FFT se používá pro spektrální analýzu deterministických i stochastických signálů, pro výpočet konvoluce, frekvenčních charakteristik i pro číslicovou filtraci.

Základní (a nejpoužívanější) algoritmy FFT jsou navrženy pro délku transformace $N=2^m$, kde m je přirozené číslo. Tyto algoritmy využívají periodičnosti a symetrií komplexní exponenciály v (3.5). Tato exponenciála se označuje otáčecí činitel (angl. Twiddle factor) a označuje se W_N:

$$W_N = e^{-j\frac{2\pi}{N}} \tag{3.11}$$

Algoritmy FFT pro $N=2^m$ redukují počet operací pro výpočet N bodů DFT na:

$$\frac{N}{2}\log_2 N = \frac{Nm}{2} \tag{3.12}$$

Poměrná úspora v počtu operací a tedy i době výpočtu exponenciálně roste s délkou transformace a výpočet pro N=1024 proběhne zhruba 200 krát rychleji než při výpočtu podle definice DFT. [21]

Protože má otáčecí činitel (3.9) jednotkovou velikost, lze výpočet DFT znázornit jednoduše maticovým schématem, kde je úhel otáčecího činitele naznačen směrem šipek jednotkových vektorů ve čtvercové matici. Vynásobením této matice sloupcovým vektorem hodnot signálu x(n) získáme sloupcový vektor hodnot DFT spektra X(k). Obrázek 3.1 znázorňuje výpočet DFT pro N=8:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \uparrow & \nearrow & \rightarrow & \searrow & \downarrow & \swarrow & \leftarrow & \nwarrow \\ \uparrow & \rightarrow & \downarrow & \leftarrow & \uparrow & \rightarrow & \downarrow & \leftarrow \\ \uparrow & \searrow & \leftarrow & \nearrow & \downarrow & \nwarrow & \rightarrow & \swarrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \swarrow & \rightarrow & \nwarrow & \downarrow & \nearrow & \leftarrow & \searrow \\ \uparrow & \leftarrow & \downarrow & \rightarrow & \uparrow & \leftarrow & \downarrow & \rightarrow \\ \uparrow & \nwarrow & \leftarrow & \swarrow & \downarrow & \searrow & \rightarrow & \nearrow \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}$$

Obrázek 3.1 Maticové schéma algoritmu osmibodové DFT [21]

3.2.4 STFT

Spektrum nestacionárních signálů se mění v čase a nás zajímá nejen jeho (okamžitý) tvar, ale také jeho umístění v čase (časová lokalizace). Obecným nástrojem pro analýzu nestacionárních signálů jsou různé časově-frekvenční distribuce. Pro získání krátkodobého spektra resp. STFT spektra lze použít časově-frekvenční analýzu krátkodobou DFT.

Klasická Fourierova analýza neumožňuje časovou lokalizaci spektra. Pokud ale v definici Fourierovy transformace použijeme signál vynásobený oknem, jehož poloha vůči signálu se mění s parametrem τ, dostaneme spektrum, které je funkcí dvou proměnných $-\omega$ a τ. Jde tedy o vztah:

$$X(e^{j\theta},m)=\sum_{n=-\infty}^{n=\infty}x(n)w(n-mN)e^{-jn\theta} \tag{3.13}$$

Kde předpokládáme N krát rychlejší vzorkování než je výpočet spektra a frekvenci ω jsme nahradili frekvencí θ normovanou k vzorkovací frekvenci f_s:

$$\theta=\frac{\omega}{f_s}=\frac{2\pi f}{f_s} \tag{3.14}$$

STFT spektrum pro diskrétní čas a diskrétní frekvence je dáno vztahem

$$X(k,m)=\sum_{n=0}^{n=N-1}x(n)w(n-mN)e^{-j\frac{2\pi nk}{M}}=\sum_{n=0}^{n=N-1}x(n)w(n-mN)W_M^{nk} \tag{3.15}$$

STFT spektrum je v obecném případě komplexní funkce dvou reálných proměnných. Pro jeho grafické znázornění se používá tzv. spektrogram, což je kvadrát absolutní hodnoty STFT spektra.

Hodnoty spektrogramu nad rovinou t, ω se znázorňují barevně, buď intenzitou šedi nebo barevnou stupnicí, která může být umístěna po straně spektrogramu. [21]

3.2.5 Výkonové spektrum

Výkon na každé frekvenci je dán přímo druhou mocninou amplitudy složky Fourierova spektra. Amplituda složky $G(f_k)$ je pak dána následujícím vztahem:

$$G(f_k)=\frac{A_k}{2} \tag{3.16}$$

, kde A_k je amplituda k-té sinusovky, takže její druhá mocnina je $\frac{A_k^2}{4}$.

Jelikož je amplitudové spektrum sudá funkce, platí to podobně pro zápornou frekvenci a celkový výkon na frekvenci f_k bude dán součtem těchto složek:

$$G(f_k) = \frac{A_k^2}{2} \tag{3.17}$$

Celkový výkon může být získán buď integrací časového signálu nebo sečtením druhých mocnin amplitud všech frekvenčních složek (Parsevalův teorém). Protože už není ve výkonovém spektru obsažena informace o fázi, není možné z něj získat zpět původní časový signál.

3.3 Analýza signálů v časové oblasti

Hodnotu stochastických signálů pro určitý okamžik v budoucnu nelze vypočítat, ani když máme záznam jejich celého průběhu až do současného okamžiku (tzv. realizaci). Tyto signály lze popisovat pouze statisticky.

Stochastický signál (stochastický proces) je náhodná funkce času. V určitém okamžiku t_0 je hodnota této náhodné funkce náhodná veličina, která se nazývá řez náhodným procesem. Stochastický proces lze tedy definovat jako množinu těchto náhodných veličin pro všechna reálná t pro analogové procesy, případně pro všechna diskrétní t_n=nT, n je celé číslo, pro procesy diskrétní v čase.

Nestacionární náhodné signály mohou mít v čase proměnnou např. střední hodnotu, rozptyl nebo frekvenční spektrum. Náhodné signály se popisují z hlediska amplitudové struktury a z hlediska frekvenční nebo časové struktury. Popisy ve frekvenční a časové oblasti jsou vzájemně zastupitelné. [21]

Teorie odhadů

Střední hodnota

$$\bar{x} = \frac{1}{N}\sum_{n=0}^{N-1} x(nT) \tag{3.18}$$

Střední hodnota kvadrátu

$$\overline{x^2} = \lim_{N\to\infty} \frac{1}{N}\sum_{n=0}^{N-1} x^2(nT) \tag{3.19}$$

Rozptyl pro konečný počet vzorků N:

$$D[x] = \frac{1}{N-1}\sum_{n=0}^{N-1}(x(nT)-\bar{x})^2 \tag{3.20}$$

Středně kvadratická odchylka

$$\sigma_x = \sqrt{D[x]} \tag{3.21}$$

Standardní odchylka

$$s = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x(n)-\bar{x}\right)^2} \tag{3.22}$$

3.4 Segmentace

EEG signál je obecně nestacionární a nelze jej zpracovávat celý najednou. Proto je potřeba nejdříve provést segmentaci signálu. Segmentací pak obdržíme po částech stacionární úseky, které lze popsat matematickými metodami. Segmentace může být konstantní nebo adaptivní.

3.4.1 Konstantní segmentace

Konstantní segmentace je nejjednodušší způsob, jak rozdělit nestacionární signál. Nevyžaduje žádnou analýzu signálu, pouze rozděluje signál na části konstantní délky, v důsledku toho nemají hranice segmentů žádný vztah k charakteru signálu. Tyto segmenty pak mohou obsahovat směs vln různého tvaru a různých frekvencí, které si nejsou vůbec podobné. Pro přesnou klasifikaci tak není tato metoda vhodná. Výhodou této metody je především její výpočetní rychlost.

3.4.2 Adaptivní segmentace

Adaptivní segmentace řeší nedostatky konstantní segmentace. Výsledkem je signál rozdělený na segmenty stejných vlastností.

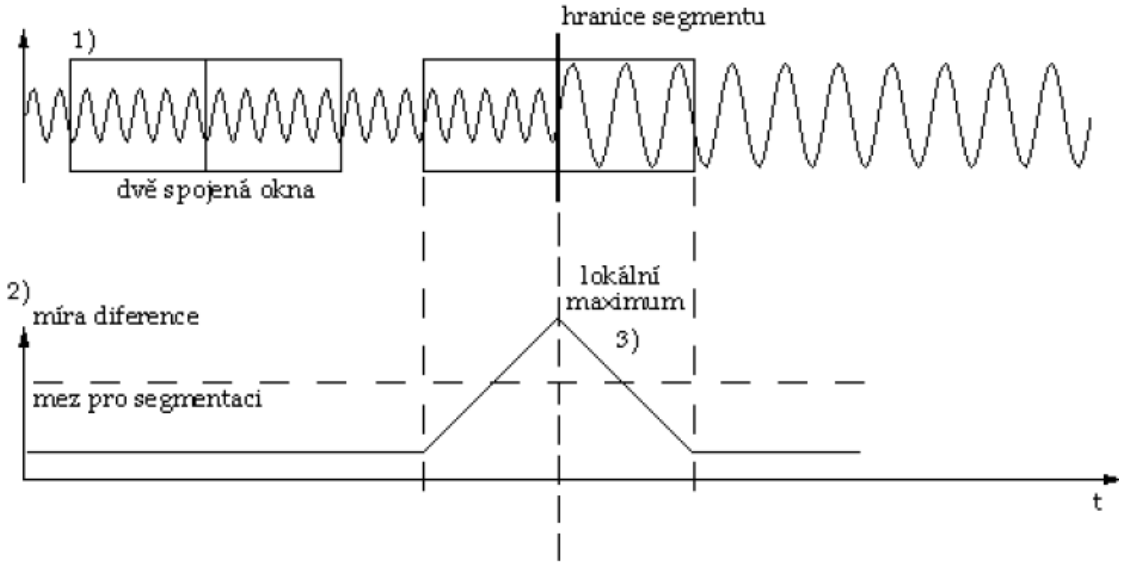
Metoda dvou oken

Algoritmus využívá okna, které je rozdělené na dvě poloviny a posouvá se po signálu, aby spočetl pro každý vzorek signálu hodnotu diference. Velikost této hodnoty je úměrná roz-

dílnosti dvou stacionárních segmentů. Hranice segmentů jsou umístěny v lokálních maximech takto spočteného signálu.

Algoritmus:

1) Posouvající okno je umístěno v okolí aktuálního vzorku
2) Okno se rozdělí na dvě poloviny tak, aby byl aktuální vzorek uprostřed okna
3) Pro obě poloviny se vypočte charakteristika signálu
4) Míra diference signálu se určí z rozdílu charakteristik obou polo oken



Obrázek 3.2 Princip adaptivní segmentace pomocí metody dvou oken [17]

Obrázek 3.2 ukazuje princip adaptivní segmentace za použití metody dvou oken. Pro výpočet charakteristiky signálu v polo okně se používá např. výpočet Teager energy.

Teager Energy

Výpočet Teager Energy je založen na nelineárním operátoru (NLEO). NLEO se vypočte dle následujícího vztahu:

$$\psi(n) = x^2(n) - x(n-1)x(n+1) \tag{3.23}$$

Celková velikost Teager Energy polo okna je pak vypočtena jako střední hodnota NLEO v polo okně:

$$TE = \frac{1}{N} \sum_{i=M}^{M+N-1} \psi(i) \tag{3.24}$$

3.5 Klasifikace

Segmentovaný EEG signál je následně klasifikován pomocí příznakových metod. Příznaky, kterými popisujeme objekt, můžeme uspořádat do *n*-rozměrného vektoru, který nazýváme vektor příznaků. Objekty jsou pak reprezentovány body v n-rozměrném prostoru. Klasifikátor zobrazuje příznakový prostor objektů na množinu indikátorů tříd. Pro funkci řady klasifikátorů je stěžejní výpočet míry podobnosti (popř. nepodobnosti) objektů.

Klasifikace je matematická metoda, kdy vstupní objekty x(i) jsou rozřazovány do tříd podle podobnosti. V našem případě se budeme zabývat pouze klasifikací s učitelem, která pro svou funkcionalitu potřebuje tzv. trénovací množinu (množina ukázek jednotlivých klasifikačních tříd). Základními představiteli klasifikace s učitelem jsou k-NN klasifikátor (1-NN – metoda nejbližších sousedů), neuronové sítě, rozhodovací stromy atd.

3.5.1 Podobnost objektů

Stanovení míry podobnosti objektů odpovídá nalezení vhodného předpisu π přiřazujícího každé dvojici objektů (O_i, O_j) číselnou hodnotu $\pi(O_i, O_j) = \pi_{i,j}$, která odpovídá míře podobnosti daných objektů a platí pro ni následující vztahy:

$$\pi(O_i, O_j) \geq 0 \ , \ 1 \leq i, j \leq N \tag{3.25}$$

$$\pi(O_i, O_j) = \pi(O_j, O_i) \tag{3.26}$$

$$\pi(O_i, O_i) \text{ je maximální} \tag{3.27}$$

, kde N je počet objektů.

Z předchozích vztahů vyplývá, že čím je větší hodnota $\pi(O_i, O_j)$, tím větší je vzájemná podobnost objektů.

V některých klasifikačních metodách je naopak vhodnější použít míru nepodobnosti. Její vlastnosti uvádějí následující vztahy:

$$\pi(O_i, O_j) \geq 0 \ , \ 1 \leq i, j \leq N \tag{3.28}$$

$$\pi(O_i, O_j) = \pi(O_j, O_i) \tag{3.29}$$

$$\pi(O_i, O_i) = 0 \tag{3.30}$$

Základními typy předpisu π jsou:

- koeficienty asociace
- koeficient korelace představující míry podobnosti objektů
- metriky představující míry nepodobnosti objektů

V našem případě se bude zabývat pouze nejběžnějším způsobem vyjádření podobnosti mezi objekty a to jsou metriky vycházející z geometrického modelu dat. Tyto metriky přiřadí dvojici bodů A, B v prostoru E_p číslo $\rho(A, B)$ a platí pro ně následující:

$$\rho(A, B) = 0 \Leftrightarrow A = B \tag{3.31}$$

$$\rho(A, B) \geq 0 \tag{3.32}$$

$$\rho(A, B) = \rho(B, A) \tag{3.33}$$

$$\rho(A, C) \leq \rho(A, B) + \rho(B, C) \tag{3.34}$$

Přiřadíme-li každému objektu popsanému p atributy jeden bod v euklidovském prostoru E_p jako jeho model, můžeme euklidovskou metriku (vzdálenost) použít jako míru nepodobnosti objektů. Objekty jsou si pak podobnější tím více, čím nižší je jejich vzdálenost. Definujme tedy dva body v prostoru následovně $A=\{a_1,a_2,\ldots,a_p\}$ a $B=\{b_1,b_2,\ldots,b_p\}$, Euklidovská vzdálenost pro tyto dva body je pak definována následujícím vztahem (3.35). [17]

$$\rho(A, B) = \sqrt{\sum_{k=1}^{p} \left(a_k - b_k\right)^2} \tag{3.35}$$

3.5.2 Standardizace dat

Různé hodnoty jednotlivých znaků objektů mohou způsobovat zdánlivou dominantnost některých z nich nebo naopak mohou mít jen minimální vliv na celkovou klasifikaci. Někdy je proto vhodné upravit data tak, aby byly navzájem souměřitelné.

Jednou z metod standardizace dat je standardizace směrodatnou odchylkou a provede se dle následujícího vztahu:

$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_i}{s_i} \tag{3.36}$$

Tato metoda se doporučuje použít v případech, kdy jsou atributy měřené v odlišných škálách a jednotkách. Takto standardizované hodnoty znaků pak mají střední hodnotu rovnu 0 a rozptyl 1.

3.5.3 Normalizace objektů

Objekty pro klasifikaci jsou určeny vektory o p složkách představujících hodnoty vybraných p znaků. Normy těchto vektorů mohou někdy nežádoucím způsobem ovlivňovat výsledky kvantitativního hodnocení podobností objektů. V takových případech je vhodné normalizovat tyto vektory, aby měli stejnou normu (nejlépe jednotkovou).

3.5.4 Metoda nejbližších sousedů

Metody klasifikace založené na vzdálenosti využívají skutečnosti, že pokud jsou dva záznamy klasifikovány do stejné třídy, musí mít něco společného, tj. musí si být podobné. Vlastní klasifikace pak spočívá ve zvolení vhodné metriky podobnosti záznamů. Na základě této metriky vyhledáme pro právě předpovídanou instanci k nejbližších sousedů v množině trénovacích příkladů. Vlastní klasifikace nového záznamu probíhá tak, že je vybráno k trénovacích záznamů, které jsou nejbližší klasifikovanému záznamu. Analyzovaný záznam je klasifikován do třídy, do které náleží nejvíce z vybraných trénovacích záznamů. KNN používá pro určení blízkosti záznamů vzdálenost vektorů v Euklidovském prostoru (jsou tedy nutné číselné atributy s uspořádáním) podle vztahu (3.35).

3.5.4.1 Hlavní vlastnosti

Rychlost klasifikace

V obecném případě klasifikace podle všech prvků trénovací množiny je třeba porovnat všechny její prvky s právě klasifikovaným, tudíž časová náročnost algoritmu je úměrná počtu prvků v trénovací množině. Časovou náročnost lze částečně snížit vhodným setříděním prvků nebo jejich vhodným rozdělením do skupin (tzv. k-D stromy), případně kondenzací trénovací množiny.

Přesnost klasifikace

Přesnost klasifikace je obecně horší než u Bayesovské klasifikace. Pro počet prvků trénovací množiny blížící se do nekonečna se chyba blíží chybě Bayesovské klasifikace. Pomocí „editace“ trénovací množiny se chyba dostane na úroveň

$$\lim_{K\to\infty} P^{NN}(\varepsilon) < 2P(\varepsilon) \tag{3.37}$$

Při praktickém použití se chyba klasifikace k-NN blíží chybě mnohem složitějších metod, jako např. neuronových sítí.

$$P^{NN}(\varepsilon) \approx P^{Neuronové\ sítě} \tag{3.38}$$

Proto se s úspěchem používá jako referenční metoda.

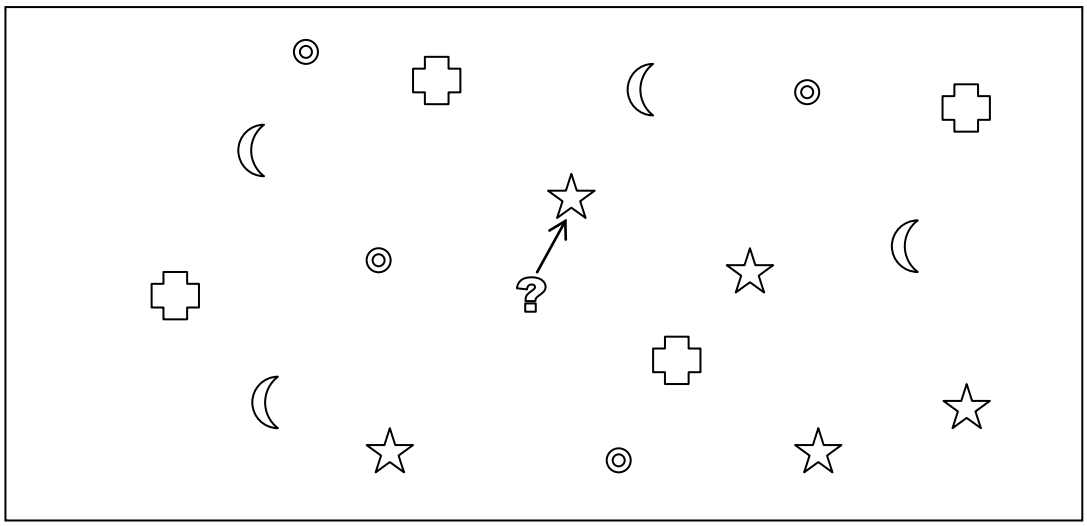
Obtížnost implementace

Algoritmus je sám o sobě velice jednoduchý, není třeba žádných složitých a dlouhých výpočtů. Jedná se prakticky pouze o porovnávání prvků trénovací a testovací množiny. Paměťová náročnost je úměrná celkovému počtu prvků a její náročnost lze snížit kondenzací trénovací množiny.

3.5.4.2 Postup klasifikace

1-NN

1) Zjistíme vzdálenosti všech prvků trénovací množiny od neznámého prvku
2) Vybereme ten prvek trénovací množiny, který je nejblíže a neznámý prvek pak klasifikujeme do stejné třídy



Obrázek 3.3 Klasifikace nejbližšího souseda 1-NN

Obrázek 3.3 ukazuje příklad klasifikace podle nejbližšího souseda 1-NN do čtyř tříd. Máme-li čtyři třídy: kolečko, hvězdička, křížek a měsíček, pak neznámý prvek „?“ klasifikujeme jako hvězdičku.

4 Zobrazení dat

Pro zobrazení výsledků byl zvolen programovací jazyk JAVA. V následujících kapitolách jsou teoreticky popsány vybrané části tohoto programovacího jazyka, které byly důležité pro vlastní implementaci aplikace.

4.1 Java SWING

Java Swing je knihovna funkcí naprogramovaná v Javě, která je součástí frameworku Java Foundation Classes (JFC) a je primárně určena k tvorbě uživatelského rozhraní. Součástí této knihovny jsou především: komponenty pro práci s daty, dialogy, grafikou a obrázky. Výhodou této knihovny je podpora tzv. systémových Look&Feels, což je sada funkcí, které napodobují vzhled a chování grafického rozhraní operačního systému, na kterém aplikace běží, a tím zvyšují intuitivnost ovládání programu. Podstatnou výhodou oproti konkurenční technologii Standard Widget Toolkit (SWT) je to, že knihovny jsou integrovanou součástí nejrozšířenější standardní distribuce J2SE (Java 2 Standard Edition).

4.2 Java 2D

Oproti AWT (Abstract Windowing Toolkit), které je pro kreslení 2D grafiky neefektivní, poskytuje Java 2D API robustní balíček nástrojů pro kreslení a práci s obrázky ve vysoké kvalitě. Toto API umožňuje kreslit transparentní obrázky, gradienty, použití lokálních typů písma, definovat typy kreslícího pera a transformaci souřadnic. Java 2D optimalizuje základní kreslící funkce a zrychluje tak celkový čas potřebný pro generování 2D obrázků.

4.3 Soubory v jazyce Java

V balíku java.io najdeme třídu, která nám poskytne vše potřebné k práci se soubory a tou je třída File. Nepředstavuje přímo konkrétní soubor, ale tzv. abstraktní cestu (tedy obecně jakoukoli cestu identifikující nějaký soubor). Může odkazovat na platný soubor, ale také nemusí.

V řadě tříd ze standardních knihoven Javy najdeme metody, které vyžadují jako svůj argument název souboru. Prakticky ve všech případech lze použít jak textový řetězec (String), tak právě instanci třídy File.

Vstupně/výstupní operace lze v Javě realizovat několika způsoby, základními z nich jsou streamy. Stream si lze představit jako trubku, jejíž konec máme k dispozici a můžeme

"čerpat" data z něho (tedy číst) nebo naopak do něho (tj. zapisovat). Streamů existuje (z hlediska implementace) celá řada. Pro streamy je charakteristická především jejich sekvenčnost.

4.3.1 RandomAccessFile

Tato třída umožňuje náhodný přístup k souboru a práci s ním. Nepatří do hierarchie tříd streamů. Implementuje rozhraní DataInput a DataOutput a tyto dvě rozhraní implementují DataInputStream resp. DataOutputStream. Soubor může být otevřen buď pro čtení nebo pro čtení i zápis zároveň. Způsob práce se souborem je určen podle parametru v konstruktoru ("r" pro čtení nebo "rw" pro zápis i čtení). Po otevření souboru je ukazatel nastaven na začátek souboru a načítání dat je prováděno pomocí metody read(). RandomAccessFile je thread-safe a jeho použití je vhodné především pokud načítáme ze souboru malé úseky dat nebo se po souboru často přesouváme.

4.3.2 Streamy

V Javě je každý stream reprezentován jako objekt, tedy instance nějaké třídy. Balík java.io obsahuje hierarchii základních tříd, se kterými si pro běžné operace vystačíme (další streamy najdeme i v jiných standardních balících).

Známe dvě hlavní kategorie streamů: binární a textové. Liší se způsobem práce se znaky, binární streamy pracují se "surovými" bajty (tak, jak jsou), zatímco streamy textové pojímají bajty, resp. skupiny bajtů způsobem, který odpovídá nastavení prostředí. Binární vstupní streamy jsou odvozeny od abstraktní třídy InputStream, výstupní od třídy OutputStream. Textové pak od třídy Reader, resp. Writer.

4.3.2.1 Ošetření chyb

Základem práce se streamy je ošetření chyb, které se mohou vyskytnout. Téměř všechny chybové stavy jsou řešeny výjimkami. Tou základní je IOException, kterou může vyhodit naprostá většina streamových metod.

4.3.2.2 Základní druhy streamů

Výhodou streamů je jejich univerzálnost, pracujeme s nimi prakticky stejně, ať už se jedná o data v souborech na disku, o síťovou komunikaci, komunikaci mezi vlákny apod. Zmíníme se zde tedy o základních typech streamů.

Souborové streamy

Z hlediska Javy se nerozlišují vlastnosti souborového systému, se soubory se pracuje vždy stejně, jen nás zajímá, zda soubor existuje, lze z něj číst nebo do něj zapisovat. Konec souboru je definován hodnotou -1.

Protože je FileInputStream thread-safe, zabírá čtení poměrně dost času. Pro zvýšení efektivity můžeme např. použít načítání dat do většího bytového pole (bufferu).

Bufferované streamy

Protože píšeme platformově nezávislé programy, nemůžeme spoléhat na prostředky operačního systému. Tyto nedostatky odstraňují streamy, které obsahují vlastní buffer a optimalizují tak přístup k datům. Buffer navíc minimalizuje zbytečné přístupy na disk, pokud bychom zapisovali nebo četli malé objemy dat (např. bajty). Při použití bufferovaných výstupních streamů není zaručeno, kdy se data z bufferu přesunou do navazujícího streamu. K zajištění zápisu dat z bufferu můžeme zavolat metodu flush().

4.3.3 FileChannel

FileCahnnel je nový způsob I/O z balíčku java.nio, který je dostupný od JDK 1.4 a navyšuje výkon při práci se soubory díky přiblížení strukturám I/O v OS. Kanál se vytvoří pomocí metody getChannel() a s kanálem se komunikuje pouze přes ByteBuffer. ByteBuffer, je vytvořen pomocí allocateDirect() metody, která uloží data do nativních datových struktur OS. Tyto buffery tak nabízejí vyšší výkon, v závislosti na OS.

4.4 Ukládání a přístup k vícerozměrným datům v jazyce Java

Tato část se věnuje možnostem ukládání vícerozměrných dat v jazyce Java. Pozornost je věnována především několika základním přístupům, které jsou porovnány z hlediska použitelnosti pro různé operace.

4.4.1 Array

Toto základní pole každého programovacího jazyka je z hlediska procházení a ukládání dat nejrychlejší. Není vhodné, pokud se délka pole mění a je třeba ho často upravovat (přidávat nebo odebírat prvky pole). Nejrychlejší metodou pro přesun prvků v poli je systémová metoda arraycopy.

4.4.2 Rozhraní Collection

Toto rozhraní reprezentuje základní abstraktní typ kontajnerů povahy seznam nebo množina. Rozhraní umožňuje ukládání do kolekce (metody add, addAll), vyhledávání a přístup k prvkům kolekce (contains, containsAll, iterator), vyřazení z kolekce (clear, remove, removeAll, retainAll) a další pomocné metody (isEmpty, size, toArray, hashCode, equals). Rozhraní Collection také obecně nespecifikuje, jak (kam) má probíhat ukládání nových prvků metodami add, addAll.

4.4.3 ArrayList

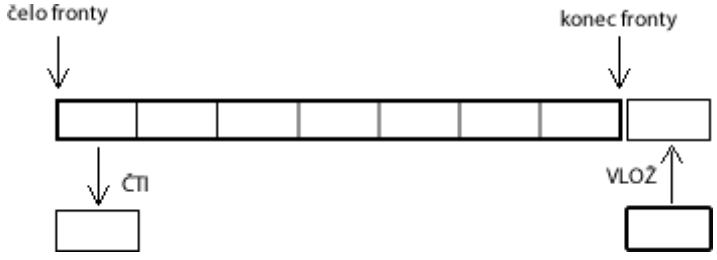
ArrayList je kontajner implementující přímo rozhraní List. Ve skutečnosti se jedná o kontajner postavený nad běžným polem a umožňující to, co klasické pole neumí: měnit počet uložitelných prvků, nové prvky vkládat mezi již uložené atd. Nevýhodou tohoto pole je pomalé procházení jednotlivými prvky.

4.4.4 LinkedList

Je též implementací List, ale kromě metod obecných seznamů disponuje navíc šikovnými metodami pro přímé vkládání a odebírání prvků na začátku a na konci seznamu. Díky tomu je možná LinkedList chápat jako zásobník (stack), frontu (queue) nebo oboustrannou frontu (dequeue). Tento seznam je nejvhodnější pro procházení pomocí iterátoru.

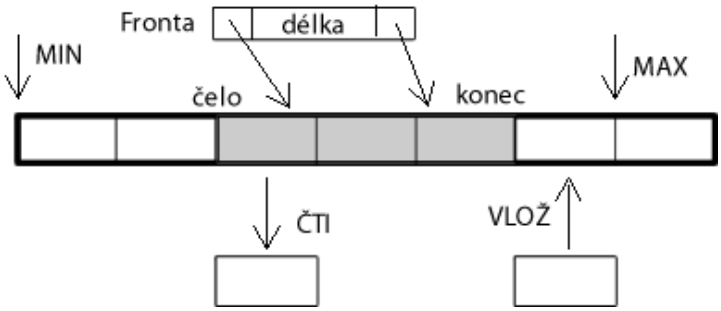
4.4.5 Datový typ fronta

Do fronty lze libovolně přidávat a odebírat prvky. Prvky se do fronty zařazují na konec, ale čte se z ní vždy ze začátku (z čela fronty). Ruší se prvek, který byl do fronty zařazen jako první a nachází se v čele fronty. Proto se fronta označuje jako struktura FIFO (z angl. First Input, First Output). Používá se všude, kde je potřeba požadavky uspokojovat v tom pořadí, jak vznikaly. [20]



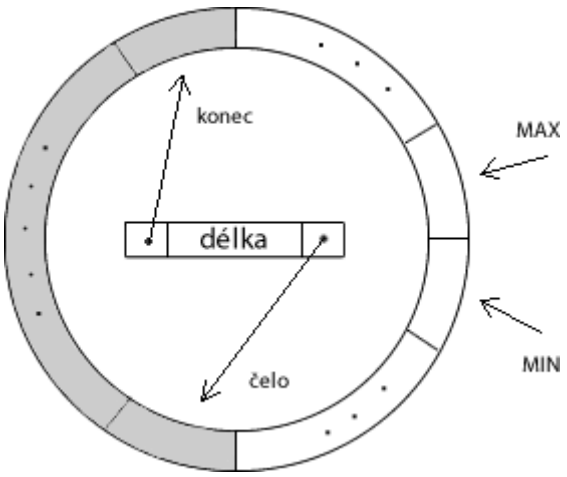
Obrázek 4.1 Fronta s pevným čelem [20]

Rušení prvního prvku ve frontě můžeme chápat tak, že se všechny prvky ve frontě posunou směrem k začátku fronty, čímž prvek v čele fronty „z fronty vypadne“. Frontu můžeme rovněž implementovat polem nebo spojovým seznamem. Při implementaci polem bude fronta reprezentována polem prvků a indexem prvku v čela a na konci fronty. Čelo fronty může být trvale v prvním prvku pole (viz. Obrázek 4.1).



Obrázek 4.2 Fronta reprezentovaná polem s posunem čela fronty [20]

Popsaná reprezentace není optimální. Operace vyjmutí prvního prvku je časově náročná neboť vyžaduje přesun všech prvků fronty kromě prvního. Rušení prvního prvku můžeme rovněž dosáhnout změnou ukazatele čelo na začátek fronty. Položku ve frontě zrušíme tak, že posuneme ukazatel čelo (viz. Obrázek 4.2). Avšak výběrem položek z fronty se uvolňují prvky na začátku pole a může snadno nastat situace, že nelze položku do fronty zařadit, i když není pole zaplněné. Tento nedostatek snadno odstraníme tím, že pole uspořádáme do kruhu (viz. Obrázek 4.3). Tím se stane prvek s minimálním indexem sousedem prvku s maximálním indexem. [20]

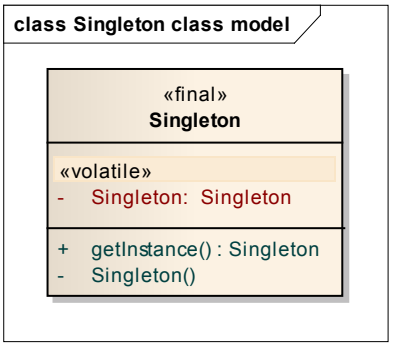


Obrázek 4.3 Kruhová reprezentace pole [20]

4.5 Singleton pattern v jazyce Java

Singleton patter je způsob, jak zajistit, aby pro daný objekt existovala pouze jedna jeho instance. To je vhodný způsob, jak sdílet data mezi více objekty. Základem Singleton patternu je, aby všechny konstruktory byly typu private. Přístup k proměnným tohoto objektu je zajištěn pomocí public metod. Aby se zabránilo vytvoření objektu pomocí clone, musí být tento objekt definován jako final. Metoda clone() je definována jako privátní nebo může jako public, ale a v jejím těle musí být přidán kód throw new CloneNotSupportedException().

4.5.1 Class diagram



4.5.2 Vytvoření singletonu pomocí synchronizace

Následující řešení je bezpečně použitelné ve více vláknové aplikaci v Javě verze 5.0 a výše:

```
final public class Singleton {

   private volatile static Singleton singleton;

   private Singleton() {}

   public static Singleton getSingleton() {
      if (singleton == null) {
         synchronized(Singleton.class){
                           // potřebné, když dvě vlákna čekají na monitor v době, kdy se singleton vytváří
            if(singleton == null) {
               singleton = new Singleton();
            }
         }
      }
      return singleton;
   }

   @Override
   public Object clone() throws CloneNotSupportedException {
      throw new CloneNotSupportedException();
   }

}
```

4.6 Multithreading v jazyce Java

Java umožňuje tzv. multithreading neboli paralelní běh dvou či více částí programu. Každá paralelně běžící část programu se v Javě nazývá vlákno (thread). Důležité je, že jednotlivá vlákna lze naprogramovat téměř nezávisle a pouze v případě, že sdílí společná data nebo používají stejné prostředky (zařízení), je třeba zajistit jejich řádnou synchronizaci.

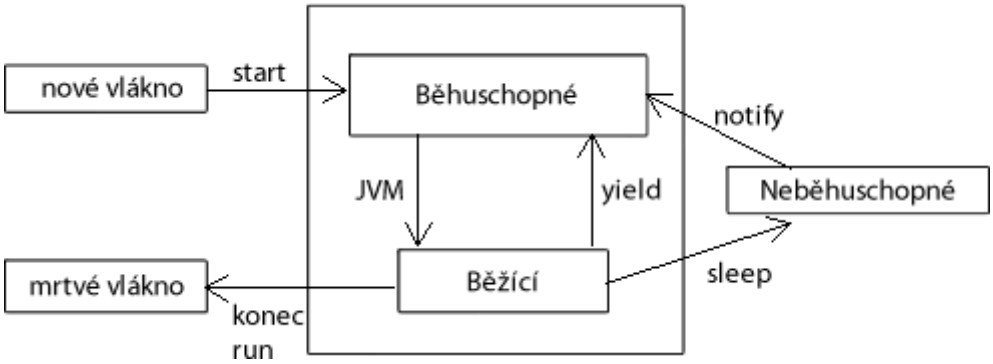
Na většině dnešních počítačů s jedním procesorem se samozřejmě nejedná o fyzicky současný běh vláken, ale jednotlivá vlákna se na procesoru střídají. Více se o vláknech se můžeme dočíst např. v [16].

4.6.1 Vlákna

Každé vlákno se v daném okamžiku nachází v právě jednom z těchto stavů:

- Nové vlákno (new thread) - vlákno je vytvořeno, ale ještě nebylo spuštěno (nejsou ještě alokovány systémové prostředky vlákna).
- "Běhuschopný" stav (runnable) – do tohoto stavu se vlákno dostane zavoláním metody start(), na počítači s jedním procesorem je pouze jedno právě běžící
- "Neběhuschopný" stav (not runnable) - do tohoto stavu se vlákno dostane, pokud:
    - je uspáno metodou sleep(),
    - je "odstaveno" metodou suspend(),
    - čeká v metodě wait(),
    - čeká na vstupní/výstupní zařízení.
- Mrtvé vlákno (dead thread) – vlákno, jehož metoda run() skončila nebo byla zavolána metoda stop()

Přechody mezi jednotlivými stavy znázorňuje Obrázek 4.4.



Obrázek 4.4 Stavy vláken

Na počítači s jedním procesorem, který nepodporuje paralelní běh instrukcí, se vlákna o procesor dělí, podle pravidel, která určuje plánovač (scheduler).

4.6.2 Plánování (scheduling)

Java používá plánování podle priority: každé vlákno má přiděleno číslo, prioritu. Rozsah priorit je určen konstantami třídy Thread MIN_PRIORITY až MAX_PRIORITY. Číselný rozsah těchto priorit je od 0 do 9, kde vyšší hodnota znamená vyšší prioritu. Běžící vlákno musí mít nejvyšší prioritu, pokud se dostane do běhu schopného stavu vlákno s vyšší prioritou, je mu přiřazen procesor. Zavoláním metody yield() může běžící vlákno poskytnout procesor jiným vláknům se stejnou prioritou. Potřeba synchronizace vzniká všude tam, kde je možné (avšak nepřípustné) používat současně zařízení nebo sdílet společná data.

4.6.3 Synchronizace kritických sekcí

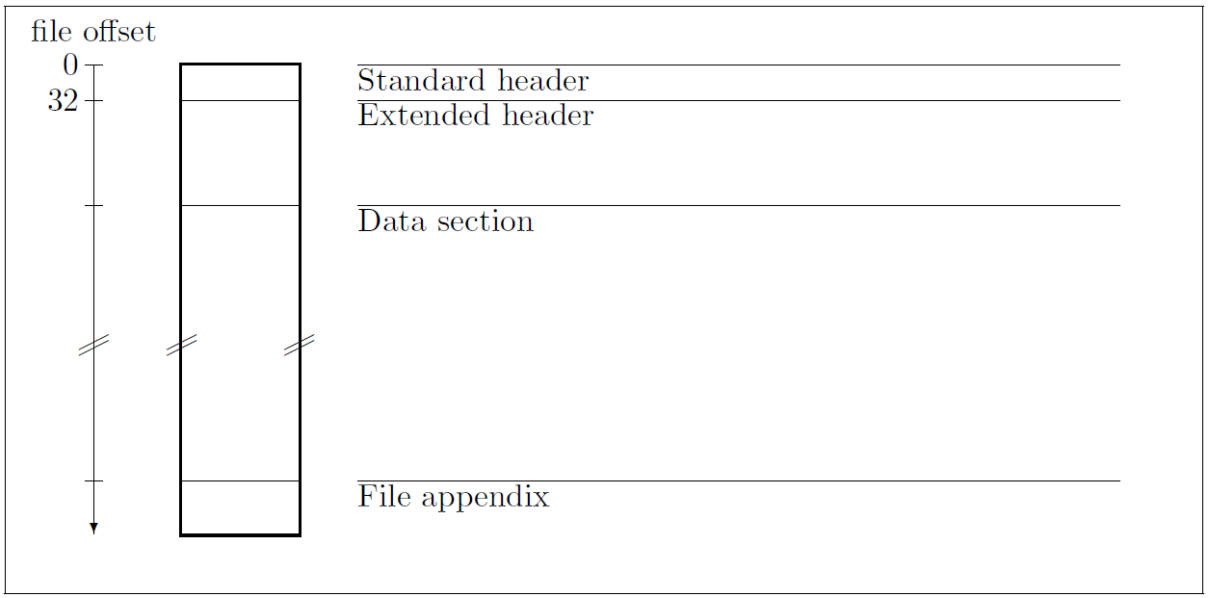
Vyloučení současného běhu kritických sekcí lze provést například pomocí tzv. monitorů. Monitor se obecně skládá z dat a funkcí (metod) jako zámků (locks) nad daty. Při vstupu do synchronizované kritické sekce vlákno získá monitor (zámek je uzamčen). Monitor je uvolněn ve chvíli, kdy vlákno opustí kritickou sekci. Do kritické sekce může tedy vždy vstoupit pouze jedno vlákno. Synchronizovanou kritickou sekcí v Javě může být blok nebo metoda. Synchronizovaná metoda se v programu označí modifikátorem synchronized a je vždy provedena jako nedělitelný (atomický) celek, bez možnosti přerušení.

5 Vstupní zdroje dat

5.1 D-file soubor

D-file je formát souboru, který je navržen speciálně pro ukládání EEG záznamů. Tento soubor je vytvářen v zaznamenávacím programu EASREC. Tyto soubory mohou být přehrávány např. v prohlížecím programu EASKERN nebo Easys2.

D-file soubory poskytují vhodnou reprezentaci digitalizovaných dat, která jsou zaznamenávána z několika kanálů současně. Hlavní jednotkou těchto souborů je vzorek (data_sample). D-file je lineární sekvencí těchto vzorků, které byly v pravidelných časových okamžicích snímány z měřícího zařízení. Všechny vzorky jsou uloženy jako integery (běžný princip A/D převodníků) a současně je v souboru uloženo měřítko (scaling_factor) pro přepočet na jejich přesnou skutečnou hodnotu.



Obrázek 5.1 Rozdělení D-file souboru

D-file se skládá z několika sekcí:

- header – obsahuje celkové informace o datech a struktuře d-file souboru
- data – obsahuje sekvenci vzorků dat
- file appendix – obsahuje dodatečné informace

Header se může skládat ze dvou částí: standard header a extended header (někdy také označován jako "xheader"). Standard header je povinný a má vždy stejnou délku. Obsahuje

všechny základní informace k tomu, abychom mohli správně přečíst uložená data. Extended header je volitelný a obsahuje dodatečné informace, např. popis kanálů, kalibrační informace, ID pacienta apod. Navíc může extended header obsahovat důležité informace pro budoucí analýzu a manipulaci s daty.

Začátek záznamu extended header je uložen ve standard header. Velikost extended header může být různá, maximální délka je určena celkovou velikostí standard header. Velikost je definována v kilobajtech, 32 bytů je rezervováno pro standard header a zbylá část sekce pak může být využita pro extended header.

Data section následuje hned za extended header. Začátek datové sekce je uložen ve standard header a její velikost je dána celkovým počtem uložených vzorků a délkou jednoho vzorku.

Appendix section je poslední sekcí D-file souboru a je volitelná. Běžně se v ní ukládají tagové informace. Reference na tuto sekci je uložen v extended header.

Velikost header section (standard + extended header) nesmí přesáhnout 64 KB, obdobně tak ani velikost seznamu tagů nesmí přesáhnout 64 KB.

5.2 EEG zařízení

Pro komunikaci s programem bylo využito zařízení od firmy Unimedis - BrainScope - systém pro digitální záznam, analýzu a archivaci EEG/EP a polysomnografie. Byla použita standardní sestava pro rutinní měření EEG, která se skládá ze dvou hlavních položek (nahrávací stanice, popisovací stanice).

Nahrávací stanice

- digitální EEG zesilovače EADS 221 – 32 kanálů (headbox)
- programovatelný výbojkový fotostimulátor
- základní sada EEG příslušenství
- pojízdný stolek umožňující pohodlné převážení celého systému
- nahrávací počítač a monitor
- klávesnice, myš, operační systém

- programový systém EASYS3 pro natáčení grafů
- databázový systém pro vedení kartotéky pacientů a EEG grafů
- programové vybavení WaveFinder pro popis záznamů

Popisovací stanice

- popisovací počítač a monitor s vysokým rozlišením
- klávesnice, myš, operační systém
- laserová černobílá nebo barevná tiskárna
- databázový systém pro vedení kartotéky pacientů a EEG grafů
- programové vybavení WaveFinder pro popis záznamů

6 Implementace programu

Aplikace byla původně implementována v programovacím jazyce JAVA verze 6. Během této doby společnost Oracle vyvinula stabilní beta verzi JAVA 7, a proto jsem se rozhodl upravit zdrojové kódy a přizpůsobit je novější verzi. Z důvodu optimalizace nebyl v aplikaci implementován algoritmus pro filtraci signálu a odstranění šumu.

Výsledná implementace aplikace umožňuje v reálném čase sledovat signály z kanálů, frekvenční analýzu (aktuální i dlouhodobý trend), relativní výkony ve frekvenčních pásmech (aktuální i dlouhodobý trend) a klasifikaci signálů. Aplikace využívá metody popsané v předešlých kapitolách.

Aplikace je navržena tak, že existuje jeden základní *frame* a do něj je vložen jeden *pane* a v něm existuje několik dalších *Jpanel* objektů. Každý z těchto objektů je samostatné vlákno, které se vykonává nezávisle na ostatních. Veškerá data jsou uložena v objektu *DataHolder*, který reprezentuje pattern typu *singleton*. K tomuto základnímu objektu pak přistupují všechny komponenty aplikace pro získání nebo uložení dat.

Aplikace je napsána plně objektově a objekty jsou rozděleny do balíčků následovně:
communication – obsahuje objekty pro komunikaci se vstupními zdroji dat
datamanipulation – obsahuje objekty na úpravu dat (FFT, Artefakty)
eeg – obsahuje základní objekty aplikace (hlavní třídu, Segment, Artefakt a DataHolder)
graphics – obsahuje objekty typu *Pane*, které visuálně zobrazují data
threads – obsahuje sadu vláken, které samostatně zpracovávají data (klasifikace)

6.1 Nastavení aplikace

Vzhledem k tomu, že je aplikace výpočetně náročná, byly hodnoty, které významně ovlivňují výkonnost aplikace, napevno definovány ve zdrojovém kódu a uživatel nemá možnost je měnit. Vlastnosti, které by měl mít uživatel možnost měnit, jsou editovatelné v nastavení aplikace.

Nastavení aplikace se otevře pomocí menu: O programu -> Nastavení aplikace. Po kliknutí na tuto položku v menu je zobrazeno okno, kde je možné přenastavit tyto parametry:

Počet kanálů

Počet kanálů, které se budou zobrazovat v okně kanálů a pro které se bude provádět klasifikace.

Způsob kreslení

Určuje metodu překreslování okna kanálů, jsou zde na výběr dvě možnosti (cyklický signál se překresluje stále dokola, posun – signál se v čase posouvá zprava do leva).

Číslo kanálu pro spektrogram

Určuje pozici kanálu, pro který se bude počítat frekvenční analýza, která je pak následně zobrazena v okně spektra, spektrogramu a dlouhodobého spektrogramu. Tento index se může během analýzy měnit (minulé hodnoty zůstávají a nově přidané hodnoty už odpovídají frekvenčním charakteristikám nově zvolené elektrody).

Předpokládaná délka záznamu

Je doba, pro kterou chceme signál analyzovat. Podle této hodnoty si aplikace nastaví měřítka pro přepočet dlouhodobých trendů (např. počet vzorků na pixel).

IP adresa a port

Toto nastavení je nutné, pokud chceme komunikovat se zařízením pomocí TCP/IP. Nastavíme tedy IP adresu a port, kde chceme pakety přijímat. Tyto hodnoty musejí být shodné s nastavením IP a portu, které se nastaví měřícímu zařízení jako cílová adresa.

Vzorkovací frekvence

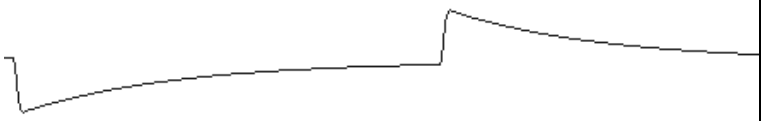
Slouží především pro komunikaci pomocí TCP/IP a určuje vzorkovací frekvenci, se kterou bude zařízení signál vzorkovat. Pro D-file soubor se vzorkovací frekvence načítá přímo z hlavičky souboru a toto nastavení je ignorováno.

Nastavení aplikace je nutné provést ještě před načtením D-file souboru nebo připojením k TCP/IP.

6.2 Příprava trénovací množiny

Pro trénovací množinu bylo využito dat, která byla naměřena v laboratoři na katedře biokybernetiky. Signály byly pomocí adaptivní segmentace, která je implementována v aplikaci, rozděleny na segmenty. Z vytvořených segmentů byly vybrány segmenty, které nejvíce zastupovaly námi zvolenou klasifikační třídu. Následně byly signály z D-file souboru

načteny do Matlabu. Hodnoty pro daný segment, který je určen začínajícím indexem a koncovým indexem, byly vyexporovány a uloženy do textového souboru s artefakty. Tyto segmenty pak sloužily jako klasifikační třída v trénovací množině. Tabulka 6.1 uvádí průběhy jednotlivých klasifikačních tříd a jejich vlastnosti.

Klasifikační třída	Vlastnosti
Inicializace 	Počet vzorků: 506 Stř. hodnota ampl. fr. složek: 595uV Min: -96,2uV Max: 93,1uV Rozptyl: 1678uV^2 Absolutní delta: 78,67*10^6 uV^2 Absolutní theta: 14,56*10^6 uV^2 Absolutní alpha: 5,65*10^6 uV^2 Absolutní beta 9,81*10^6 uV^2 Relativní delta: 72,4% Relativní theta: 13,4% Relativní alpha: 5,2% Relativní beta: 9% Střední hodnota: -1,47uV
Alfa vlny 	Počet vzorků: 401 Stř. hodnota ampl. fr. složek: 221uV Min: -33uV Max: 17,7uV Rozptyl: 77,65uV^2 Absolutní delta: 1,76*10^6 uV^2 Absolutní theta: 0,22*10^6 uV^2 Absolutní alpha: 0,19*10^6 uV^2 Absolutní beta 2,57*10^6 uV^2 Relativní delta: 36,6% Relativní theta: 4,7% Relativní alpha: 4,1% Relativní beta: 54,6% Střední hodnota: -3,27uV
Relaxace	Počet vzorků: 303 Stř. hodnota ampl. fr. složek: 56uV Min: -6,9uV Max: 5,8uV Rozptyl: 7,19uV^2 Absolutní delta: 0,12*10^6 uV^2 Absolutní theta: 0,08*10^6 uV^2 Absolutní alpha: 0,05*10^6 uV^2 Absolutní beta: 0,08*10^6 uV^2 Relativní delta: 37,9% Relativní theta: 23,1% Relativní alpha: 16% Relativní beta: 23% Střední hodnota: 1,08uV
Svalová aktivita 	Počet vzorků: 241 Stř. hodnota ampl. fr. složek: 429uV Min: -93,9uV Max: 52,8uV Rozptyl: 735,7uV^2 Absolutní delta: 9,43*10^6 uV^2 Absolutní theta: 0,13*10^6 uV^2 Absolutní alpha: 0,07*10^6 uV^2 Absolutní beta: 2,51*10^6 uV^2

	Relativní delta: 77,7% Relativní theta: 1% Relativní alpha: 0,6% Relativní beta: 20,7% Střední hodnota: -17,04uV

Tabulka 6.1 Klasifikační třídy

6.3 Načítání z D-file souboru

Pro načtení informací z hlavičky D-file souboru byl použit přístup pomocí RandomAccessFile. Podle specifikace D-file se sekvenčně prochází celá hlavička souboru a načítají se z ní tato data:

- název programu
- typ souboru
- počet kanálů
- počet dalších kanálů (např. EOG, ECG, EVT, atd.)
- vzorkovací frekvence
- celkový počet vzorků pro jeden kanál
- validační pole
- násobící koeficient (kalibrace dat)
- fyzická nula
- offset, na kterém začíná blok s daty
- offset, na kterém začíná blok s rozšiřujícími informacemi o záznamu

Po načtení těchto základních informací se ukazatel přesune na začátek bloku s rozšiřujícími informacemi a odtud přečte názvy všech kanálů, které jsou uloženy v souboru. Po přečtení rozšiřujících informací je ukazatel souboru nastaven na začátek bloku s daty. Další načítání ze souboru je prováděno pomocí FileInputStream, Channel a ByteBuffer.

Při čtení dat ze souboru dochází k rozdílným odezvám. Obecně platí, že čím méně z disku čteme, tím je průměrná doba, která je potřebná k přečtení dat, menší. Pro čtení dat z disku je také lepší načítat jeden větší blok než vícekrát menší úseky dat. Abych minimalizoval přístup na disk při čtení ze souboru, zvolil jsem použití metody pomocí FileInputStream s použitím bufferu. Velikost bufferu je závislá na počtu kanálů v D-file souboru a je vypočtena následovně:

buffer_size = pocet_kanalu * 2048

Z tohoto výpočtu vyplývá, že pro každý kanál je alokována paměť o velikosti 2KB. Pro 24 kanálů je tedy velikost bufferu 48KB.

Program načte jeden velký blok dat a v pravidelných časových okamžicích (určeny vzorkovací frekvencí) se data načítají z bufferu. Tento způsob implementace odstraňuje různé doby odezvy při čtení z disku a urychluje tak celý proces načítání dat ze souboru.

Při načítání dat z D-file souboru jsme narazili na problém s datovými typy. I když Java poskytuje pro načítání primitivních datových typů nativní funkce, nejde je při čtení souboru použít. Všechny primitivní proměnné jsou totiž z pohledu Javy znaménkové. Z tohoto důvodu jsem musel vytvořit speciální funkce, které data načtené ze souboru převedou na požadovaný datový typ. V následujícím kódu je funkce, která upraví dva načtené byty (uložené v poli bytů) na primitivní typ integer:

```
public static int arr2int (byte[] arr, int start) {
    int low = arr[start] & 0xff;
    int high = arr[start+1] & 0xff;
    return (int)( high << 8 | low );
}
```

Problematika datových typů v Javě je popsána např. [39].

6.4 Komunikace se záznamovou stanicí

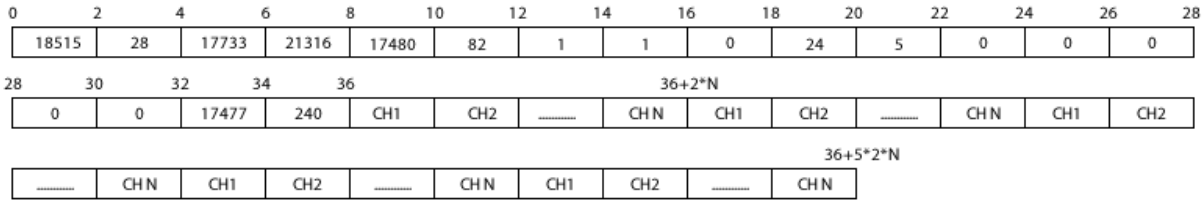
Pro testování a simulaci načítání EEG signálů ze zařízení firmy Unimedis byl použit simulátor, který nám firma poskytla. Tento simulátor se spouští pomocí dávkového souboru v MS-DOS a umožňuje nastavit několik parametrů:

- název D-file souboru, ze kterého se budou data načítat a simulovat takto fyzické zařízení
- vzorkovací frekvence
- počet kanálů
- IP adresu a port, na který se budou data posílat

Ukázka konfiguračního souboru:

```
verbose 5
tcpserver start EEDSPC 22222 crlf
set udpserver 127.0.0.1 3001
set udpfsamp 250
set udpnchan 24
set udpnvect 5
set sigtype dfile 024p04.d
make start
```

Firma Unimedis bohužel neposkytla žádné další informace týkající se komunikace zařízení pomocí TCP/IP a s tím i související datovou strukturu paketu. Proto jsme nejprve museli pakety v aplikaci odchytávat a vypisovat si jejich obsah. Následně jsme se pak snažili v paketech vyhledat závislosti, které by definovali jejich datovou strukturu. Z jejich struktury se nám podařilo vypozorovat jen některé informace, tyto informace jsou pro analýzu dat v paketu nejdůležitější a tak byly i dostačující k tomu, abychom mohli data získaná ze zařízení zobrazit v aplikaci.



Obrázek 6.1 UDP paket z EEG zařízení

Obrázek 6.1 ukazuje vysledovanou datovou strukturu paketu. Na prvních 36-ti bytech jsou hodnoty, které se nám nepodařilo identifikovat, kromě bytu 15 a 16, které určují pořadí paketu. Byte 17 a 18 určuje počet kanálů, to je důležitá informace pro načtení vzorků z kanálů. Po bytu 36 následuje sekvence vzorků z kanálů a má následující tvar: kanál1 až kanál N, kde N je počet kanálů definovaných v paketu. Tato sekvence se opakuje v paketu pětkrát.

Z následujícího pozorování tedy vyplývá, že pro vzorkovací frekvenci 250Hz se UDP paket odesílá každých 20ms a obsahuje pět vzorků pro každý kanál.

Aplikace tyto pakety přijímá a parsuje z nich jednotlivé vzorky, které následně přesune do svého bufferu. Z tohoto bufferu si samostatné vlákno v pravidelných intervalech (urče-

no vzorkovací frekvencí) vybírá uložené vzorky a předává je dále do singleton objektu, kde k nim mají přístup další moduly aplikace a mohou je použít pro analýzu.

6.5 Načítání artefaktů

Artefakty se načítají z textového souboru, ve kterém jsou uloženy v jednoduchém formátu:

Název artefaktu 1: data1, data2, data3, …
Název artefaktu 2: data1, data2, data3, …

Tento model umožňuje načíst libovolný počet artefaktů. Důležité je, aby měly artefakty stejnou vzorkovací frekvenci jako právě analyzovaný signál. Načtené artefakty jsou zobrazeny v Tabulka 6.1.

6.6 Frekvenční analýza

Program byl optimalizován pro vzorkovací frekvenci 250Hz, což je v současné době nejvíce používaný standard pro vzorkování EEG signálů. S rostoucí vzorkovací frekvencí se zvyšují nároky na výpočet a je tedy omezujícím faktorem pro použití aplikace.

Frekvenční analýza využívá algoritmu pro rychlou Fourierovu transformaci (FFT). Její výpočet je prováděn každou vteřinu a spouští se v samostatně běžícím vlákně vždy, když je ze vstupního zdroje načten počet vzorků odpovídající vzorkovací frekvenci. Aktuální data se překopírují do tohoto vlákna pomocí systémové funkce *copy()*. Pokud je jejich velikost menší než N^2, je zbytek doplněn nulami. Pro vzorkovací frekvenci 250Hz se tedy doplní šest nul. Pro tuto posloupnost je pak následně provedena Fourierova transformace. Výsledky Fourierovy transformace se překopírují do pole, které udržuje jejich aktuální hodnotu. Aby se minimalizovala paměťová náročnost, jsou překopírovány pouze frekvenční složky od 0Hz do 40Hz.

Společně s tímto polem ještě existuje pole, které slouží jako dočasný buffer pro přepočítávání průměrné frekvenční amplitudy pro dlouhodobý spektrogram a dlouhodobý trend relativních výkonů ve frekvenčních pásmech. Velikost tohoto pole je určena předpokládanou dobou záznamu, kterou uživatel zadá v nastavení aplikace, a odpovídá počtu sekund na jeden pixel. Pokud je toto pole plné, vypočte se průměrná hodnota a ukazatel se přesune opět na začátek pole.

Protože jsou do signálu doplněny nuly, dojde ke zmenšení rozestupu mezi frekvencemi, které jsou vypočteny pomocí FFT a je proto potřeba přepočítat skutečné frekvenční hodnoty pomocí frekvenčního binu, který se vypočte dle vztahu (3.10).

Pro STFT se používá průměrování bez překrytí a na jednotlivé úseky signálu se před výpočtem FFT aplikuje okno Hann. Pro zrychlení výpočtů se okno aplikuje pouze pro zobrazení spektrogramu. Při analýze segmentů se na jednotlivé segmenty aplikuje pouze obdélníkové okno.

6.7 Segmentace

V programu je implementována adaptivní segmentace, která využívá dvou posuvných oken a v každém z nich počítá Teager energy (TE), která je založena na průměrné hodnotě nelineárního operátoru v celém polo okně. Minimální délka segmentu je nastavena na 0,3s a maximální délka segmentu byla zvolena na 2s.

Vzhledem k časové náročnosti při výpočtu TE byly tyto hodnoty experimentálně zvoleny a nastaveny napevno tak, aby je nemohl uživatel měnit, jelikož by jejich změnou mohlo dojít k nesprávnému chodu aplikace.

Délka segmentu byla zvolena tak, aby byla co nejkratší a to z několika důvodů:

- není potřeba použít velký buffer s posledními hodnotami z kanálů pro hledání lokálních maxim TE signálu
- výsledek následné klasifikace segmentu je zpožděn maximálně o 2s

Pokud by byla maximální délka segmentu např. 30s, lékař by dostal informaci o tom, v jakém stavu se pacient nacházel až za 30s. Toto by nebylo vhodné především tam, kde se jedná o analýzu závažných artefaktů např. epileptický.

Algoritmus adaptivní segmentace

1) Pro každé polo okno se spočítá TE dle vztahu (3.24)
2) Spočtené hodnoty TE obou oken jsou od sebe odečteny a získáme tak míru diference signálů, která je následně uložena do speciálního pole
3) Pokud je míra diference větší než stanovený práh, je program nastaven do režimu hledání lokálního maxima a aktuální hodnota je uložena jako lokální maximum

4) Pokud klesne míra diference pod stanovený práh nebo je míra diference menší než 50% poslední maximální hodnoty a je-li vzdálenost maxima od poslední hranice větší než minimální délka segmentu, je hranice segmentu umístěna do místa, kde byla míra diference maximální
5) Pokud nebyl překročen práh a poslední hranice segmentu je od aktuální pozice vzdálena o maximální délku segmentu, je v aktuálním místě vytvořena hranice segmentu

Práh jsme v naší aplikaci experimentálně odladili a nastavili na hodnotu 5. Obrázek 6.4 ukazuje ohraničení segmentů, které představují zatnutí svalů.

6.8 Klasifikace segmentů

Segmenty signálu jsou klasifikovány v samostatně běžícím vláknu. Pro klasifikaci segmentů byla použita metoda nejbližšího souseda (1-NN). Hlavním důvodem použití této metody byla především její malá výpočetní náročnost při poskytnutí výsledků přibližně stejné kvality, jako kdybychom použili neuronovou síť.

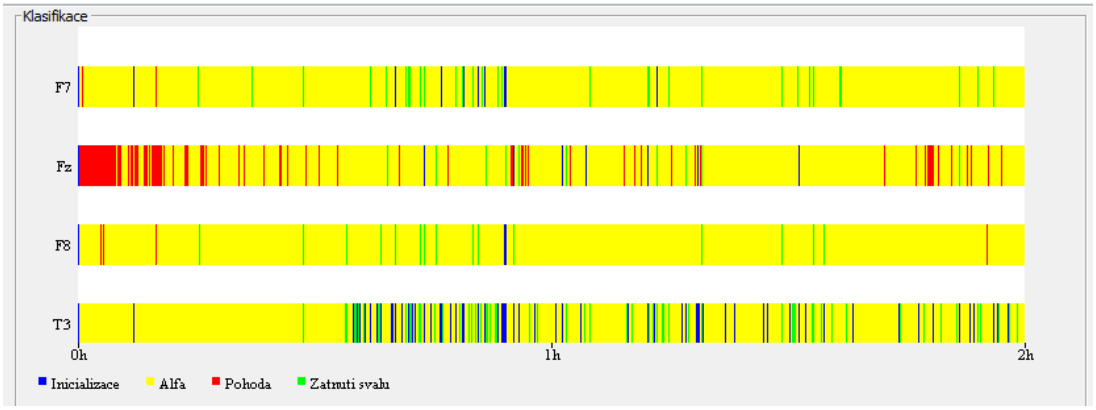
Aby byl potlačen vliv příznaků, které dosahují mnohem vyšších hodnot, jsou jednotlivé příznaky normalizovány na jednotkový vektor. Dále je pro každý příznak stanovena váha, která určuje, jak moc konkrétní příznak ovlivňuje podobnost segmentů a tím i výslednou klasifikaci. Tabulka 6.2 ukazuje vybrané příznaky, podle kterých byla provedena klasifikace, s jejich maximální hodnotou a váhou. Maximální hodnota slouží pro normalizaci na jednotkový vektor a je pevně stanovena z vysledovaných hodnot jednotlivých příznaků. Její nastavení na pevně danou hodnotu má výhodu především v tom, že se nemusí při analýze nového segmentu vždy zjišťovat maximální hodnota pro konkrétní příznak a tím i provádění přepočtu klasifikace již analyzovaných a oklasifikovaných segmentů. Nevýhodou tohoto řešení je, že při nevhodně stanové hodnotě pak může aplikace klasifikovat některé segmenty do jiné klasifikační skupiny.

Váhy pro jednotlivé příznaky byly nastaveny podle defaultních hodnot v programu EEGlab [34] a jejich hodnoty nemůže uživatel měnit. Jak je vidět z Tabulka 6.2 má na podobnost segmentů největší vliv absolutní hodnota výkonu v pásmu theta, naopak nejmenší vliv má absolutní hodnota výkonu v pásmu beta.

Příznak	Maximální hodnota	Váha
Maximální amplituda	120uV	1
Minimální amplituda	120uV	1
Rozptyl	$1000uV^2$	2,6
Střední hodnota	100uV	1
Absolutní delta	$1000V^2$	0,3
Absolutní theta	$1000V^2$	3,7
Absolutní alfa	$1000V^2$	3
Absolutní beta	$1000V^2$	0,001
Relativní delta	100%	1
Relativní theta	100%	1
Relativní alfa	100%	1
Relativní beta	100%	1

Tabulka 6.2 Příznaky použité pro klasifikaci segmentů

Obrázek 6.2 ukazuje výsledek klasifikace na 2hodinovém spánkovém záznamu. Jak je vidět na začátku všech kanálů probíhala inicializace (modrá barva), kterou aplikace správně oklasifikovala. Na většině kanálů pak převážně převládala aktivita, která je nejvíce podobná vlnám alfa.

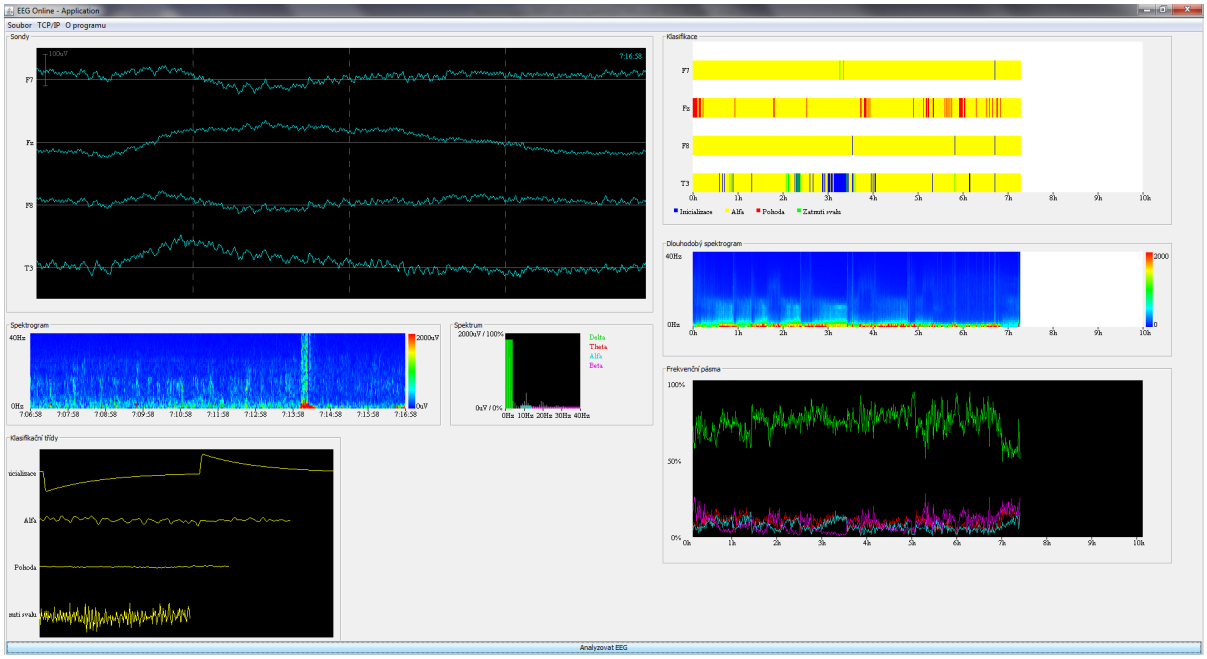


Obrázek 6.2 Klasifikace spánkového záznamu

6.9 Popis aplikace

Aplikace je opticky rozdělena do dvou polo částí: v levé části jsou zobrazena okna, která zobrazují aktuální data, v té pravé jsou pak zobrazeny dlouhodobé trendy. Mezi aktuální

data patří: signály z kanálů, aktuální 10-ti minutový spektrogram, aktuální frekvenční složky a relativní výkony v jednotlivých frekvenčních pásmech.



Obrázek 6.3 Ukázka aplikace

6.9.1 Kanálové okno

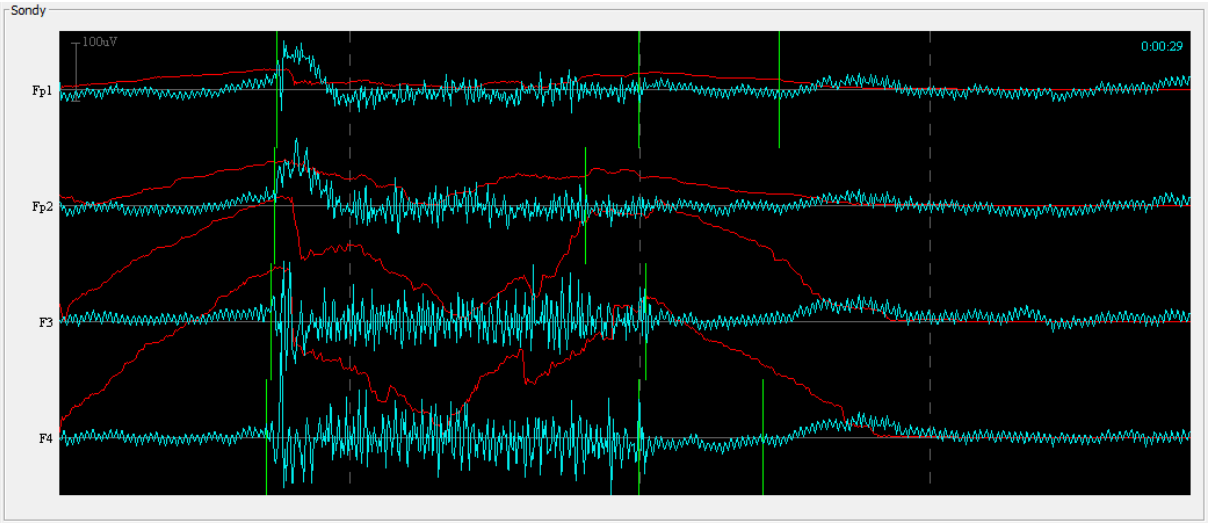
Toto okno slouží pro zobrazení signálů z kanálů. Okno má konstantní šířku a pro vzorkovací frekvenci 250Hz zobrazuje poslední 3s signálu. Společně se signály kanálů je také možné zobrazit signál sloužící pro segmentaci (Teager Energy). Hranice segmentů jsou pak odděleny zelenou čarou. Z důvodu ušetření procesorového času bylo vykreslování hranic segmentů a segmentačního signálu zakázáno a je možné je povolit pouze ve zdrojovém kódu při ladění aplikace.

Okno se překresluje každých deset milisekund, je na něm tedy aplikovaný 100Hz scan. V nastavení aplikace si uživatel může volit ze dvou způsobů vykreslování signálů. První z nich je plynulé vykreslování, tzn, že se celý signál neustále posouvá zprava doleva. Druhým způsobem, který je v aplikaci nastaven jako defaultní, je cyklické vykreslování. V tomto režimu se signál vykresluje postupně zleva doprava, pokud dojde vykreslování signálu na konec okna, začíná se vykreslovat opět od začátku a předchozí signál se překresluje aktuálním. V pravé části okna se zobrazuje uplynulý čas měření, který se neustále aktualizuje.

Okno přepočítává poměr pro kreslení jednoho kanálu podle celkového počtu kanálů. Rozsah jednoho kanálu je pevně nastaven na hodnoty od -100uV do 100uV, což odpovídá typickému rozsahu signálu EEG pro dospělého člověka. Pokud je amplituda signálu větší, než

je tento rozsah, je signál zobrazen buď v prostoru sousedních kanálů, nebo není signál vidět. Toto omezení je implementováno především kvůli rychlosti překreslování. Tato implementace ušetří procesorový čas, který by jinak musel procesor využít na přepočítání měřítka při každém nově načteném vzorku.

Překreslování signálů lze pozastavit kliknutím levého tlačítka myši na okno kanálů, pokud uživatel klikne podruhé, signály se začnou znovu překreslovat.



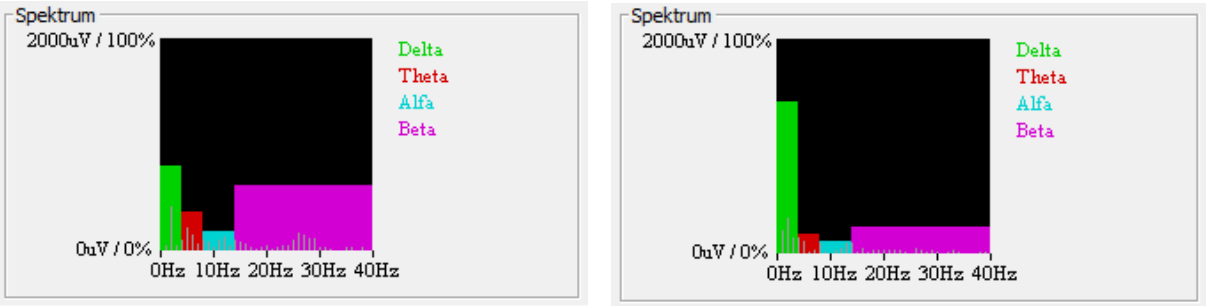
Obrázek 6.4 Okno kanálů

6.9.2 Okno aktuálních spekter

V tomto okně se zobrazují amplitudy frekvencí, které jsou obsaženy v signálu. Okno je napevno nastaveno tak, aby zobrazovalo frekvence od 0Hz do 40Hz. Pro úsporu počtu výpočtů, je také napevno stanovena hodnota maximální amplitudy frekvenční složky. Z pozorování maximálních hodnot frekvenčních složek jsem tuto hodnotu nastavil na 2000uV. Pokud je amplituda frekvenční složky větší než 2000uV, je její hodnota oříznuta na 2000uV.

Kromě frekvenčních složek jsou zde zobrazeny i relativní výkony v pásmech delta, theta, alfa a beta. Tyto výkony jsou počítány jako poměr k celkovému výkonu na frekvencích 0Hz až 40 Hz. Maximální hodnota výkonu tedy odpovídá 100% a součet všech výkonů je pak také 100%. Výkony jsou od sebe barevně odděleny a jsou vykresleny vždy pod frekvencemi, které odpovídají příslušnému pásmu.

Celé okno se aktualizuje každou 1s a zobrazuje tak uživateli aktuální zastoupení frekvenčních složek v signálu a jeho relativní výkon v pásmech delta, theta, alfa a beta.



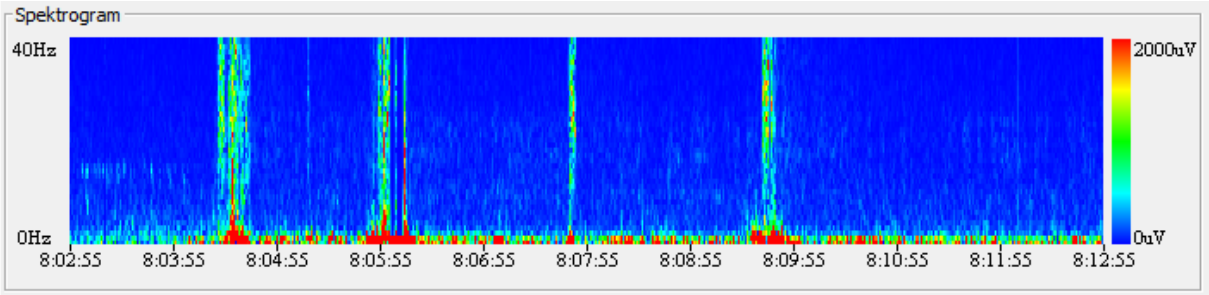
Obrázek 6.5 Aktuální frekvenční analýza

6.9.3 Aktuální spektrogram

Toto okno slouží pro zobrazení zastoupení frekvencí za posledních deset minut. Každý sloupec o šířce jednoho pixelu odpovídá 1s záznamu. Vodorovná osa odpovídá času měření a je rozdělena po minutě. Na svislé ose je zobrazena frekvence od 0Hz do 40Hz. Barva v grafu odpovídá amplitudě frekvence, převod amplitudy na barvu je zobrazen v pravé části okna. Červená odpovídá nejvyšším amplitudám frekvenčních složek a modrá naopak těm nejnižším.

Obdobně jako v okně aktuálních spekter je zde napevno nastavena maximální hodnota amplitudy frekvenční složky na 2000uV. Pokud je v daném okamžiku hodnota amplitudy některé frekvenční složky větší než 2000uV je tato hodnota oříznuta na 2000uV.

V rámci optimalizace a minimalizace provádění operací na něj není aplikován žádný filtr plovoucího průměru ani normalizace dat v jednotlivých řádcích. Spektrogram je tedy vykreslován přímo bez dalších dodatečných úprav či normalizací.

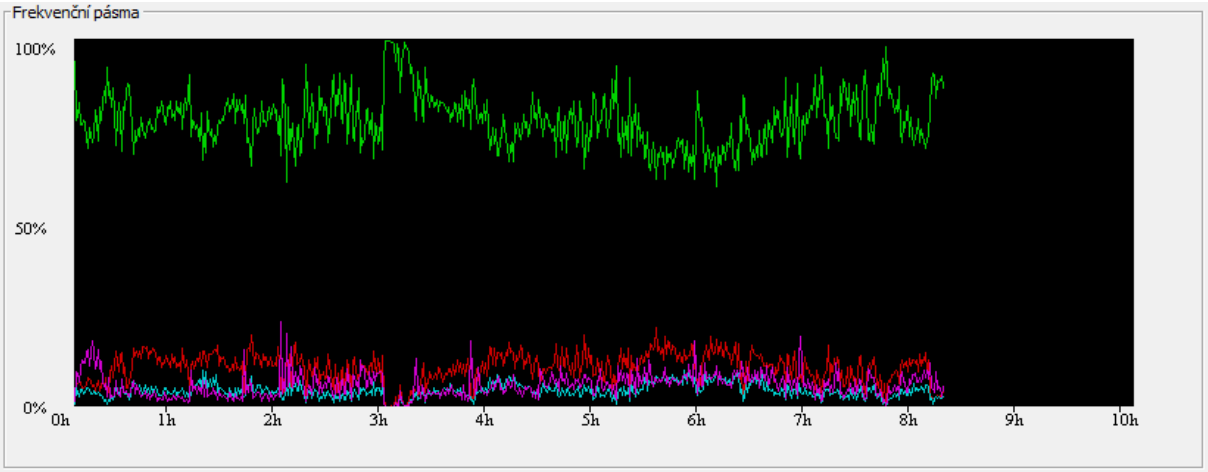


Obrázek 6.6 Aktuální spektrogram

6.9.4 Celovečerní frekvenční pásma

Tento graf zobrazuje dlouhodobý trend relativních výkonů ve frekvenčních pásmech. Na vodorovné ose je zobrazen čas, který uživatel zadal v nastavení aplikace. Na základě toho-

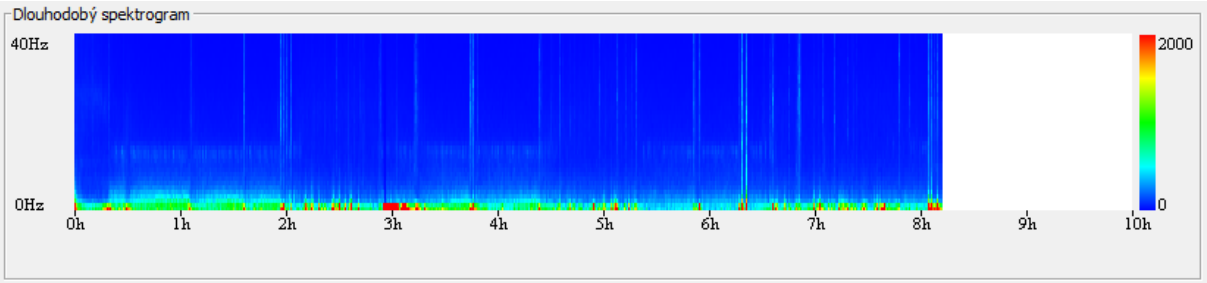
to času se upraví měřítko pro vykreslování dat. Na svislé ose je pak vyneseno procentuální zastoupení frekvenčního pásma. Maximální hodnota odpovídá stoprocentnímu zastoupení, nejnižší je nulové zastoupení frekvenčního pásma v signálu. Barevná legenda k jednotlivým frekvenčním pásmům je uvedena v okně aktuálních spekter a barvy jsou zde shodné. Na obrázku je vidět, jak se během spánku měnilo zastoupení jednotlivých pásem, kolem třetí hodiny u pacienta téměř převažovala delta aktivita (zelená barva), což odpovídá hluboké fázi spánku.



Obrázek 6.7 Dlouhodobý trend frekvenčních pásem

6.9.5 Celovečerní spektrogram

Celovečerní spektrogram je obdobou aktuálního spektrogramu, ale místo aktuálních dat, zobrazuje vývoj během celé doby měření. Obdobně jako předchozí okno, upravuje měřítko pro správné vykreslování dat na ose x. Tento graf je umístěn pod oknem relativních výkonů a je v něm vidět podobnost trendů během měření.

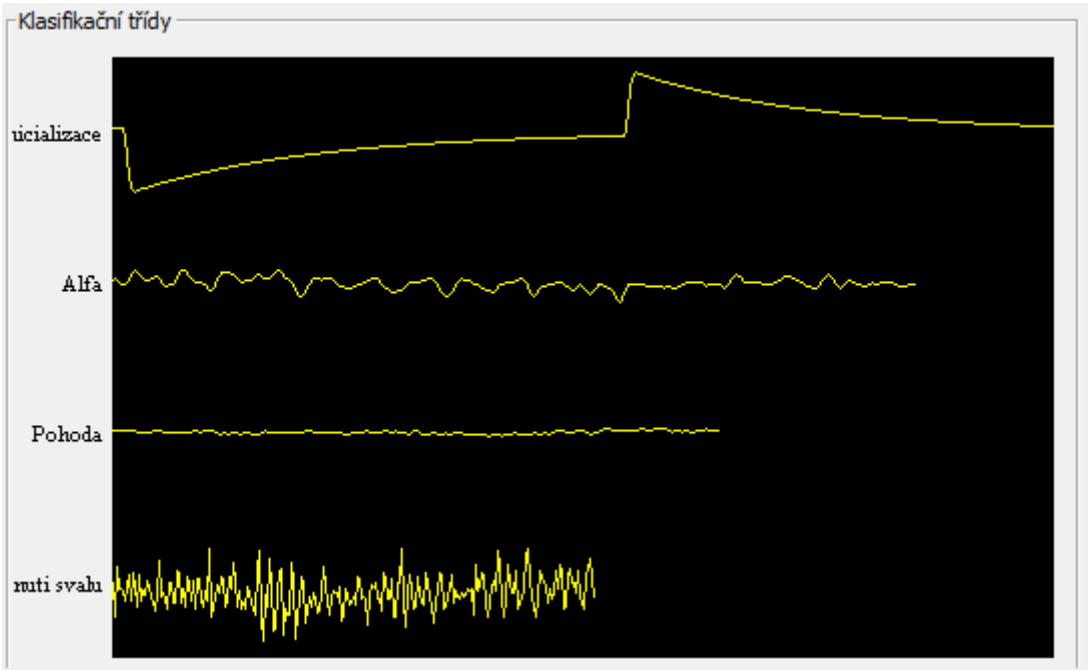


Obrázek 6.8 Dlouhodobý spektrogram

6.9.6 Načtené artefakty

V tomto okně jsou zobrazeny názvy a průběhy klasifikačních tříd, které jsou načteny z textového souboru. Vertikální rozmezí pro vykreslení jedné klasifikační třídy je obdobně jako v okně kanálů upraveno podle počtu načtených tříd. Pro maximální a minimální hodnotu

amplitudy je pevně dána hodnota 120uV. V levé části okna jsou pak vypsány názvy jednotlivých klasifikačních tříd.



Obrázek 6.9 Klasifikační třídy

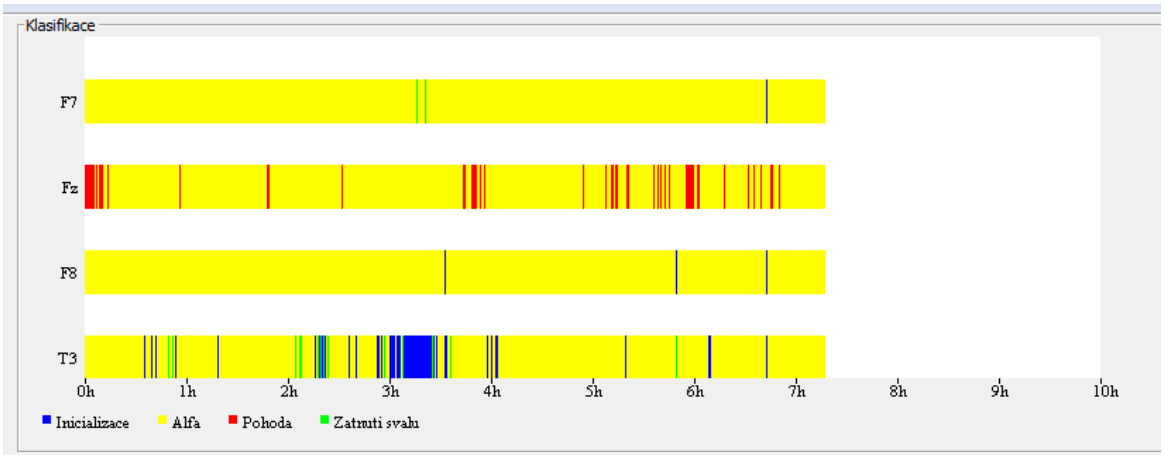
6.9.7 Klasifikace

Toto okno zobrazuje výsledky klasifikace záznamu z jednotlivých kanálů. Jeho vertikální rozsah pro jeden kanál je přepočítáván podle počtu kanálů, které uživatel v aplikaci nastaví. Názvy jednotlivých kanálů, pro které se klasifikace provádí, jsou zobrazeny na svislé ose. Pokud se data načítají ze záznamové stanice, jsou názvy kanálů číslovány od 1. Na vodorovné ose je zobrazen čas. Tato osa se také dynamicky přepočítává podle předpokládaného času měření, který uživatel v aplikaci nastaví. Tento čas pak také slouží k přepočtu vzorků, které se použijí pro zobrazení jednoho pixelu. Pod vodorovnou osou s časem je zobrazena legenda klasifikačních tříd. Barvy pro klasifikační třídy jsou nastaveny napevno a nelze je v aplikaci měnit.

Vlákno, které zobrazuje toto okno, postupně prochází segmenty, které jsou oklasifikovány, a určuje barvu na aktuálním pixelu dle následujícího algoritmu:

1) Pro aktuální pixel stanoví začínající a koncovou pozici oblasti (vypočte se z předpokládané doby záznamu zadané uživatelem a šířky okna).

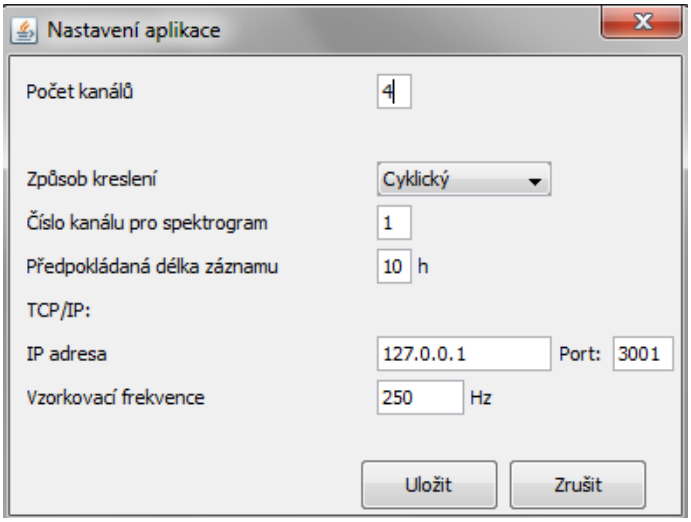
2) Pokud je segment oklasifikován a jeho hranice protínají nebo segment leží celý v oblasti aktuálního pixelu, zvýší se počet vzorků třídy (shodná s klasifikační třídou segmentu) oblasti o počet vzorků, které spadají do oblasti.
3) Aktuální pixel je pak obarven podle klasifikační třídy, která je v oblasti nejvíce zastoupena.



Obrázek 6.10 Klasifikace EEG signálů

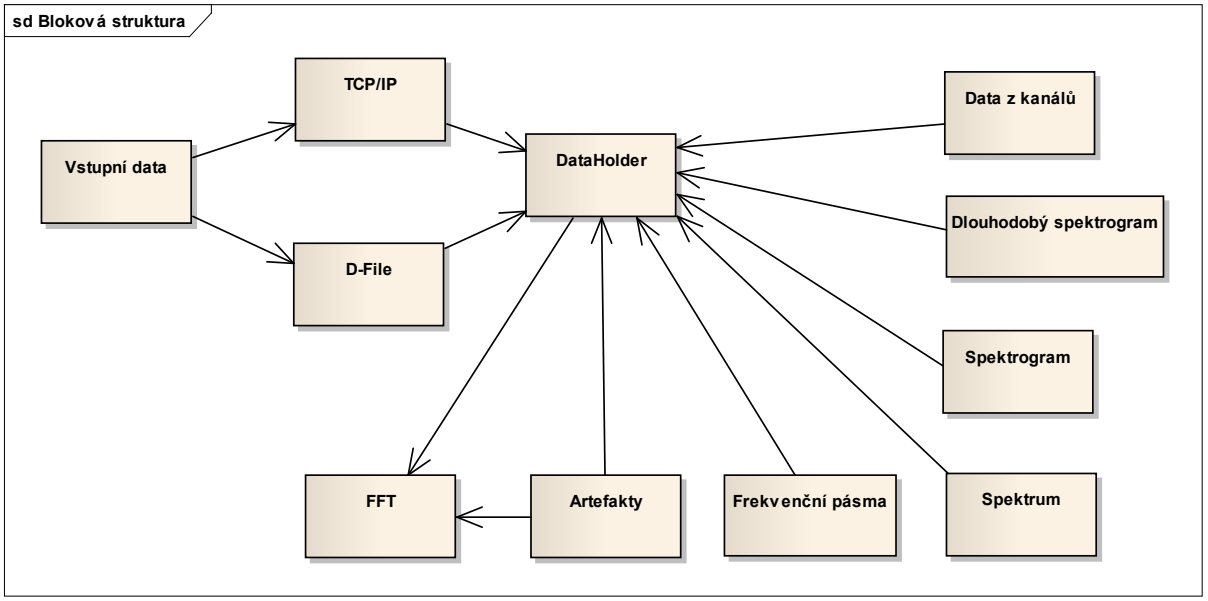
6.9.8 Nastavení aplikace

Obrázek 6.11 ukazuje vzhled okna, které slouží pro nastavení aplikace. Změna všech hodnot se musí potvrdit stisknutím tlačítka uložit. Způsob vykreslování signálu se promítne ihned do aplikace bez nutnosti potvrzení tlačítkem uložit. Pokud uživatel klikne na tlačítko zrušit, okno nastavení se zavře a žádné změny se neprovedou.



Obrázek 6.11 Nastavení aplikace

6.10 Bloková struktura programu



Obrázek 6.12 Bloková struktura programu

Z obrázku datové struktury je vidět, že stěžejním objektem pro tuto aplikaci je *DataHolder*. Ten byl implementován na základě *singleton patternu* a jsou v něm ukládána všechna společná data. Díky tomuto koncepčnímu návrhu může aplikace komunikovat s různými zdroji dat. V našem případě je v aplikaci implementována podpora pro dva různé vstupní zdroje dat: zařízení firmy Unimedis přes TCP/IP a D-file soubor.

6.11 Optimalizace kódu

Během implementace programu jsem s postupným přidáváním funkcionalit narazil na nedostatky, které způsobovaly zpomalení běhu aplikace nebo její neschopnost analyzovat celý záznam z D-file souboru. Funkcionalit přibývalo a kód se začal zvětšovat a s ním i nároky na paměť a vytížení procesoru. Proto jsem musel věnovat značnou část při implementaci aplikace optimalizaci a ladění. Zaměřil jsem se především na práci s pamětí a správnou volbu polí, které jsou pro celkový výkon a plynulý běh aplikace stěžejní. V následujícím textu bych se chtěl zaměřit na nejdůležitější aspekty při ladění aplikace. Více o optimalizaci kódu se můžeme dočíst např. v [32] nebo [33].

Synchronizace

Aby bylo dosaženo co nejrychlejšího předání dat do hlavního datového úložiště a nemuselo se čekat na vlákna, až opustí kritické bloky, byla odstraněna synchronizace sekcí na

místech, kde to není nutné nebo lze synchronizace dosáhnout jiným způsobem. Aby byla zachována konzistentnost dat, jsou pak v některých částech programu předávány kopie polí pomocí metody clone().

Double vs. Float

Dalším důležitým krokem pro minimalizaci paměťových nároků aplikace je použití primitivních typů float místo double. Double v paměti zabere 64 bitů a float jen polovinu, tedy 32 bitů. Paměťové nároky aplikace jsou jednoduchou výměnou primitivních typů poloviční. Double je vhodné použít např. v 3D grafických aplikacích, kde potřebujeme vysokou přesnost při výpočtech. V našem případě nám pro výpočty Fourierovy transformace a pro následnou klasifikaci float plně postačuje.

Načítání ze souboru

U načítání dat ze souborů obecně platí, že je lepší číst ze souboru ne tak často, ale po větších datových úsecích než po malých a častěji. Pro načítání menších datových úseků je rychlejší použití metody pomocí RandomAccessFile. Pokud načítáme veliký soubor, je pro jeho načtení rychlejší metoda pomocí buffer s využitím Chanel. Více informací o zvyšování performance při čtení ze souboru je uvedeno zde [40], srovnání jednotlivých metod je uvedeno zde [41].

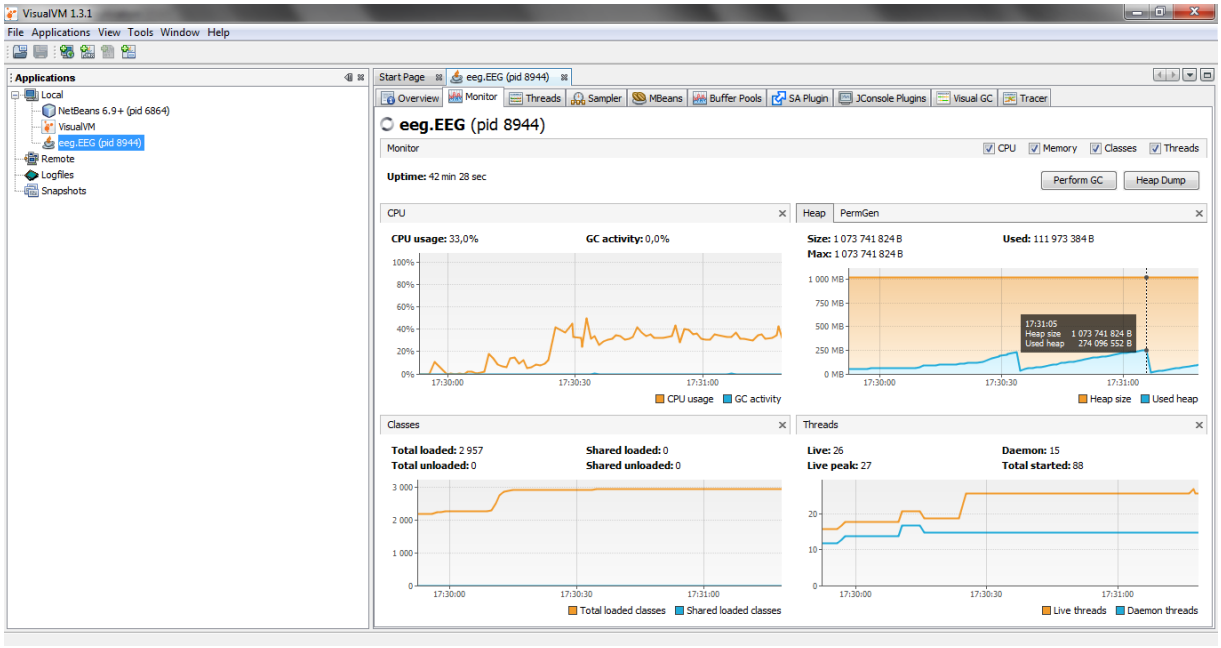
Pro optimalizaci kódu byl použit nástroj Java VisualVM (JVisualVM), který je standardně dodáván ke každému JDK, a je mu věnována následující podkapitola, kde je ukázáno, jak aplikace vytěžuje procesor a kolik paměťového místa zabírá. Tato podkapitola se také věnuje nejdůležitějším grafům a jejich významu při ladění aplikace.

6.11.1 JVisualVM

JVisualVM je nástroj, který slouží k detailnímu grafickému zobrazení informací o Java aplikacích, které aktuálně běží na Java Virtual Machine (JVM). Všechny informace jsou rozděleny do několika skupin a přehledně zobrazeny. Všechny data je také možné uložit pro další analýzu.

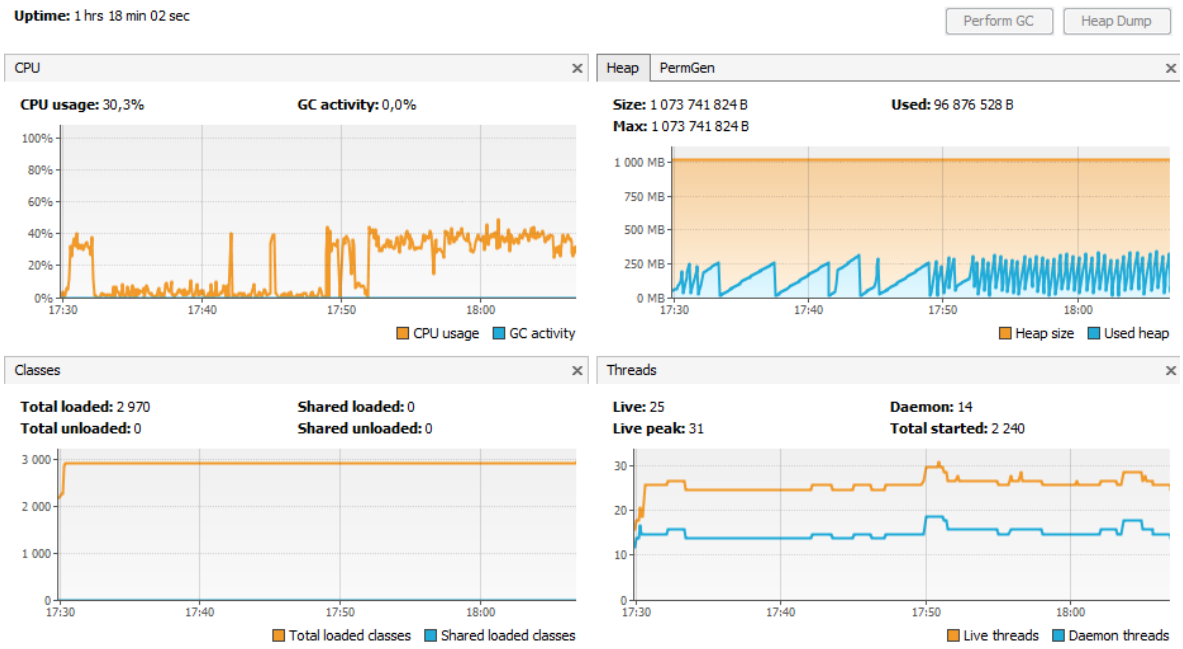
JVisualVM přehledně zobrazuje čas, který potřebují jednotlivá vlákna pro svůj běh a je také schopen zobrazit, kolik procesorového času zabírají jednotlivé metody objektů. Další možností tohoto nástroje je sledování paměťových zdrojů jak pro jednotlivé datové typy, tak i pro celou aplikaci.

Obrázek 6.13 je celkový pohled na aplikaci JVisualVM a ukazuje jaké je vytížení procesoru a kolik paměti je alokováno po 42 minutách běhu aplikace. Vytížení procesoru se pohybuje okolo 40-ti procent. Aktuální velikost využití paměti je 111MB a maximální hodnota paměti, kterou aplikace využívala, byla 274MB. V okně pro paměť jsou vidět dva vrcholy alokace paměti, která je následně minimalizována na hodnotu okolo 20MB. V této době se volá Garbage Collector (GC), který z paměti odstraní nepotřebná data. Frekvence volání GC ovlivňuje rychlost běhu aplikace, protože GC pro svoji činnost potřebuje procesorový čas. V naší aplikaci byla ponechána režie volání GC na JVM. Java verze 7 v tomto směru přináší vylepšení, které práci s GC značně urychluje.



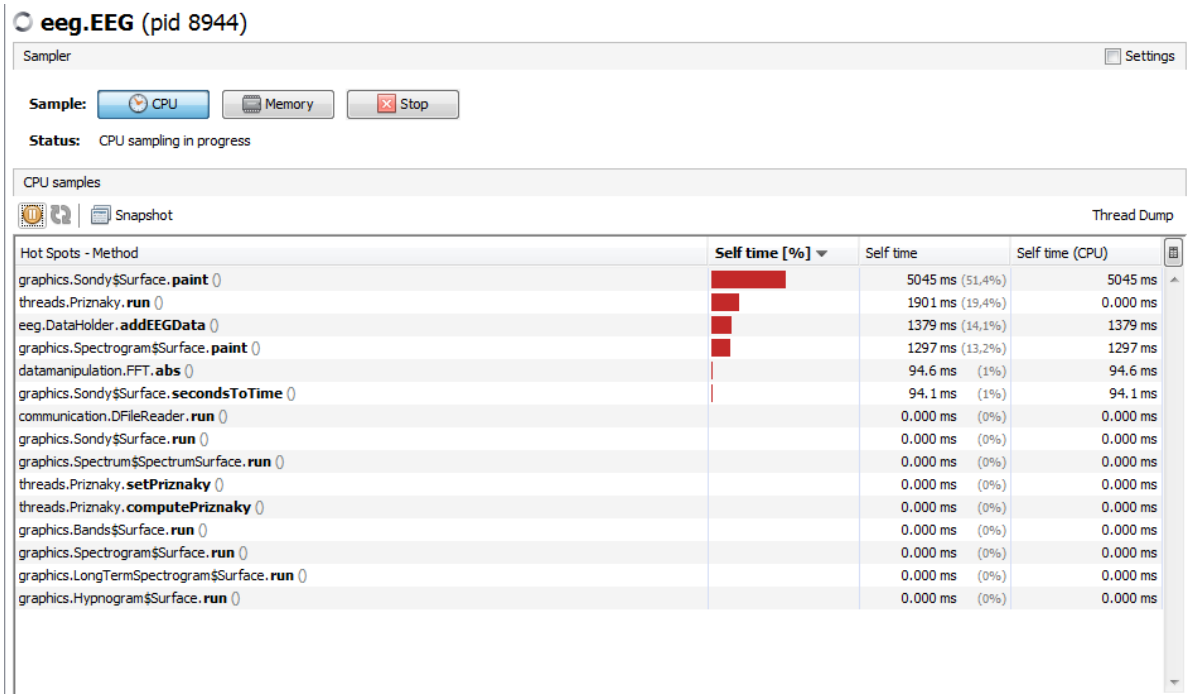
Obrázek 6.13 Alokace paměti a zatížení procesoru během 42 minut běhu aplikace

Následující Obrázek 6.14 zobrazuje stav aplikace po 78 minutách běhu. V obrázku je vidět častější volání GC a v některých okamžicích zatížení procesoru blízké nule. Maximální hodnota potřebné paměti se téměř nezměnila.



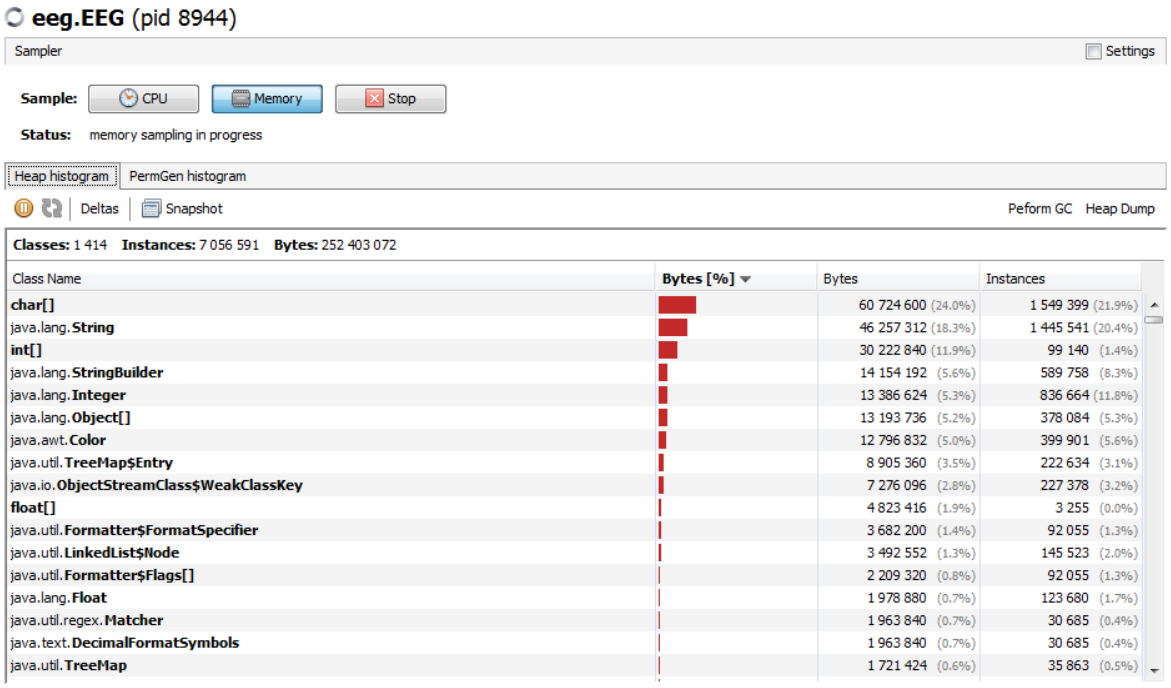
Obrázek 6.14 Alokace paměti a zatížení procesoru během 78 minut běhu aplikace

V dalším obrázku (Obrázek 6.15) je zobrazena informace o tom, kolik času potřebují jednotlivé metody pro svůj běh. Nejdéle trvající metodou naší aplikace je metoda *paint()* v třídě *Sondy*, která se stará o překreslování signálů ze vstupních dat. Při optimalizaci aplikace jsme sledovali metodu, která zabírala nejvíce času, a tu jsme se následně snažili optimalizovat tak, aby byl čas potřebný pro běh metody co nejmenší.



Obrázek 6.15 Doba vykonávání jednotlivých metod aplikace

Poslední Obrázek 6.16 zobrazuje paměťovou alokaci datových typů, které jsou v aplikaci obsaženy. Ladění aplikace spočívá v nalezení těch datových typů, které podle JVisualVM zabírají nejvíce místa. Tyto proměnné je nutné v kódu sledovat od jejich vytvoření až po jejich použití. Nejčastěji se špatné zacházení s proměnnými vyskytuje u vláken. Např. pokud je v daném vlákně definováno pole, které se používá v každém cyklu běhu vlákna, můžeme ho vytvořit pouze jednou v konstruktoru vlákna místo vytváření tohoto pole pokaždé, když je vlákno znovu spuštěno.



Obrázek 6.16 Paměťová alokace datových typů

6.12 Praktické využití aplikace

Aplikace umožňuje nahrát sadu předem definovaných artefaktů a detekovat ty části EEG záznamu, které jsou pro ošetřujícího lékaře zajímavé. Metodou nejbližšího souseda jsou pak přiřazeny segmenty do správné kategorie artefaktů. Mezi artefakty může patřit např. odpadlá elektroda, mrknutí, svalový artefakt apod. Na základě detekce některého z artefaktů může být upozorněn personál, který se stará o pacienta.

7 Závěr

Tato práce se zabývá analýzou dlouhodobých EEG záznamů v reálném čase. V této práci se mi podařilo navrhnout a implementovat program, který je schopen načítat data z několika různých zdrojů a ty následně analyzovat. Program je schopný zpracovávat a analyzovat data až z 24 kanálů. Data v reálném čase přehledně zobrazuje na obrazovku uživatele. Aplikace je navržena a odladěna tak, aby byla schopna běžet na dvou jádrových procesorech o minimální frekvenci 2,5GHz a 4GB RAM. Rozměry pro zobrazení aplikace byly zvoleny s ohledem na nejmodernější trendy a jsou optimalizovány pro rozlišení obrazovky 1920x1080 pixelů, což odpovídá standardu Full HD.

Celá aplikace je navržena modulárně tak, aby nebyla závislá na vstupním zdroji dat. V této práci jsem implementoval načítání ze dvou zdrojů data: z D-file souboru a z EEG zařízení od firmy Unimedis, které posílá data pomocí TCP/IP protokolu.

Aplikace umožnuje zpracovat celý celovečerní záznam a oklasifikovat signály z D-file souboru. V oknech aplikace jsou zobrazeny nejdůležitější trendy, které slouží pro detailnější posouzení aktuálního stavu pacienta. Všechny dlouhodobé trendy včetně klasifikace lze kdykoliv v průběhu analýzy EEG signálů exportovat jako samostatné obrázky ve formátu png pro pozdější analýzu. Funkčnost aplikace byla odzkoušena na reálných klinických datech, které poskytl Ústav pro péči o matku a dítě v Praze.

Nejvíce pozornosti bylo věnováno optimalizaci algoritmů tak, aby byly co nejefektivnější a vyžadovaly pro svůj běh minimum procesorového času a paměti. Algoritmy pro analýzu EEG byly voleny vždy jako kompromis mezi rychlostí a přesností algoritmu. Přestože aplikace neposkytuje tak přesné výsledky jako off-line metody sloužící pro analýzu dlouhodobých EEG záznamů, umožnuje ošetřujícímu lékaři poskytnout přesnější informaci o aktuálním stavu pacienta. Přesnost klasifikace závisí na trénovací množině, která je do aplikace načtena. Pro zlepšení výsledků klasifikace je potřeba načíst do aplikace trénovací množinu, která byla sestavena zkušeným neurologem.

Realizovaná aplikace může být v budoucnu rozšířena o další moduly – pokročilejší metody klasifikace, jiné typy vizualizací, podporu dalších vstupních zdrojů dat apod. Tyto moduly byly nad rámec této diplomové práce a budu je nadále vyvíjet pro firmu Unimedis a Fakultní nemocnici Na Bulovce.

8 Literatura

[1] Malmivuo, J., Plonsey, R. *Bioelectromagnetism:* Principles *and Applications of Bioelectric and Biomagnetic Fields*. New York: Oxford University Press, 1995. ISBN 0-19-505823-2.

[2] Goldensohn, E. S., Legatt A. D., Koszer, S., Wolf, S. M. *Goldensohn's EEG interpretation: Problems of Overreading and Underreading*. 2. vyd. Armong, 1999.

[3] Liu, H., Motoda, H. *Feature extraction, construction and selection: a data mining perspective*. Sinagapore: National University of Singapore, 1998. ISBN 0-7923-8196-3.

[4] Svatoš, J. *Biologické signály I: Geneze, zpracování a analýza*. 2. vyd. Praha: Vydavatelství ČVUT, 1998. ISBN 80-01-01822-9.

[5] Gevins, A. S., Rémond, A. *Handbook of electroencephalography, Volume 1: Methods of Analysis of Brain Electrical and Magnetic Signals*. Amsterdam, New York, Oxford 1987. ISBN 0-444-80125-1.

[6] Lopes, F. H., Storm, W., Rémond, A. *Handbook of electroencephalography, Volume 2: Clinical Applications of Computer Analysis of EEG and other Neurophysiological Signals*. Amsterdam, New York, Oxford, 1986. ISBN 0-444-80125.

[7] Stern, J. M., Engel, J. *Atlas of EEG Patterns*. Philadelphia, 2005. ISBN 0-7817-4124-6.

[8] Teofilo, L., Lee-Chiong. *Sleep: a comprehensive handbook*. Naboken: John Wiley & Sons, Inc., 2006.

[9] Barlow, J. S., Creutzfeldt, O. D., Michael, D., Houchin, J., Epelbaum, H. *Automatic adaptive segmentation of clinical EEG.* Electroencephalography and Clinical Neurophysiology, 1981. s. 512—525.

[10] Krajča, V., Petránek, S., Patáková, I., Värri, A. *Automatic Identification of Significant Graphoelements in Multichannel EEG Recordings by Adaptive Segmentation and Fuzzy Clustering*. Biomed Comput. 28, 1991. s. 71-89.

[11] Schlesinger, M. I., Hlaváč V. *Ten lectures on statistical and structural pattern recognition*. Praha, ČVUT, 1999.

[12] Sanei, S., Chambers, J. A., *EEG Signal Processing*. England: John Wiley & Sons Ltd., 2007. ISBN 978-0-470-02581-9.

[13] Smith, S. V. *Digital Signal Processing: A practical Guide for Engineers and Scientists*. USA: Elsevier Science, 2003. ISBN 0-750674-44-X.

[14] Ifeachor, E., Jervis, B. *Digital Signal Processing: A Practical Approach*. 2. vyd. USA: Pearson Education, Ltd., 2002. ISBN 978-0-201-59619-9.

[15] Jones, N., Watson, J. *Digital Signal Processing: principles, devices and applications*. Londýn: Peter Peregrinus, Ltd., 1990. ISBN 0-86341-210-6.

[16] Oaks, S., Wong, H. *Java Threads*. 3. vyd. USA: O'Reilly Media, Inc., 2004. ISBN 0-596-00782-5.

[17] Rieger, J. *Zpracování dlouhodobých EEG záznamů*. Diplomová práce. České vysoké učení technické v Praze, elektrotechnická fakulta, katedra kybernetiky, 2004.

[18] Gerla, V., BioDat Group (Biomedical Data Processing Group). *Pokročilé metody zpracování biologických signálů: zpracování EEG* [prezentace]. ČVUT, FEL, Katedra kybernetiky, Praha 2010.

[19] Pitner, T. *JAVA – začínáme programovat*. 1. Dotisk. Praha: GRADA, 2004. ISBN 80-247-0295-9.

[20] Hudec, B. *Programovací techniky*. Skriptum. Praha: Vydavatelství ČVUT, 2004. ISBN 80-01-02374-5.

[21] Hlaváč, V., Sedláček M. *Zpracování signálů a obrazů*. Skriptum. Praha: Vydavatelství ČVUT, 2005. ISBN 80-01-03110-1.

[22] Mařík, V., Štěpánková, O., Lažanský, J. a kol. *Umělá inteligence 1*. Praha: ACADEMIA, 1993. ISBN 80-200-0496-3.

[23] Mařík, V., Štěpánková, O., Lažanský, J. a kol. *Umělá inteligence 4*. Praha: ACADEMIA, 2003. ISBN 80-200-1044-0.

[24] Mikulec, M., Havlíček, V. *Základy teorie elektrických obvodů 2*. Praha: Vydavatelství ČVUT, 2004. ISBN 80-01-02462-8.

[25] Psutka, J., Müller, L., Matoušek, J., Radová, V. *Mluvíme s počítačem česky*. Praha: ACADEMIA, 2006. ISBN 80-200-1309-1.

[26] Neloun, M., Militký, J., Hill, M. *Počítačová analýza vícerozměrných dat v příkladech*. Praha: ACADEMIA, 2005. ISBN 80-200-1335-0.

[27] Grill, P. *EEG – Elektroencefalograife* [prezentace]. ČVUT, FEL, Katedra kybernetiky, Praha 2005.

[28] Faber, J. EEG atlas do kapsy, Praha: Triton, 1997.

[29] De Tollenaere, J., Deburchgraeve, W., De Vos, M., Van Huffel, S. *New Approaches for the Adaptive Segmentation of Neonatal EEG Signals* [online]. Dept. of Electrical Engineering (ESAT), Katholieke Universiteit Leuven, Belgium.

<ftp://ftp.esat.kuleuven.ac.be/pub/SISTA/wdeburch/reports/paper_EMBC_Twente_Joost.pdf>.

[30] Richardson, W. C., Avondolio, D., Schrager, S. Mitchell, M. W., Scanlon, J. *Professional Java JDK 6 Edition*. USA: Wiley Publishing, 2007. ISBN 978-0-471-77710-6.

[31] Bruce, E. *Thinking in Java*. 3. vyd. USA: Prentice-Hall Inc., 2003. ISBN 0-13-100287-2.

[32] Bloch, J. *Effective Java*. USA: Sixth printing, 2008. ISBN 978-0-321-35668-0.

[33] Shirazi, J. *Java Performance Tuning*. 2. vyd. USA: O'Reilly Media, Inc., 2003. ISBN 0-596-00377-3.

[34] Rieger, J. Program EEG Lab, Program byl vytvořen v rámci diplomové práce: Zpracování dlouhodobých EEG záznamů, ČVUT, Praha, Květen 2004.

[35] MATLAB [počítačový program]. Ver. 7.8.0.347(R2009a). 1984-2009. Prostředí pro maticové operace.

[36] NetBeans [počítačový program]. Ver 6.9.1. 2010. Vývojové programové prostředí pro jazyk Java.

[37] JVisualVM - Java Virtual Machine Monitoring [počítačový program]. Ver 1.3.1. Profilovací nástroj pro sledování Java aplikací.

[38] Novák, R., Gerla, V., Lhotská, L. *Analýza dlouhodobých EEG záznamů v reálném čase*. In 57. společný sjezd české a slovenské společnosti pro klinickou neurofyziologii, 12.-13. listopadu, Praha 2010. MH Consulting. Praha, MH Consulting, 2010. strana 48.

[39] AllExperts. *Java: bit representation, high order bit, memory block* [online]. Poslední revize 2007-8-26 [cit. 2011-1-4]. <http://en.allexperts.com/q/Java-1046/bit-representation.htm>.

[40] Oracle. *Articles: Tuning Java I/O Performance* [online]. Poslední revize 2010-2-16 [cit. 2011-1-4]. <http://java.sun.com/developer/technicalArticles/Programming/PerfTuning/>.

[41] Nadeau – software consulting. *Java tip: How to read files quickly* [online]. Poslední revize 2010-12-28 [cit. 2011-1-4]. <http://nadeausoftware.com/articles/2008/02/java_tip_how_read_files_quickly>.

Příloha A

CD s výslednou aplikací

Adresář	**Obsah**
/EEG4/	NetBeans projekt
/zdrojové kódy/	Zdrojové kódy aplikace
/aplikace/	Spustitelná aplikace
/D-file soubory/	D-file soubory pro analýzu
/artefakty/	Soubory s artefakty pro klasifikaci
/text/	Text diplomové práce v elektronické podobě ve formátu pdf.