GENERÁTOR KYTAROVÝCH AKORDŮ
(zápočtový program)

PROGRAMÁTORSKÁ DOKUMENTACE

1. Anotace

Mým zápočtovým programem je takzvaný generátor kytarových akordů, tedy přesněji aplikace, která k zadanému akordu vygeneruje všechny použitelné prstoklady, a to ve formě jednoduchých diagramů.

2. Trocha hudební teorie

Pro správné pochopení fungování programu je třeba mít nezbytné základy hudební teorie související se vznikem akordu.

a) intervaly

Interval je jednoduše řečeno vzdálenost dvou tónů. Měří se většinou v půltónech a označuje se počeštělými formami latinských řadových číslovek (prima, sekunda atd.). Vzhledem k tomu, že nebudeme intervaly vztahovat ke konkrétní stupnici (kde by nám stačilo označení prima - oktáva, neboť standardní stupnice má sedm tónů), přidáváme k intervalům přívlastky (např. malá, velká, čistá, zmenšená atd.). Ukažme si to na příkladu, kdy pojmenujeme intervaly mezi tónem C a ostatními tóny chromatické stupnice:

tón	půltóny	název
C	0	prima (=tentýž tón)
C#/Db	1	malá sekunda/malá nóna
D	2	velká sekunda/velká nóna/zmenšená tercie
D#/Eb	3	malá tercie
E	4	velká tercie
F	5	čistá kvarta/čistá undecima/zvětšená tercie
F#/Gb	6	zmenšená kvinta/zvětšená kvarta/zvětšená undecima
G	7	čistá kvinta
G#/Ab	8	zvětšená kvinta/malá sexta/malá tercdecima
A	9	velká sexta/velká tercdecima/zmenšená septima
B	10	malá septima
H	11	velká septima
C	12	oktáva

Jak je z tabulky vidět, pro stejné intervaly lze použít různé pojmenování, a to především v závislosti na kontextu. Kdybychom chtěli být přesní, tabulka by měla ještě pokračovat a intervaly nóna, undecima a tercdecima (česky "devítka, jedenáctka a třináctka") by měly být až nad oktávou.

b) vznik akordu

Definice akordu dle wikipedie: *"Akord je označení pro souzvuk tří a více současně znějících různých tónů, uspořádaných podle určitého systému."* Zmíněný systém se někdy označuje jako **terciový** a akord vzniká **přidáváním tercií k základnímu tónu.** Díky tomu, že máme 4 různé tercie (zmenšenou, malou, velkou a zvětšenou), existuje mnoho různých akordů odvozených od jednoho základního tónu (nicméně zmenšená a zvětšená se používají velmi zřídka).

Opět příklad: budeme tvořit akord odvozený od tónu C:

1. přidáme velkou tercii = +4 půltóny = tón E
2. přidáme malou tercii = +další 3 půltóny, tj. 7 půltónů nad C = tón G = **čistá kvinta** Nyní už máme tři různé tóny tvořící akord. Takový akord se nazývá **kvintakord.** Buď můžeme skončit nebo pokračovat v přidávání dalších tercií.
3. přidáme malou tercii = +další 3 půltóny, tj. 10 půltónů nad C = tón B = **malá septima.** Akord se septimou se nazývá **septakord.**
4. přidáme velkou tercii = +další 4 půltóny, tj. 14 půltónů nad C = tj. 2 půltóny nad C "přes oktávu" = tón D = **velká nóna.** Akord tvořený pěti tóny se nazývá **nonakord**.

Obdobně můžeme ještě přidat **undecimu** (např. čistou = tón F) a **tercdecimu** (např. velkou = tón A). Všimněme si, že vhodným přidáváním pouze malé a velké tercie dosáhneme charakteristické struktury akordu - tercie, kvinta, septima, nóna, undecima, tercdecima. Přidáváním modulovaných tercií se často může struktura změnit natolik, že vzniklý souzvuk za akord v pravém slova smyslu neoznačujeme (např. přidáním dvou zmenšených tercií k základnímu tónu C získáme tóny D (+2) a E (+4), přičemž tón E ani při nejlepší vůli nelze označit za kvintu k tónu C).

c) pojmenování akordů

V našem kontextu budeme pojmenováním rozumět přiřazení akordové značky - např. $Cmi^{9b/5\#}$. První složkou značky je základní tón = C. Dále naznačíme tercii: malá = mi (akord "cé moll"), velká = " " (akord "cé dur", neznačí se), zmenšená = sus2, zvětšená = sus4. Ostatní modulace píšeme již většinou do horního indexu. Tedy dále: čistá kvinta = " ", zmenšená = 5b nebo 5-, zvětšená = 5# nebo 5+. Malá septima (pokud je obsažena) = 7, velká septima = maj nebo Δ (běžné umístění, nikoliv horní index). Malá nóna = 9b nebo 9-, velká nóna = 9. Podobně undecima a tercdecima.

Poznámka: Pokud máme akordovou značku např. D^{13}, znamená to nejen, že v akordu je velká třináctka, ale také malá septima, velká nóna a čistá jedenáctka. Nezapomínejme, že akord byl vystavěn po terciích, tedy tyto nižší intervaly jsou v akordu automaticky obsaženy a pokud jsou nemodulované, neznačí se. (Za nemodulované intervaly se považují: velká tercie, čistá kvinta, malá septima, velká nóna, čistá undecima a velká tercdecima). Pokud bychom chtěli přidat pouze třináctku, použijeme značení D^{add13}. Takový akord pak obsahuje pouze základní kvintakord (velká tercie, čistá kvinta) a zmíněnou tercdecimu.

Další obvyklá značení: C^6 obsahuje čistou sextu, neboli zmenšenou septimu. Vznikne přidáním zmenšené tercie k čisté kvintě.

Akord Cdim vznikne přidáním postupně tří malých tercií k základnímu tónu, a obsahuje tedy malou tercii, zmenšenou kvintu a zmenšenou septimu.

Akord s velkou tercií a zvětšenou kvintou se někdy značí jako C aug ("augmented" - zvětšený)

2. Ošetření vstupu

Rozhraní programu je grafické, vstup je ošetřen formou checkboxů, comboboxů a buttonů. Rozlišíme vstup pro nastavení generátoru (komponenty DomainUpDown s názvy **capoUpDown**, **UDLadeni1 - UDLadeni6;** tlačítko **BtnDefaultTuning**) a vstup pro zadání akordu (rozbalovací boxy **comboRoot, comboBas, comboNazev, comboTercie, comboKvinta, comboSeptima, comboNona, comboUndecima, comboTercdecima**; zaškrtávací boxy **checkBox1 - checkBox12** a **checkVynechRoot;** tlačítko **buttonGeneruj**). Veškeré změny provedené součástech vstupu pro zadání akordu se projeví až po stisknutí tlačítka **buttonGeneruj**, změny provedené ve vstupu pro nastavení generátoru se projeví okamžitě.

Princip fungování vstupu: po kliknutí na tlačítko **buttonGeneruj** jsou zkontrolovány všechny ostatní komponenty vstupu a jejich aktuální hodnoty jsou použity pro vytvoření objektu typu ***Akord.*** S tím se pak dále pracuje při samotném generování.

a) pomocná třída Akord

Každá instance této třídy nese kompletní informaci o konkrétním akordu. Konstruktor vypadá následovně:

```
public Akord(List<int> tony, int root, int tercie, int kvinta, int septima, int nona, int
undecima, int tercdecima, int bas, bool lzeVynechatRoot)
```

První položkou jsou konkrétní tóny v akordu ve formě čísel 1–12 (kdy 1 = tón C, 2 = tón C#/Db ... 12 = tón H), dále základní tón (opět číslo 1–12), "přívlastky" intervalů tercie až tercdecima (např. tercie: 0 = žádná, 1 = zmenšená, 2 = malá, 3 = velká, 4 = zvětšená), basový tón (1-12) a booleanovskou informaci o tom, zda lze v prstokladu vynechat základní tón (viz. níže). Všechny proměnné jsou *private,* kromě **root** a lze**VynechatRoot** jsou přístupné pomocí *properties.*

Třída dále obsahuje konstantu MAXROZPETI, ta má význam při generování, vrátíme se k ní později.

Dále zde nalezneme několik metod, ty však slouží pro generování nebo výstup, zatím je tedy pouze zmíníme a podrobně je rozebereme později:

- VygenerujZnacky
- VygenerujPrstoklady
  - DalsiStruna
  - JeSmysluplny
  - MaxRozptyl
  - ObsahujeVseCoMa
- VypustVhodneTony
- OhodnotPrstoklady
  - ComparePrst

b) vstup pro zadání akordu

Uživatel má na výběr tři možnosti, jak zadat akord: názvem, výčtem tónů nebo popisem struktury. V každém případě musí nejprve vybrat základní tón v rozbalovacím seznamu **comboRoot** (přednastavený je tón C) a případně i bas, pokud je různý od základního tónu. Při změně **comboRoot** se automaticky mění i **basRoot** na stejnou hodnotu (tedy defaultně se počítá s tím, že bas je shodný se základním tónem). Dále uživatel zadává akord v jedné ze sekcí *název akordu, tóny v akordu* a *struktura akordu.* Mezi těmi se přepíná pomocí trojice radioButtonů (**radioNazev, radioTony, radioStruktura**), komponenty příslušné "neaktivním" sekcím se při přepínání deaktivují (pomocí vlastnosti *Enabled*).

Všechny tři sekce jsou propojené, takže změna vstupu v jedné sekci automaticky vyvolá přepsání ostatních dvou. To je realizováno událostí vyvolanou změnou v některé komponentě dané sekce. Nutnou podmínkou je, aby sekce, jejíž komponenty jsou přepisovány, nebyla aktivní. Jinak by nastal cyklus (např. změna v sekci *název akordu* vyvolá přepsání komponent sekce *struktura akordu*, což vyvolá další událost, která přepíše sekce *tóny v akordu* a *název akordu*, což opět vyvolá přepsání komponent sekce *struktura akordu*...).

i) zadání názvem akordu

Uživatel v rozbalovacím seznamu vybere požadovaný název akordu. Na výběr má z určitého množství přednastavených názvů, uložených v souboru *schemata.in.* Z toho se čte v konstruktoru formuláře **FGenerator** pomocí *StreamReaderu* (proto je v hlavičce zahrnuta knihovna *System.IO*). Každý řádek vstupního souboru obsahuje jeden akord, a to ve formátu `název;tercie;kvinta;septima;nona;undecima;tercdecima`, například `maj7;3;2;3;0;0;0`. Z každého akordu se vytvoří objekt typu ***Schema***, což je jednoduchá pomocná třída obsahující konstruktor:

```
public Schema(string nazev, int tercie, int kvinta, int septima, int nona, int undecima,
int tercdecima)
```

a odpovídající proměnné + příslušné properties. Třída neobsahuje žádné metody. Je to v podstatě velmi osekaná třída ***Akord***, která slouží jen a pouze pro uložení schémat načtených ze souboru *schemata.in.*

Načtené objekty typu ***Schema*** se uloží do *Listu* **znamaSchemata**, současně si také uložíme názvy schémat do samostatného seznamu List<string> **znamaSchemataNazvy**, kerý použijeme jako *DataSource* pro **comboNazev.**

Při změně indexu v **comboNazev** je vyvolána událost **comboNazev_SelectedIndexChanged**, která nejprve zkontroluje, zda je sekce *název akordu* aktivní, pak podle aktuálního indexu **comboNazev** vybere odpovídající schéma ze **znamaSchemata** a podle něj přepíše údaje v sekci *struktura akordu.* To vyvolá přepsání údajů i v sekci *tóny v akordu.*

ii) zadání strukturou akordu

V této sekci uživatel prostřednictvím šesti rozbalovacích comboBoxů navolí, jaké harmonické funkce jsou obsaženy v akordu. Změna indexu ve všech šesti

comboBoxech je obsluhována společnou metodou **comboStruktura_SelectedIndexChanged**. Tato metoda zavolá funkci **PrepisCheckboxy**, která přepíše data v sekci *tóny v akordu*, ovšem pouze pokud je neaktivní.

Metoda **PrepisCheckboxy** pracuje s checkboxy v sekci *tóny v akordu*, které reprezentují dvanáct tóny chromatické stupnice. Během konstruktoru **FGenerator** jsou tyto checkboxy uspořádány do pole check[0] - check[11], pro pohodlnější práci. Nyní tedy metoda **PrepisCheckboxy** celé pole "vynuluje" (tzn. nastaví vlastnost *Checked* na 'false') a následně opět nastaví checkboxy odpovídající hodnotám v komponentách **comboTercie** až **comboTercdecima** a samozřejmě i v **comboRoot**. Tedy vždy přičte k hodnotě **root** číslo odpovídající zvolenému indexu v comboBoxech v sekci *struktura akordu*, vymodulí dvanácti a checkbox s výsledným indexem zaškrtne.

Metoda **comboStruktura_SelectedIndexChanged** dále zkontroluje, zda je sekce *název akordu* neaktivní (tedy impuls změny nepřišel od ní), a pokud je, pokusí se najít k hodnotám v comboBoxech v sekci *struktura akordu* odpovídající schéma v Listu **znamaSchemata**. Postupně prochází všechny položky v seznamu **znamaSchemata** a pro každý zavolá funkci **JeZnameSchema**. Pokud funkce vrátí true, nastaví se index v **comboNazev** na odpovídající index. Pokud žádné schéma nesedí, je **comboNazev** nastaven na -1.

Metoda **JeZnameSchema** je velice jednoduchá. Na vstupu dostane objekt typu ***Schema***, jehož položky (Tercie až Tercdecima) porovná s aktuálními daty v sekci *struktura akordu* (tzn. **comboTercie** - **comboTercdecima**). Pokud se všechny shodují, vrátí true.

iii) zadání výčtem tónů

Uživatel pomocí dvanácti checkBoxů vybere tóny, které mají být v akordu obsaženy. Všechny změny obsluhuje metoda **checkBox_CheckedChanged.** Ta zkontroluje, zda je sekce *tóny v akordu* aktivní a impuls změny tedy skutečně vychází od ní. Následně zkontroluje, zda při změně v checkBoxech nebyl odznačen tón plnící funkci základního tónu = rootu. Pokud ano, vybere z checkBoxů nejnižší zaškrtnutý a označí jej za nový root. Při neúspěchu je za root dosazena hodnota -1.

Následně je zavolána funkce **SestavSchema.** Ta se pokusí sestavit z tónů zadaných v sekci *tóny v akordu* smysluplné akordové schéma. Vrací objekt typu ***Schema***, případně null, pokud neuspěje. Funkce nejprve vytvoří objekt typu ***Schema*** s názvem *sestavovaneSchema,* do kterého se bude snažit doplnit harmonické funkce. Funkce postupně zkouší pro každou harmonickou funkci (tercie až tercdecima) najít a přiřadit jeden z tónů, přičemž zkouší všechny možné modulace (tj. např. malou, velkou, zmenšenou i zvětšenou tercii). Pořadí, v jakém se zkouší jednotlivé funkce a jejich modulace je velmi důležité, neboť může docházet ke "kolizím", kdy jeden tón může teoreticky plnit zároveň více než jednu funkci (např. pro základní tón C může být tón D zmenšenou tercií, ale také velkou nónou).

Kolize ukazuje následující tabulka:

Tón	C	C#	D	D#	E	F	F#	G	G#	A	B	H
Půltóny	0	1	2	3	4	5	6	7	8	9	10	11
tercie			--	-	+	++						
kvinta							--	o	++			
septima										--	-	+
nóna		-	+									
undecima						o	++					
tercdecima									-	+		

Tabulka potenciálních harmonických funkcí jednotlivých tónů vzhledem k základnímu tónu C.
Vysvětlivky: **--** *zmenšená;* **-** *malá;* **o** *čistá,* **+** *velká,* **++** *zvětšená*

Proto se funkce testují v pořadí: tercie, kvinta, septima, nóna, undecima, tercdecima. U každé funkce se nejprve testují nemodulované variace (čistá, malá, velká) a teprve pokud tyto neuspějí, zkouší se modulované (zmenšená, zvětšená).

Ukažme si fungování funkce na příkladu. Mějme základní tón C a tóny obsažené v akordu jsou C, D, E, G#, A. Funkce proběhne následovně:

1. tercie
    1. malá = tón D# - není obsažen
    2. **velká = tón E - je obsažen, sestavovaneSchema.Tercie = 3, break**
2. kvinta
    1. čistá = tón G - není obsažen
    2. zmenšená = tón F# - není obsažen
    3. **zvětšená = tón G# - je obsažen, sestavovaneSchema.Kvinta = 3, break**
3. septima
    1. malá = tón B - není obsažen
    2. velká = tón H - není obsažen
    3. **zmenšená = tón A - je obsažen, sestavovaneSchema.Septima = 1, break**
4. nóna
    1. **velká = tón D - je obsažen & neplní funkci tercie (sestavovaneSchema.Tercie > 1), sestavovaneSchema.Nona = 2, break**
5. undecima
    1. čistá = tón F - není obsažen
    2. zvětšená = tón F# - není obsažen
6. tercdecima
    1. velká = tón A - je obsažen, ale plní funkci septimy
    2. malá = tón G# - je obsažen, ale plní funkci kvinty

Výsledkem je tedy akord s velkou tercií, zvětšenou kvintou, zmenšenou septimou a velkou nónou – $C^{9/6/5\#}$.

Na příkladu je vidět, že kdyby se například tercdecima zkoumala dříve než septima, byla by tónu A přiřazena jiná (nesprávná) harmonická funkce.

Pro každý přiřazený tón inkrementujeme pomocnou proměnnou **nalezenych**, kterou potom porovnáme s počtem zadaných tónů (v cyklu spočteme zaškrtnuté checkBoxy) v proměnné **zadanych**. Pokud se rovnají, funkce vrátí **sestavovaneSchema**, pokud ne, vrátí *null*, neboť to znamená, že pro nějaký tón se

nepodařilo najít harmonickou funkci. Funkce **checkBox_CheckedChanged** hodnoty objektu typu ***Schema*** vráceného funkcí **SestavSchema** dosadí do sekce *struktura akordu* (v případě *nullu* je vynuluje).

c) vstup pro nastavení generátoru

Skupina komponent ovlivňující ladění a umístění kapodastru.

i) ladění

Komponenty DomainUpDown s názvy **UDLadeni1 – UDLadeni6** jsou obsluhovány funkcí **UDLadeni_SelectedItemChanged.** Ta mimo jiné zajišťuje "cykličnost" přepínačů. Samotné tóny jsou na indexových pozicích 1 - 12, nicméně seznam obsahuje i pozice 0 a 13. Jakmile uživatel nalistuje pozici 0, funkce automaticky změní index na 12. Analogicky pro 13 změní index na 1.

Funkce dále zkontroluje stav globální proměnné **kresliSe**, která je true, pokud už bylo uživatelem kliknuto na tlačítko **buttonGeneruj** a byl vykreslen první diagram. Pokud má **kresliSe** hodnotu true, funkce **UDLadeni_SelectedItemChanged** "klikne" na tlačítko **buttonGeneruj** a způsobí okamžité překreslení diagramu (již se změněným laděním). Pokud je **kresliSe** false, nic se nestane.

Tlačítko **BtnDefaultTuning** nastaví komponenty **UDLadeni1 – UDLadeni6** na hodnoty standardního ladění EADGHe.

ii) kapodastr

Změny komponenty **capoUpDown** ošetřuje funkce **capoUpDown_ValueChanged**, která po kontrole proměnné **kresliSe** klikne na **buttonGeneruj** a překreslí diagram s nově umístěným kapodastrem.

3. Generování prstokladů

Proces generování začíná kliknutím na tlačítko **buttonGeneruj**, tedy vyvoláním funkce **buttonGeneruj_Click.**

a) statická třída Hmatnik

Tato statická třída implementuje abstraktní konstrukci kytarového hmatníku. Obsahuje privátní konstanty **MAXPRAZEC** (nejvyšší pražec, do kterého se hledají vhodné pozice - nastaveno na 15), **POCETSTRUN** (nastaveno na 6) + veřejnou property **PocetStrun** s funkcí *get*, dále privátní statické pole integerů **ladeni** (defaultně nastaveno standardní ladění EADGHe) a s ním související metody **GetLadeni(int index)** a **SetLadeni(int index, int value).** Poslední metodou této třídy je funkce **NajdiTonyNaHmatniku(List<int> tony)**, jejíž návratovou hodnotou je **List<List<int>>** - přesněji šestice Listů integerů, které reprezentují čísla pražců obsahující zadané tóny pro každou ze šesti strun. Těžištěm funkce je vnořený cyklus (0 až POCETSTRUN, 0 až MAXPRAZEC), který postupně otestuje každý tón na hmatníku - nejprve z čísla aktuálního pražce a ladění na aktuální struně spočte, o jaký tón se jedná, a pak se podívá, jestli je tento tón obsažen ve vstupním parametru **List<int> tony**. Pakliže ano, do pomocné proměnné **List<List<int>> tonyNaHmatniku** do vnitřního listu s indexem aktuální struny přidá číslo aktuálního pražce.

Pro první dvě basové struny (E a A ve standardním ladění) je do vnitřního seznamu ještě přidána hodnota -1, což reprezentuje zatlumenou (=nehranou) strunu. Tedy pro první dvě struny je povoleno zatlumení explicitně, u ostatních jen pokud neexistuje hratelná pozice (což je ošetřeno během generování prstokladů).

b) funkce buttonGeneruj_Click

Celá funkce je podmíněna tím, že **root** > -1, tedy že je zvolen základní tón.

Nejprve je z komponent **UDLadeni1 - UDLadeni6** dosazeny hodnoty ladění do **Hmatnik.ladeni** pomocí funkce **Hmatnik.SetLadeni**, přičemž je ke každé struně přičtena hodnota komponenty **capoUpDown** (umístění kapodastru na n-té pozici zvýší ladění o n půltónů).

Dále je vytvořena instance třídy ***Akord*** s názvem **aktualniAkord**. Do ní je ze vstupních komponent dosazen **root, tercie, kvinta, septima, nona, undecima, tercdecima, bas a lzeVynechatRoot**, a to přímo v konstruktoru. Seznam tónů je zatím vytvořen prázdný, dosazujeme do něj až nyní. Jednoduše v cyklu projdeme checkBoxy a tóny odpovídající těm zaškrtnutým přidáme do seznamu.

V další fázi je na instanci **aktualniAkord** zavolána metoda **VygenerujPrstoklady**

c) generování

Nejprve idea: Funkce má k dispozici seznam tónů v akordu. Nejprve zjistí jejich pozice na hmatníku. Pak zafixuje první pozici na první struně a pokračuje na další strunu. Na té opět zafixuje první pozici a pokračuje dál, postupně až na šestou strunu, čímž je dokončen prstoklad. Zkontroluje se, zda je smysluplný a hratelný, pokud ano, je zařazen do výstupního seznamu. Poté funkce zkusí druhou pozici na šesté struně a opět zkontroluje a případně zařadí. Takto pokračuje, dokud nedojdou pozice na šesté struně. Pak se vrátí o strunu zpět, tam zafixuje další, ještě neprozkoumanou pozici a opět postoupí na šestou strunu, kterou prochází opět od začátku. Tedy procházíme do hloubky strom (resp. přesněji les), jehož listy tvoří kompletní prstoklady a vnitřní uzly prstoklady částečné. Kromě kontroly po nalezení prstokladu provádíme ještě testy během procházení stromu a průběžně jej prořezáváme (například procházet větve vedoucí z částečného prstokladu, kde tiskneme první strunu na třetím a druhou na čtrnáctém pražci, nemá smysl).

Technická realizace: Proces zahajuje "zaváděcí" funkce **VygenerujPrstoklady**, která inicializuje potřebné proměnné pro rekurzivně volanou funkci **DalsiStruna**, která odvádí nejdůležitější práci. Metoda nejprve vygeneruje pozice tónů na hmatníku, a to pomocí k tomu určené metody statické třídy ***Hmatnik*** s názvem **NajdiTonyNaHmatniku**, které je předán jako argument seznam tónů v aktuálním akordu (this.Tony). Kromě toho však také vygeneruje pozice basového tónu na hmatníku, protože ten je třeba hledat jako první. Až po jeho úspěšném nalezení může funkce hledat ostatní tóny obsažené v akordu. Dostáváme tedy dva seznamy List<List<int>> **tonyNaHmatniku** a **basyNaHmatniku.**

Dále inicializujeme proměnné, které se budou ve funkci DalsiStruna rekurzivně předávat. Jsou to: int[] **aktualniPrstoklad** - pole šesti integerů, ve kterém se předává rozpracovaný prstoklad; List<int[]> **prstoklady** - seznam šestic integerů reprezentujících doposud nalezené prstoklady; int **basNaStrune** - proměnná indikující, na které struně byl nalezen basový tón. Tato proměnná pomáhá funkci DalsiStruna při "přepínání" mezi

hledáním všech tónů a hledáním pouze basů. Poté je zavolána funkce DalsiStruna, která vygeneruje vhodné prstoklady a uloží je do seznamu List<int[]> **prstoklady**, který potom funkce VygenerujPrstoklady vrací jako svůj výstup.

Funkce **DalsiStruna** je volána s těmito parametry:

```
private void DalsiStruna(ref List<List<int>> tonyNaHmatniku, ref List<List<int>>
basyNaHmatniku, ref int basNaStrune, ref List<int[]> prstoklady, int aktualniStruna, ref
int[] prstoklad)
```

Funkce je nejprve volána s předem inicializovanými proměnnými, aktualniStruna je rovna 0 (stejně tak jako **basNaStrune**). Metoda obsahuje lokální proměnnou List<List<int>> **naHmatniku**, do které se v každém volání dosadí buď **tonyNaHmatniku**, nebo **basyNaHmatniku**, což funkce pozná podle proměnné **basNaStrune**. Je-li rovna proměnné **aktualniStruna**, znamená to, že jsme ve fázi hledání basového tónu. Je-li nižší, bas už jsme našli na nižší struně a nyní jsme hlouběji v rekurzi a hledáme ostatní tóny.

V každém případě procházíme pozice v seznamu **naHmatniku[aktualniStruna]**, přičemž přidáme jeden průchod cyklem navíc. Tento poslední průchod je kontrolní, neboť při procházení si ještě udržujeme lokální booleanovskou proměnnou **nasel,** která indikuje, zda byla na aktuální struně nalezena aspoň jedna použitelná pozice. Pokud ne - a **nasel** je tedy po všech průchodech stále false - je v závěrečném kontrolním průchodu dosazena do prsokladu na index aktuální struny hodnota -1, tedy zatlumení. (Tedy: pokud na struně neexistuje použitelná pozice, je povoleno její zatlumení.) V případě, že jsme ve fázi hledání basového tónu (tedy **basNaStrune** == **aktualniStruna**), je třeba ještě přesunout hledání basu na další strunu (**basNaStrune** = **aktualniStruna** + 1).

Pokud se nejedná o poslední kontrolní průchod, dosadíme vybraný tón do prstokladu. Vzhledem k tomu, že funkce **NajdiTonyNaHmatniku** přidá do seznamu pozic pro první dvě struny i hodnotu -1 (tedy zatlumení), je třeba opět provést kontrolu, zda neprobíhá hledání basu a případně jej opět deklarovat o strunu výše.

Nyní je třeba zkontrolovat smysluplnost částečného prstokladu po přidání nového tónu. O to se stará funkce bool **JeSmysluplny**(int[] akord, int aktualniStruna) (viz níže). Pokud je test v pořádku, zkontrolujeme, na které jsme aktuálně struně. Jestliže nejsme na poslední, zavoláme rekurzivně funkci **DalsiStruna** pro **aktualniStruna + 1.** Jestliže jsme na poslední struně a tedy jsme zkompletovali prstoklad, zavoláme druhou testovací funkci s názvem **ObsahujeVseCoMa** a validní prstoklad přidáme do seznamu **prstoklady**.

Funkce **JeSmysluplny** testuje částečný prstoklad. Ten získáme z parametru **prstoklad**, z něhož vybereme prvních x tónů - x je dáno druhým parametrem **aktualniStruna** - a vytvoříme z nich pomocnou proměnnou **int[] KPorovnani**. Následují jednotlivé podmínky. Zatím jedinou implementovanou podmínkou je test na rozpětí prstokladu. (V úvahu přichází i další testy, avšak prořezávání, které by způsobily, je oproti skutečně velice efektivní podmínce na rozpětí prstokladu zanedbatelné.) Test na rozpětí (tedy nejnižší a nejvyšší pražec, na kterém se prstoklad drží) je realizován funkci **MaxRozptyl**, která je posléze porovnána s konstantou třídy *Akord* **MAXROZPETI**.

Funkce **MaxRozptyl** vrací maximální vzdálenost dvou tónů v prstokladu, tedy maximální počet pražců, přes které by bylo třeba roztáhnout prsty. Princip je prostý. Nejprve z prstokladu eliminujeme prázdné a zatlumené struny (ty se nechytají, takže na

rozpětí nemají vliv) a ze zbylých pak vybereme minimum a maximum, jejichž rozdíl je požadovaným výsledkem.

Funkce **ObahujeVseCoMa** kontroluje *kompletní* prstoklad před jeho zařazením do výstupního seznamu.

Zde je třeba zmínit jednu věc. Původní myšlenkou bylo, že před zařazením nového prstokladu se zkontroluje jednak tónová kompletnost, ale také hratelnost prstokladu (tedy jestli může daný prstoklad chytnout obyčejný smrtelník pěti prsty). První složka je implementována v této funkci, od té druhé jsem nakonec upustil. Otázka, který prstoklad ještě jde zahrát a který již ne, je totiž velice subjektivní a závisí i na fyzických dispozicích kytaristy. Někde pomohou dlouhé prsty, někde vhodně chycené baré (prst položený přes více strun) - viděl jsem dokonce kytaristy, kteří zmáčknou špičkou jednoho prstu dvě sousední struny. Proto je testování hratelnosti nastaveno velice tolerantně a na výstupu se tak mohou objevit i velice krkolomné prstoklady. Zde je tedy na každém kytaristovi, aby si vybral ten, na které jeho možnosti stačí. Obratnější hráči tak na druhou stranu nejsou ochuzeni o některé exotické a nezvyklé prstoklady.

Nejprve jsou z aktuálního prstokladu vyextrahovány v něm obsažené tóny do proměnné **tonyVPrstokladu.** Tóny se vypočtou z aktuálního ladění a pozic v prstokladu na jednotlivých strunách. Duplicitní tóny se nezařazují vícekrát. Pak se tento vzniklý seznam porovná se seznamem tónů obsažených v aktuálním akordu (this.Tony). Pokud žádný nechybí, funkce vrátí true.

Před porovnáním seznamů je třeba ještě ošetřit případ "velkých akordů", jako jsou například tercdecimové akordy ("třináctky"). Například akord C^{13} obsahuje tóny C, E, G, B, D, F a A; celkem tedy sedm tónů. Což je na šestistrunnou kytaru docela velké sousto. Proto se na akordy s velkým počtem tónů nejprve volá funkce **VypustVhodneTony.** Ta v souladu s hudební teorií určí tóny, jejichž funkce v akordu není až tak významná a mohou být z prstokladu vypuštěny. Kromě toho také může vypustit základní tón, pokud toto uživatel povolí zaškrtnutím checkBoxu **checkVynechRoot.** Toto je častá praxe v jazzových uskupeních, kde základní tón často hraje basa či basová kytara a kytarista tedy může starost o něj přenechat basistovi a zaměřit se na vyšší harmonie.

d) ohodnocení prstokladů

Nyní tedy máme seznam vhodných prstokladů. Před jejich vypsáním/vykreslením je však ještě vhodné je setřídit. Jak už bylo zmíněno v předchozí sekci, generátor je nastaven velmi tolerantně, my však chceme, aby byly ty krkolomnější a exotičtější prstoklady zařazeny až na konec a na prvních místech aby byly naopak prstoklady pohodlnější a jednodušší. Nejprve tedy prstoklady ohodnotíme podle několika kritérií, a poté je vhodně seřadíme.

i) třída *Prstoklad*

Tato jednoduchá pomocná třída obsahuje dvě proměnné: int[] **prazceVPrstokladu** - tedy samotná šestice pozic na jednotlivých strunách; a int **penalizace** - celočíselná hodnota udržující hodnocení prstokladu. Penalizace má negativní charakter - tedy čím vyšší penalizace, tím hůře. Dále už ve třídě najdeme jen odpovídající properties a konstruktor.

ii) funkce OhodnotPrstoklady

Vstupním parametrem je seznam List<int[]> **vhodnePrstoklady**, výstup je ve stejném formátu. Funkce inicializuje lokální proměnnou **List<Prstoklad> prstokladySHodnocenim,** tedy seznam prstokladů - tentokrát už i s hodnocením. Poté prochází položky v seznamu **vhodnePrstoklady**, každý prstoklad ohodnotí a přidá do seznamu **prstokladySHodnocenim.**

Kritéria hodnocení jsou tři: **počet vynechaných tónů, pozice na hmatníku a počet zatlumených strun** (v pořadí od nejzávažnějšího). Počet vynechaných tónů se vztahuje na "velké akordy", na které byla volána funkce **VypustVhodneTony**. Čím méně jich bylo vypuštěno, tím lépe. Každý vynechaný tón má váhu sta "trestných bodů". Pozice na hmatníku je reprezentována pozicí nejvýše položeného pražce v prstokladu. Čím níže je akord umístěn, tím pohodlněji se hraje. Váha kritéria je nastavena takto: *penalizace += prstoklad.Max() * 10*. Posledním kritériem je počet zatlumených not. Za každou -1 v prstokladu je jeden trestný bod, neboť prstoklady, kde se tlumí struny, nemají tak bohatý a hutný zvuk.

Prstoklady v seznamu **prstokladySHodnocenim** nyní setřídíme podle penalizace (pomůžeme si uživatelsky definovanou porovnávací funkci typu Comparison<Prstoklad>. Do výstupního seznamu **List<int[]> serazenePrstoklady** zkopírujeme už opět pouze šestice čísel (**prazceVPrstokladu**), penalizaci už nepotřebujeme.

4. Výpis

Setříděné prstoklady máme nyní v proměnné **ohodnocenePrstoklady**, před zavoláním funkce **Vypis()** si ještě vygenerujeme harmonické značky pro jednotlivé tóny vzhledem k aktuálnímu akordu. Budeme je později vypisovat pod diagram k jednotlivým tónům.

a) harmonické značky

Funkce **VygenerujZnacky** je metodou třídy ***Akord.*** Pro každý tón od C až po H vygeneruje "značku" - tedy zkratku pro jeho harmonickou funkci v akordu (případně prázdný řetězec, pokud v akordu není). Značky se ukládají do pole dvanácti stringů **znacky.** Základní tón má značku '**R**' (root), basový tón **(B)** - pokud neplní jinou harmonickou funkci v akordu. Ostatní tóny jsou značeny např. '**5**' (kvinta), '**11#**' (zvětšená undecima), '**m3**' (malá tercie) apod. Při přiřazování značek opět záleží na pořadí, neboť v akordu může zase docházet ke kolizím. Např. akord může obsahovat zároveň velkou nónu a zmenšenou tercii - což je ve skutečnosti tentýž tón. (Pozn.: Zadávat akord takto samozřejmě nedává smysl, nicméně teoreticky to možné je.) V takovém případě chceme, aby značka pro tercii přepsala značku pro nónu. Proto harmonické funkce vyhodnocujeme v pořadí: nóna, undecima, tercdecima; tercie, kvinta, septima. Tedy intervaly "přes oktávu" jsou označovány jako první, intervaly "pod oktávou" až poté, takže je mohou přepsat.

b) funkce Vypis

Funkce převede prstoklady v seznamu **ohodnocenePrstoklady** do textové podoby a vypíše je do ListBoxu **listOut.** První prstoklad vykreslí do diagramu. Textová reprezentace prstokladu vypadá např. následovně: `X | X | 0 | 2 | 3 | 2` . Znamená to:

struny E a A nehrát/tlumit, strunu D nechat znít prázdnou, strunu G držet na druhém pražci, strunu H na třetím a nejvyšší strunu E opět na druhém.

Nejprve je deklarována lokální proměnná **List<string> items**, do které budeme přidávat textové prstoklady. Následně v cyklu procházíme **ohodnocenePrstoklady** a ve *StringBuilderu* **item** vytváříme textovou reprezentaci každého jednotlivého prstokladu. Pro každý prstoklad procházíme struny od nejnižší a pomocí funkce Append připojíme odpovídající číslo pražce do proměnné **item.** Místo -1 připojíme X. Tóny oddělujeme znakem '|' a vhodným počtem mezer. Hotový prstoklad v proměnné **item** převedeme na string a přidáme do seznamu **items.** Po dokončení cyklu celý seznam přiřadíme jako *DataSource* pro komponentu **listOut.** Na závěr funkce **Vypis** zavolá funkci **PrekresliDiagram** na první prstoklad.

V seznamu prstokladů **listOut** lze po vygenerování listovat pomocí tlačítek **buttonPredchozi** a **buttonDalsi**. Zešednutí tlačítek na začátku a na konci seznamu zařídí fukce **listOut_SelectedIndexChanged,** která detekuje přelistování na první či poslední položku v seznamu a příslušné tlačítko zneaktivní. Při listování seznamem je pokaždé opětovně volána funkce **PrekresliDiagram.**

c) funkce PrekresliDiagram

Funkce vykreslí diagram do objektu **pictureBox1** a vyplní popisky pod diagramem - pod každou hranou strunou je popisek s názvem znějícího tónu a jeho harmonickou funkcí v akordu. Mimo to také přepne indikátor **kresliSe** na true.

i) diagram

Pro vykreslení diagramu použijeme čtyři externí obrázky: **background.png, dot.bmp, cross.bmp** a **emptydot.bmp**. Kreslíme na objekt **pictureBox1**, na němž tedy vytvoříme objekt **grafika** třídy ***Graphics*** pomocí funkce **CreateGraphics**. Nejprve vykreslíme pozadí metodou **DrawImage**. Pak se vyrovnáme s omezením kreslící plochy - konkrétně s faktem, že diagram má pouze pět pražců. Pro výše umístěné prstoklady se zobrazení posouvá a pouze vedle diagramu upravíme popisek s informací, od kterého pražce je diagram vykreslen. Technicky to provedeme takto: vybereme nejvýše umístěnou pozici z prstokladu. Pokud je větší než 5, nastává posun. Nyní ze seznamu všech pozic v prstokladu opět smažeme prázdné či zatlumené noty (neboť ty se vykreslují symbolicky nad diagram) a ze zbylých vybereme minimum. Od tohoto pražce se tedy bude vykreslovat. Tuto informaci udržujeme v proměnné **prazec** a rovněž ji vypíšeme do popisku vedle diagramu.

Nyní vykreslíme samotný prstoklad formou koleček a křížků. Procházíme prstoklad po strunách a podle pozice vykreslíme: **křížek** pro zatlumenou strunu, **prázdné kolečko** pro otevřenou strunu a **plné kolečko** pro tisknutou strunu. Křížek a prázdné kolečko kreslíme nad diagram, plné kolečko na odpovídající pozici v obrázku. Tu vypočteme z indexu aktuálně zpracovávané struny, čísla pražce v prstokladu a posunu zobrazení uloženého v proměnné **prazec.** Pokud vhodně zvolíme číselné konstanty, tečky se vykreslí přesně na místa na hmatníku, která se mají držet.

ii) popisky

Pro každou strunu vypočteme z ladění a tisknutého pražce aktuálně hraný tón. Ten je ve formátu čísla od 1 do 12. Skutečné názvy tónů udržujeme ve statické třídě ***Tony*** - ta obsahuje pouze pole dvanácti stringů odpovídajících jednotlivým tónům a metodu **VratNazev(int index)**, která "převádí" vstupní celočíselný formát tónu na jeho název. Pomocí této funkce tedy získáme název aktuálně znějícího tónu a ten doplníme do popisku pod aktuální strunou. Do popisku ve druhém řádku doplníme harmonickou funkci tónu, kterou už jsme si předem vygenerovali a je uložena v poli **znacky**.

5. Závěr

Program je vystavěn s ohledem na co největší modularitu a s tím související možnost pozdějších rozšíření. Nabízí se například implementování více kritérií třídění prstokladů, více podmínek pro hratelnost, rozšíření výstupu o dodatečné informace o akordu a použitelných stupnicích, uživatelské hodnocení prstokladů ("oblíbené" prstoklady pro daný akord), multimediální modul, který by dokázal akord přehrát, a mnohé další.