FUNDAÇÃO EURÍPIDES SOARES DA ROCHA
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

YUJI MATUNOBU

DESENVOLVIMENTO DE SOFTWARE EDUCATIVO PARA TREINAMENTO EM PERCEPÇÃO MUSICAL

MARÍLIA
2010

YUJI MATUNOBU

DESENVOLVIMENTO DE SOFTWARE EDUCATIVO PARA TREINAMENTO EM PERCEPÇÃO MUSICAL

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profª. Drª. JULIANA DE OLIVEIRA

MARÍLIA
2010

Matunobu, Yuji
Desenvolvimento de Software Educativo para Treinamento em Percepção Musical / Yuji Matunobu; orientadora: Juliana de Oliveira. Marília, SP: [s.n.], 2010.
80 f.

Trabalho de Conclusão de Curso - Curso de Bacharelado em Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – Univem, Marília, 2010.

1. Software Educativo 2. Percepção Musical

CDD: 005.12

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Yuji Matunobu

DESENVOLVIMENTO DE SOFTWARE EDUCATIVO PARA TREINAMENTO EM PERCEPÇÃO MUSICAL

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10,0 (Dez)

Orientador: Juliana de Oliveira

1°. Examinador: Ildeberto de Gênova Bugatti

2°. Examinador: Renata Aparecida de Carvalho Paschoal

Marília, 02 de dezembro de 2010.

A Deus-Parens e Oyassama, pela proteção recebida diariamente.

Aos meus pais, Paulo e Yuriko, por 24 anos guiando meus passos.

Aos meus irmãos, Mie, Kiyo, Akio, Tami e Lie por estarem juntos nesta caminhada.

A minha namorada, Andressa, por entrar em minha vida.

AGRADECIMENTOS

Agradeço à minha família, por sempre me apoiar e incentivar, suprindo a minha ausência nas obrigações durante todo este período.

À minha namorada, por ter diminuído a exigência de 100% do meu tempo para 99, 99%, pelo enorme e lindo sorriso que sempre me incentiva, e por estar sempre presente, não necessariamente calma, mas sempre compreensiva.

À Associação Itiretsu-kai, pela bolsa de estudos concedida, viabilizando o desejo de concluir o curso de nível superior.

À minha sábia orientadora, Juliana de Oliveira, pela constante cobrança, cobrança e cobrança, sempre me incentivando e mostrando o caminho em momentos de dificuldades.

Ao professor Fábio Lúcio Meira, pelo material disponibilizado e por sempre responder atenciosamente as minhas dúvidas.

Ao professor Leonardo Castro Botega, por cobrar relatórios e listas de freqüência, tornando o difícil um pouco mais difícil.

À professora de música e amiga, Cláudia Helena Renó Jorge, por virar meu projeto de pernas para o ar, bagunçando todos os requisitos e assim tornar possível a concretização deste projeto, com qualidade.

Ao colega Lucas Medeiros, oráculo, por encontrar respostas que não constam em livros.

Aos meus amigos, que nada fizeram pelo projeto, mas muito fazem pela minha vida.

Quão penoso e desagradável
seja o trabalho,
se fizer julgando-o excelente,
esta razão alcançará o céu.
Ep. - 144

MATUNOBU, Yuji. **Desenvolvimento de Software Educativo para Treinamento em Percepção Musical.** 2010. 80 f. Trabalho de Conclusão de Curso de Bacharelado em Ciência da Computação – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2010.

RESUMO

Para os músicos, a percepção musical é indispensável e seu treinamento requer tempo e dedicação. Porém, pouco se pode exigir das aulas de música em escolas particulares, visto que seu tempo é limitado e saturado de técnicas práticas. Este trabalho tem como objetivo desenvolver um software educativo para que o aluno de música possa realizar exercícios de percepção musical em qualquer computador, porém, sempre com o acompanhamento do professor de música, que irá criar exercícios e acompanhar o desempenho do aluno.

Palavras-chave: Software Educativo. Percepção Musical.

MATUNOBU, Yuji. **Desenvolvimento de Software Educativo para Treinamento em Percepção Musical.** 2010. 80 f. Trabalho de Conclusão de Curso de Bacharelado em Ciência da Computação – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2010.

ABSTRACT

For musicians, music perception is essential and their training requires time and dedication. However, little can be required of music lessons in private schools, since their time is limited and saturated with practical techniques. This work aims to develop educational software for the student of music can make exercises on any computer music perception, but always with the accompaniment of music teacher, who will create workouts and track student performance.

Keywords: Educational Software. Music Perception.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BPM: Batidas Por Minuto

CAI: Computer Assisted Instruction

GUI: Graphical User Interface

IA: Inteligência Artificial

ICAI: Intelligent Computer Assisted Instruction

IDE: Integrated Development Environment

IE: Informática na Educação

ITS: Intelligent Tutoring System

JRE: Java Runtime Environment

MIDI: Musical Instrument Digital Interface

OO: Orientação a Objetos

SGBD: Sistema Gerenciador de Banco de Dados

UML: Unified Modeling Language

RUP: Rational Unified Process

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

A capacidade de identificar notas musicais é de extrema importância para os músicos, pois ajuda na musicalidade do indivíduo. Desenvolvendo a percepção musical, a audição pode se tornar tão apurada a ponto de nomear ou entoar notas sem nenhuma referência. Para desenvolver esta percepção, é necessário treinamento constante, porém, em aulas particulares muitas vezes este assunto não é abordado diretamente, seja por falta de tempo ou de interesse do aluno.

Nos cursos de graduação em música a disciplina de percepção musical é obrigatória e reserva em média duas horas por semana (GOLDEMBERG; OTUTUMI, 2008, p. 5). Não há dúvidas de que o treinamento da percepção musical com o professor seja mais dinâmico e de melhor qualidade. Porém, nos cursos particulares de instrumento musical ou canto, em que o tempo de aula é limitado, é interessante ter uma ferramenta de treinamento, para que o aluno faça os exercícios sem utilizar o tempo em aula.

Embora haja a convicção de que o uso de computadores não possa e não deva substituir o educador musical, muitos professores se manifestam contrários à adoção deste tipo de tecnologia. No contexto das Ciências Exatas as controvérsias geradas pelo uso da informática na educação são menores, pois há mais de quarenta anos o computador auxilia na solução de problemas matemáticos e lógicos. Entretanto, nas Ciências Humanas e nas Artes, como é o caso da Música, a utilização de computadores não é de longa data, portanto, o surgimento de tal resistência é compreensível (MILETTO; COSTALONGA; FLORES; FRITSCH; PIMENTA; VICARI, 2004).

O objetivo deste trabalho é o desenvolvimento de um software educativo-musical, que possibilite o professor de música a criar exercícios de percepção musical, e que este possa ser resolvido pelo aluno em qualquer computador.

Além de permitir que os exercícios sejam realizados em qualquer ambiente, o software educativo também gera um arquivo de relatório após a execução de um exercício, para que o professor mantenha o controle sobre o desempenho do aluno e possa desenvolver exercícios conforme o progresso do mesmo.

Para alcançar o objetivo deste trabalho, foi necessário o estudo sobre softwares educativos, teoria musical, banco de dados relacional, tecnologia de interface com aplicações musicais Midi (*Musical Instrument Digital Interface*) e metodologias de desenvolvimento de software.

Esta monografia está organizada da seguinte forma:

No capítulo 1 discursa-se sobre Software Educativo, descrevendo seus conceitos, origem e tipos;

O capítulo 2 trata da Percepção Musical, abordando também alguns conceitos sobre teoria musical e notação musical;

No capítulo 3 são apresentados fundamentos sobre Processo de Software, os modelos de processos de desenvolvimento de software e o detalhamento do processo utilizado;

No capítulo 4 são descritas as metodologias utilizadas para o desenvolvimento do software educativo;

O capítulo 5 descreve o desenvolvimento do software;

Por fim são apresentados as Conclusões, as Referências Bibliográficas utilizadas e dois apêndices, um com a implementação da classe AnalisadorFrases e outro com o Manual do Software.

CAPÍTULO 1 – SOFTWARE EDUCATIVO

Softwares permeiam todos os aspectos de nossa vida. Mesmo muitas vezes sem se fazer notar, tornam nossa vida mais confortável, eficiente e efetiva.

Meirelles (2004, p.34) define software como um conjunto de instruções arranjadas logicamente para serem inteligíveis pela Unidade Central de Processamento (UCP) de um computador. Sommerville (2008, p. 4) completa, afirmando que “software não é apenas o programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente”.

1.1 Definição

Para a comunidade de Informática na Educação (IE), todo o programa que utilize uma metodologia que o contextualiza no processo de ensino-aprendizagem pode ser considerado um programa educacional (GIRAFFA, 1999, p. 25).

1.2 Origem

As primeiras tentativas de desenvolvimento de softwares educacionais apoiaram-se no “behaviorismo”, ou teoria do condicionamento operante, que segundo Witter (1984) apud Fischer (2001, p. 18), a presença ou a ausência de estímulos, captados pelos sentidos, modificam o comportamento que caracteriza o humano. Burrhus Frederic Skinner, como professor da universidade de Harvard, no início de 1950 propôs uma máquina para ensinar usando o conceito de instrução programada, que consiste em modularizar o material a ser ensinado, encadeando-os de forma a controlar o avanço do estudante pelos módulos, de acordo com o seu desempenho avaliado no final de cada módulo (VALENTE, 1993).

A máquina de ensinar consistia numa caixa com uma abertura na sua parte superior onde era possível visualizar os problemas propostos, que vinham impressos em uma tira de papel. O aluno respondia movendo um ou mais dos cursores, onde estavam impressos os dígitos. As respostas eram impressas juntamente com as suas respectivas perguntas. Um botão devia ser girado ao término de cada resposta. Se esta estivesse correta, o botão giraria facilmente. Já se estivesse incorreta, o botão não giraria e o aluno teria que persistir na mesma questão até que conseguisse solucioná-la.

Pelo fato de ser apresentada na forma impressa nunca se tornou popular, pois era muito difícil a produção do material instrucional. Porém, com o advento do computador, nasceu a técnica denominada “instrução auxiliada por computador” no início da década de 60, conhecida também como *Computer Assisted Instruction* (CAI), baseada na instrução programada de Skinner (VALENTE, 1993).

A estrutura algorítmica do CAI é única e pré-definida, onde o aluno não influi na seqüencia, fazendo da transmissão do conhecimento previamente determinada. Estas características impediam uma adaptação individual ao estilo de cada aluno. Por ter esta limitação, surgiram os primeiros Sistemas Tutores Inteligentes conhecidos como *Intelligent Tutoring Systems* (ITS) ou *Intelligent Computer Assisted Instruction* (ICAI) (FISCHER, 2001, p. 45).

O ITS tem sua concepção baseada na Psicologia Cognitiva, que trata do modo como as pessoas percebem, aprendem, recordam e pensam sobre a informação (STERNBERG, 2000).

Os ITS busca tornar o programa educacional mais individualizado às necessidades cognitivas de um aluno. A estruturação do ITS também é subdividida em módulos, porém, a seqüencia se dá em função da resposta do aluno (FISCHER, 2001, p. 45).

As primeiras pesquisas em ITS utilizaram técnicas de Inteligência Artificial (IA) para que a apresentação do conteúdo a ser ministrado ocorresse de forma independente às técnicas didáticas. Porém, devido a sua complexidade na implementação, estes programas geralmente não funcionavam em microcomputadores, se tornando restrito aos centros universitários que possuíam recursos computacionais (FISCHER, 2001, p, 45).

Os CAI são oriundos de projetos da área de Educação, enquanto que os ITS surgiram a partir do desenvolvimento de projetos de pesquisas da área de IA.

Na prática, não há uma divisão clara entre CAI e ITS, pois em geral, não são nem totalmente fixos (pré-programados), nem totalmente autônomos (FISCHER, 2001, p. 47). Supondo a existência de tais softwares, os softwares totalmente fixos pré-programados seriam programas invariáveis, sem interação com o usuário, onde a entrada de dados não modifica a saída de dados. Já os totalmente autônomos, também em suposição, não dependeriam do conhecimento humano de espécie alguma para serem implementados.

1.3 Classificação

Existem diversas taxonomias, ou regras para classificação, encontradas na literatura. Valente (1993) classifica os softwares educativos de acordo com objetivos pedagógicos. Já Galvis (1997) apud Fischer (2001, p. 47), classifica os softwares de acordo com suas funções específicas. Porém, Giraffa (1999, p. 26) afirma que as taxonomias mais utilizadas não contemplam as modalidades que utilizam técnicas de IA e ambientes cooperativos. Afirma ainda que, muitas taxonomias não levam em consideração o tipo de aprendizagem proporcionada pelo ambiente, ou seja, o tipo do conhecimento a ser transmitido pelo ambiente.

A definição de software educativo permite que uma série de programas desenvolvidos para outras aplicações venham a ser utilizados como programas educacionais. Apesar de ampliar o conjunto de programas, faz necessário revisar a taxonomia utilizada, pois a classificação através de suas características estruturais e funcionais facilita o acesso em uma biblioteca de programas escolares e favorece a análise tanto sob o ponto de vista de projeto quanto do ponto de vista pedagógico (GIRAFFA, 1999, p. 25).

Na taxonomia proposta por Giraffa (1999, p. 26), os softwares são divididos em duas grandes categorias, separando-os pelo tipo de aprendizagem, que podem ser de habilidades específicas ou de habilidades cognitivas amplas, conforme seguem descritos a seguir:

Categoria 1 – Aprendizagem de Habilidades Específicas: Programas onde a aprendizagem proporcionada pelo ambiente está centrada na aquisição de habilidades específicas (motricidade fina, percepção, identificação e outras). São divididos em dois grandes grupos denominados CAI e ICAI que seguem descritos:

- Grupo dos CAI (*Computer Assisted Instruction*) – Oferecem suporte ao ensino de habilidades específicas sem a utilização do modelo do aluno para orientar a formação de interação. São divididos em quatro subgrupos:
  - *Tutoriais:* versão computacional da instrução programada, com a vantagem de poder apresentar o material com animação, som e podendo controlar o desempenho do aprendiz. Não apresentam grandes revoluções, sob o ponto de vista pedagógico.
  - *Exercício-prática:* São usados para revisar material visto em classe, envolvem memorização e repetição, como aritmética e vocabulário. Requerem a resposta freqüente do aluno, propiciando retorno imediato. A

vantagem é que o professor dispõe de uma infinidade de exercícios que o aprendiz pode resolver conforme o seu grau de conhecimento. O software educativo desenvolvido neste projeto é classificado neste subgrupo.

- *Demonstração:* O aprendiz pode controlar o ritmo e a seqüência do percurso, que visa esclarecer um determinado assunto.
- *Jogos e Simulação:* Jogo é um processo intrinsecamente competitivo (em que co-existem a vitória e a derrota). Apesar de ter grande impacto a princípio, a competitividade gera ineficiência no decorrer do tempo. A simulação é a simples execução dinâmica de um modelo previamente definido.

❖ Grupo dos ICAI (*Intelligent Computer Assisted Instruction)* – Baseiam-se no conteúdo e independe do método de ensino utilizado (estratégias e táticas empregadas na interação com o aluno) em relação ao domínio. São divididos em três subgrupos:

- *Sistemas Tutores Inteligentes (ITS):* uso de perguntas indiretas considerando o nível de compreensão do aluno.
- *Sistemas de Treinamento Inteligente:* baseiam-se no desempenho individual do aluno e fornecem conselhos para o desempenho em uma área específica.
- *Ambientes de Ferramentas Inteligentes:* trabalham junto com o usuário

Categoria 2 – Aprendizagem de Habilidades Cognitivas Amplas: Deseja-se que os alunos obtenham um nível mais elevado de aprendizagem que ultrapasse as habilidades de níveis anteriores (mais simples). Contempla os Ambientes de Ensino-Aprendizagem Computadorizados. São divididos em quatro grupos:

- *Micromundos:* Ambiente permite que o aluno trabalhe de forma diversificada, segundo seu próprio ritmo e ainda possibilita que ele construa sua solução utilizando recursos de programação inerentes ao ambiente. Foram criados para desenvolver habilidades cognitivas no aluno e para proporcionar o pensamento reflexivo.
- *Sistemas de Autoria:* Ferramentas de criação. Oferece ao aluno a possibilidade de explorar um conjunto amplo de habilidades cognitivas exercendo sua criatividade.

- *Jogos Educacionais:* Diferentes dos apresentados na categoria 1, existe um modelo de simulação onde o tipo de ação executada pelo aluno fará diferença no resultado do jogo.
- *ILE (Intelligent Learning Environment):* Conhecidos também como Sistemas Tutores Coopeerativos, ou Sistemas de Aprendizagem Social. Utilizam técnicas de IA no seu projeto e no seu desenvolvimento. Consideram mais de um aluno ou mais de um tutor trabalhando no mesmo ambiente.

Enquanto a comunidade de IE aceita que qualquer software contextualizado no processo de ensino-aprendizagem é software educativo, torna-se necessário uma análise criteriosa da taxonomia a ser utilizada. Giraffa (1999) apresenta uma visão crítica em relação às classificações mais utilizadas, apresentando uma nova classificação incluindo subdivisões antes não utilizadas. Verifica-se também que Giraffa (1999) não considera os ITS como sinônimos de ICAI, e sim, uma subdivisão.

Devido a este detalhamento na classificação proporcionado pela taxonomia apresentada por Giraffa (1999), e por ter sido elaborado visando uma melhor divisão do crescente número de softwares educativos, conclui-se que seja a melhor taxonomia a ser utilizada atualmente.

Portanto, utilizando a classificação proposta por Giraffa (1999), o software desenvolvido neste trabalho é da modalidade dos programas de aprendizagem de habilidades específicas, do grupo dos CAI e do tipo Exercício-Prática.

CAPÍTULO 2 – PERCEPÇÃO MUSICAL

A música, do grego *musiké téchne* (arte das musas) é uma arte constituída de sons e silêncios na dimensão do tempo. Sua definição pode variar de acordo com a cultura e contexto social. Os elementos artísticos que compõem a música são melodia, harmonia, ritmo, dinâmica e timbre.

Quando um músico faz uma composição musical ou uma transcrição musical, ele necessita fazer uma análise dos elementos musicais. Para tal análise, utiliza-se a percepção musical.

2.1 Conceitos

A percepção musical é uma capacidade indispensável para os músicos, pois esta ajuda na musicalidade do indivíduo. A percepção musical é a percepção sonora no contexto musical, ou seja, a capacidade de perceber ondas sonoras como parte de uma linguagem musical.

A percepção musical envolve principalmente a percepção sonora, que é a capacidade de identificar atributos físicos do som, como volume, timbre e afinação. Além da percepção sonora, percepção musical envolve também elementos musicais como melodia (percepção melódica), ritmo (percepção rítmica), e harmonia (percepção harmônica).

A capacidade de perceber tais ondas sonoras está associada à estrutura cognitiva do indivíduo.

> Nosso ouvido funciona em duas dimensões, ou seja, é capaz de observar e perceber detalhes, mas também de perceber estruturas. No entanto, a percepção auditiva desses elementos está vinculada a processos psicológicos, os quais determinarão sobre qual característica desses elementos nossa atenção se voltará. Nestes casos, a percepção é baseada em estruturas cognitivas adquiridas pela experiência vivida, ou seja, a cultura em que estamos inseridos e o nosso treinamento musical (DAMIAN, 2000, p. 1).

A freqüência das ondas sonoras, medida em Hertz (Hz) determina a altura da nota (grave ou aguda). Uma das primeiras etapas para o treinamento da percepção musical é a capacidade de identificar a altura da nota. Para esta capacidade, existe o conceito de ouvido musical, explicado no próximo item.

2.2 Ouvido Musical

Para os músicos, é muito importante desenvolver seu ouvido musical, ou a apuração e sensibilidade para a música. Existem dois tipos de ouvidos musicais: ouvido relativo e ouvido absoluto.

2.2.1 Ouvido Absoluto

Alguns músicos, mesmo os principiantes, tem o chamado “ouvido absoluto”. Vanzella, Oliveira e Werke (2008, p. 1) definem da seguinte forma:

> O ouvido absoluto é um traço cognitivo caracterizado pela capacidade de identificar a altura de qualquer tom isolado usando rótulos como dó (261 Hz) e/ou de produzir um tom específico (através do canto, por exemplo) sem nenhuma referência externa.

Este é um assunto que sempre gera muita polêmica, no meio artístico, e principalmente, dentro da comunidade científica. Isto porque apesar de muitas pesquisas realizadas ainda não está suficientemente esclarecida a origem dessa habilidade. Há autores que acreditam que tal habilidade (ouvido absoluto) é um dom inato, e outros que acreditam que pode ser adquirida através de treinamento.

Segundo Bachem (1937) apud Vanzella, Oliveira e Werke (2008, p. 2) existem diferentes graus ou níveis de precisão do ouvido absoluto.

A incidência de indivíduos que possuem ouvido absoluto na população em geral é estimada em 1/1500 a 1/10.000 (BACHEM, 1955; PROFITA; BIDDER, 1988; BACHEM, 1937 apud VANZELLA; OLIVEIRA; WERKE, 2008, p. 2).

2.2.2 Ouvido Relativo

O ouvido relativo, cuja utilização é amplamente difundida e reconhecida, realiza uma audição abstrata, necessitando de referenciais para reconhecer sons musicais através de elaboração intelectual (DAMIAN, 2000).

Para entender a diferença entres os ouvidos musicais, segue o seguinte exemplo:

> [...] imaginemos que sejam apresentadas as seguintes freqüências: 392; 440; 392; 329,6; 392; 440; 392 e 329,6 Hz para vários indivíduos. Um leigo (sem memória musical) diria que ouviu um conjunto de sons; uma pessoa comum, habituada a ouvir música informalmente, identificaria a seqüência como sendo a melodia da canção "Noite Feliz"; um estudante de música poderia citar a relação intervalar, ou seja, 2ª maior ascendente; 2ª maior descendente; 3ª menor desc.; 3ª menor asc.; 2ª maior asc.; 2ª maior desc. e 3ª menor desc. No entanto, somente a pessoa com audição absoluta diria que a sucessão de sons tocados foi: sol, lá, sol, mi, sol, lá, sol e mi (DAMIAN, 2000, p. 1).

Segundo Damian (2000), tanto o ouvido absoluto como o ouvido relativo são capacidades que podem ser adquiridos ou perdidos, afirmando que um treinamento musical auditivo prolongado e sistemático possibilitaria ao músico que possui ouvido relativo, desenvolver a percepção absoluta dos sons e mesmo a adquirir a audição absoluta, e o indivíduo possuidor do ouvido absoluto pode, pela ausência de estímulos e treinamento, enfraquecer ou vir a perder esta capacidade.

2.3 Teoria Musical

Um dos pré-requisitos para o estudo da percepção musical deve ser o domínio da teoria musical. A teoria musical é extensa e complexa, sendo possível se aprofundar em muitos detalhes, com suas denominações específicas e origens históricas. Porém, a seguir será traçado apenas o necessário para o entendimento do software desenvolvido neste projeto.

No software desenvolvido neste projeto, o aluno de música deverá transcrever em forma de partitura uma frase musical desenvolvida pelo professor. Para a transcrição, necessita-se saber sobre a notação musical, ou a forma em que a música é descrita na forma gráfica.

Ao final do exercício, é gerado um relatório de desempenho, e para se entender como a resposta do aluno é avaliada, precisa-se de conhecimentos sobre notas musicais e intervalos.

2.3.1 Notas Musicais e Intervalos

Cada nível de som claro, sustentado, é uma nota. Sua origem física é a vibração, onde a freqüência determina a altura da nota (grave ou aguda).

Para exemplificar, a nota mais grave do piano, tem freqüência aproximada de 30 Hz, e a mais aguda, 4000 Hz (HOLST, 1998). Em outros exemplos será utilizado o piano para ilustrar, porém, não é necessário possuir um piano para o estudo sobre percepção musical. Por

ser um instrumento conhecido e que dispõe de todas as notas concomitantemente, é utilizado para facilitar o entendimento.

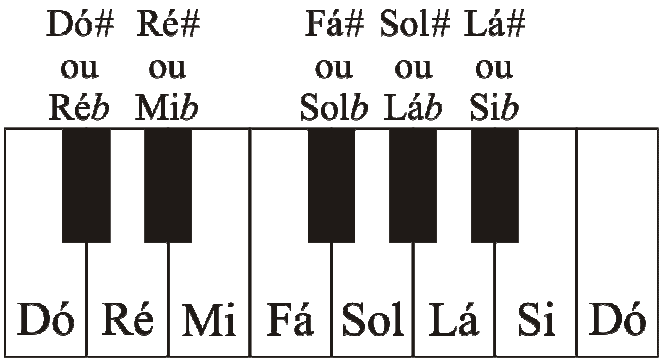
Intervalo é a diferença de altura (tonalidade) entre duas notas. A altura do som, determinada pela frequência, pode variar por graus infinitesimais.

Na música ocidental, os compositores se limitam a alguns poucos níveis de altura definidos, adotando como o menor intervalo o semitom, ou meio tom (MED, 1986).

Os sons musicais, ou notas musicais, são sete: Dó, Ré, Mi, Fá, Sol, Lá e Si. Os intervalos entre Mi e Fá, e entre Si e Dó são semitons (ou meio tom), e entre as outras notas os intervalos são de um tom. Conclui-se que existe uma nota intermediária entre notas cujo intervalo é de um tom. São as notas acidentadas, ou as teclas pretas do piano. É possível, conforme necessidade, alterar o tom de uma nota em um semitom, para cima ou para baixo. Para aumentar, utiliza se o sustenido (#), e para abaixar, o bemol (*b*).

Na Figura 1, verifica-se a disposição das notas musicas e notas acidentadas em um piano. Percebe-se que entre as notas Mi e Fá, e entre Si e Dó não existe a tecla preta, pois o intervalo entre estas são de semitons.

Figura 1 - Disposição de notas musicas em um piano



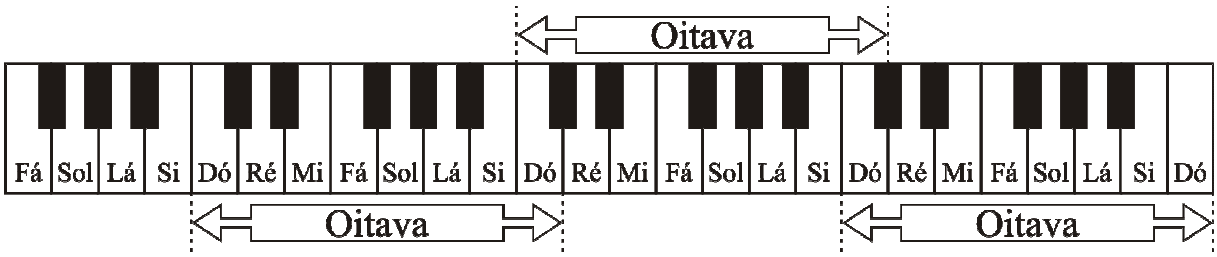
Fonte: Autoria própria

Seguindo a ordem das sete notas musicais, com a repetição do Dó no final, tem-se uma Oitava. O Dó no final de uma oitava é o mesmo Dó que inicia a oitava seguinte, conforme é mostrado na Figura 2.

Observa-se na Figura 2 que existem várias oitavas em alturas distintas, ou seja, existem vários Dó, ou qualquer outra nota, em freqüências distintas. As notas de mesmo nome, em alturas distintas têm uma relação especial. A freqüência da nota mais aguda é o dobro (ou qualquer potência de dois) da freqüência da nota mais grave e evocam quase a mesma sensação. Portanto, ao dobrar a freqüência de uma nota, obtém-se a mesma nota em

uma oitava acima (mais aguda), e ao dividir a freqüência na sua metade, obtém-se a mesma nota em uma oitava abaixo (mais grave).

Figura 2 - Disposição das oitavas em um piano



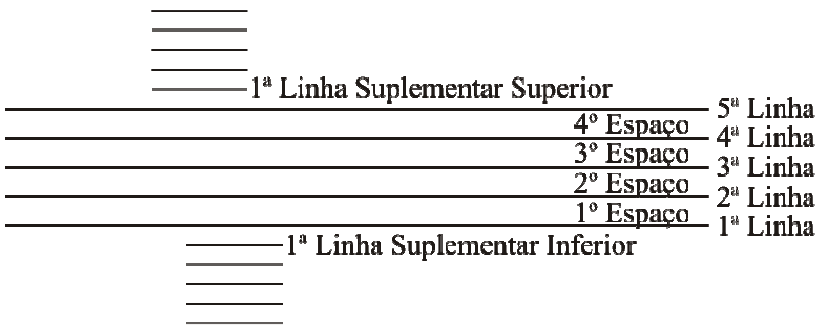
Fonte: Autoria própria

2.3.2 Notação Musical

A notação musical é utilizada para representar os sons musicais na forma escrita. Os principais elementos são: Pauta Musical, Claves, Notas e Compassos.

Pauta Musical – Atualmente chamado também de Pentagrama. É a reunião de cinco linhas horizontais, paralelas e eqüidistantes, que formam entre si quatro espaços. As linhas e os espaços são contados de baixo para cima. É nas linhas e nos espaços da pauta que se escrevem as notas. Conforme necessidade pode ser incluída linha suplementar, tanto acima da pauta, quanto abaixo, como é mostrada na Figura 3.

Figura 3 - Pauta Musical



Fonte: Autoria própria

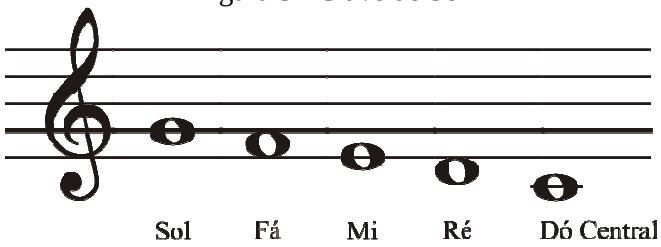
Claves – Sinal colocado na extremidade esquerda da pauta musical, determinando o nome das notas. Existem três espécies de clave: Clave de Sol, Clave de Fá e Clave de Dó, conforme são mostrados na Figura 4.

Figura 4 - Claves

Fonte: Autoria própria

No software será utilizada apenas a Clave de Sol, que atualmente é utilizada para todas as vozes e instrumentos agudos. Seu desenho laça a 2ª linha da pauta musical, que passará a representar a altura do sol acima do dó central, conforme é ilustrado na Figura 5.

Figura 5 - Clave de Sol



Fonte: Autoria própria

Notas – São figuras que indicam um som ou um silêncio, e inseridos na pauta musical, são lidas da esquerda para a direita. A sua posição (vertical) na pauta musical determina o tom, e o seu desenho indica a sua duração. A duração de uma nota não é fixa, apenas nos mostra a proporção em relação ao andamento da música.

Na tabela 1, observa-se a relação entre a duração das notas. A duração de uma semínima, por exemplo, equivale a um quarto da duração de uma semibreve e metade da duração de uma mínima. As notas podem representar um valor positivo (som) ou negativo (silêncio).

Existem outras notas com duração superior a semibreve ou inferior a semicolcheia, porém, as notas mais utilizadas na notação moderna e que serão utilizadas no software desenvolvido neste projeto estão relacionadas na tabela 1.

Compassos – Para representar a pulsação de uma música, são utilizadas barras verticais, cuja distância entre elas é denominada compasso. O compasso por sua vez é dividido em tempos. A contagem do tempo é indicada pela fórmula de compasso, que consiste em dois números sobrepostos. O número inferior representa a unidade de tempo

tendo em vista o valor de uma semibreve. O número superior indica a quantidade de unidades em cada compasso (HOLST, 1998).

Tabela 1 - Figuras, valores e nomenclatura

Figuras de Nota Valor Positivo	Nome	Equivalência	Figuras de Pausa Valor Negativo
	Semibreve	1	
	Mínima	1/2	
	Semínima	1/4	
	Colcheia	1/8	
	Semicolcheia	1/16	

Fonte: Autoria própria

Existem compassos simples e compostos, cuja diferença é a divisão da unidade de tempo. Nos compassos simples, cada unidade de tempo é dividida em duas metades. Porém, nos compassos compostos, cada unidade de tempo é dividida em três.

No software desenvolvido neste projeto, são utilizados apenas compassos simples, de fórmulas de compasso 2/4 (binário), 3/4 (ternário) e 4/4 (quaternário). As fórmulas binárias e quaternárias são muito utilizadas nos ritmos populares e em algumas músicas eruditas e jazz. A fórmula ternária é utilizada principalmente em ritmos de valsa. Em todas estas fórmulas o número inferior é quatro, ou seja, cada tempo do compasso equivale a uma semínima. Observando a tabela 1, que mostra a proporção de cada nota, conclui-se que utilizando estas fórmulas de compasso, a mínima equivale dois tempos, a semibreve quatro tempos, a colcheia meio tempo e a semicolcheia ¼ de tempo.

Porém, em cada uma das fórmulas a quantidade de unidades, neste caso a semínima, será diferente. No compasso binário, são duas semínimas, no ternário, três semínimas e no quaternário, quatro semínimas.

Se uma nota acidentada (sustenido ou bemol) for inserida em um compasso, todas as ocorrências seguintes dessa mesma nota dentro desse compasso deverão ser executadas com o mesmo acidente. Encerrando-se o compasso, as ocorrências seguintes dessa nota deverão ser executadas sem alteração. Quando se deseja que uma nota marcada com um acidente seja

executada na altura original usa-se um sinal chamado bequadro (♮), acidente musical que desativa o efeito do sustenido ou do bemol.

CAPÍTULO 3 – PROCESSO DE SOFTWARE

Na elaboração de um produto, é importante percorrer uma série de passos previsíveis (um roteiro) que resulte num trabalho de alta qualidade. Quando este produto é um software, tal roteiro é denominado “processo de software”, que produz uma variedade de programas, documentos e dados, e que nos fornece estabilidade, controle e organização no seu desenvolvimento (PRESSMAN, 2002).

Existe uma diversidade imensa de processo de software, porém, nenhum pode ser considerado um processo ideal. Para cada área de aplicação, juntamente com o tamanho do projeto ou complexidade, existe um processo mais adequado (SOMMERVILLE, 2008).

A seguir, serão apresentados conceitos sobre os modelos de processo de software mais utilizados e ao final, detalhamento do processo de software utilizado no desenvolvimento do software deste projeto, o Open UP.

3.1 Atividades Fundamentais

Apesar da grande diversidade de processo de software, há atividades fundamentais comuns a todos eles. Pressman (2002) categoriza-os em três fases genéricas: fase de definição, fase de desenvolvimento e fase de manutenção. Já Sommerville (2008), identifica quatro atividades básicas: especificação, projeto e implementação, validação e evolução. Apesar das fases genéricas de Pressman serem subdividas, não atendem muito bem aos modelos atuais mais utilizados, que enfatizam a atividade de teste e validação do software. Por esta razão, serão adotadas as atividades básicas definidas por Sommerville (2008):

1. Especificação de software: Definição das funcionalidades do software e as restrições em sua operação.
2. Projeto e implementação de software: Produção do software cumprindo sua especificação.
3. Validação de software: Verificação para garantir que o software é o que foi especificado.
4. Evolução do software: Modificação para atender às necessidades mutáveis do cliente e do mercado.

Essas atividades básicas são organizadas de modo diferente nos diversos modelos de processo de software.

3.2 Modelos de Processo de Software

Os modelos de processo de software, às vezes chamados de paradigmas de processo, são abstrações do processo que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de software. Ou seja, não são descrições definitivas de processos de software, e sim, são modelos genéricos e podem ser entendidos como *framework* de processo, que podem ser customizadas para criar processos mais específicos (SOMMERVILLE, 2008).

A seguir, serão apresentados os principais modelos de processo de software.

3.2.1 Modelo Cascata

O modelo em cascata é o primeiro modelo de processo de desenvolvimento de software publicado (SOMMERVILLE, 2008, p. 44).

Neste modelo, o desenvolvimento de um estágio (atividade fundamental) deve terminar antes de o próximo começar (PFLEEGER, 2004, p. 39).

Esta característica torna o modelo desvantajoso, pois a alteração de requisitos ou problemas no projeto pode implicar na repetição de estágios anteriores do processo (SOMMERVILLE, 2008, p. 45).

Por outro lado, devido sua simplicidade, pode ser muito útil para ajudar os desenvolvedores a descrever o que eles precisam criar. Por este fato, muitos outros modelos mais complexos são variações do modelo cascata, incorporando *loops* de *feedback* e atividades extras (PFLEEGER, 2004, p. 39).

Os principais estágios do modelo demonstram as atividades fundamentais de desenvolvimento (SOMMERVILLE, 2008, p. 44):

- *Análise e definição de requisitos:* Definição detalhada de serviços, restrições e objetivos por meio de consulta com usuários do sistema.
- *Projeto de sistema e software:* Divisão dos requisitos em sistemas de hardware ou de software. Estabelece uma arquitetura geral do sistema.
- *Implementação e teste de unidade:* O software é desenvolvido como um conjunto de programas. Cada unidade do software é testada para verificar se atende à sua especificação.
- *Integração e teste de sistema:* Integração das unidades do software e teste do sistema completo. Entrega do sistema para o cliente.

- *Operações e manutenção:* Instalação e operação do sistema. Correção de erros, aprimoramento e ampliação dos serviços conforme identificação de requisitos.

Portanto, em sistemas em que os requisitos são bem compreendidos e a probabilidade de mudanças radicais é baixa, o modelo em cascata pode ser vantajoso. Porém, quando se deseja um projeto flexível que atenda às mudanças de requisitos, este processo deve ser evitado.

3.2.2 Desenvolvimento Evolucionário

Baseia-se na idéia de desenvolvimento de uma implementação inicial com o intuito de obter *feedback* do usuário e refinar o resultado por meio de versões. As atividades de especificação, desenvolvimento e validação são intercaladas (SOMMERVILLE, 2008, p. 45).

Segundo Sommerville (2008) existem dois tipos de desenvolvimento evolucionário:

- *Desenvolvimento exploratório:* Objetivo do processo é trabalhar com o cliente explorando requisitos. Inicia com as partes do sistema compreendidas, e o sistema evolui por meio da adição de novas características propostas pelo cliente.
- *Prototipação throwaway:* Objetivo do processo é compreender os requisitos do cliente para desenvolver melhor definição de requisitos. O protótipo se concentra na experimentação dos requisitos mal definidos pelo cliente, ou seja, o cliente especifica e refina os requisitos utilizando protótipos para avaliar suas necessidades.

O desenvolvimento evolucionário é freqüentemente mais eficaz que a abordagem em cascata, pois a especificação pode ser desenvolvida de forma incremental. Porém, Sommerville (2008) observa dois problemas:

- Devido rapidez no desenvolvimento, não é viável economicamente produzir documentos que refletem cada versão do sistema, tornando o sistema invisível, ou impossível de medir o progresso.
- A mudança contínua corrompe a estrutura do software, tornando-o mal estruturado e desta forma tornando difícil a incorporação de mudanças de software.

Portanto, para sistemas complexos em que é necessária uma arquitetura estável e diversas equipes desenvolvem diferentes partes do sistema, o desenvolvimento evolucionário

pode não ser vantajoso. Porém, para sistemas de pequeno e médio porte a abordagem evolucionária pode ser o melhor método de desenvolvimento (SOMMERVILLE, 2008).

3.2.3 Modelo em Espiral

Uma vez levantados os requisitos, a probabilidade de sua concretização é denominada “risco”. Para combinar o desenvolvimento com o gerenciamento de risco, de modo a minimizar e controlar os riscos, Boehm (1988) propôs o modelo em espiral do processo de software (PFLEEGER, 2004, p. 46).

O modelo em espiral representa o processo de software como uma espiral, onde cada loop na espiral representa uma fase. Cada loop está dividido em quatro setores (SOMMERVILLE, 2008, p. 48):

- *Definição de objetivos:* Definição dos objetivos específicos desta fase. Identificação de riscos e restrições sobre o processo e o produto.
- *Avaliação e redução de riscos:* Análise detalhada para cada risco identificado. Redução de riscos, utilizando por exemplo, um protótipo.
- *Desenvolvimento e validação:* Definição do modelo desenvolvimento para o sistema.
- *Planejamento:* Revisão do projeto e tomada de decisão para o próximo loop da espiral. Se a decisão for pelo prosseguimento, serão elaborados planos para a próxima fase.

3.2.4 Metodologias Ágeis

Os métodos ágeis surgiram da insatisfação de pequenas e médias empresas, que utilizando abordagens de desenvolvimento complexas e baseadas em planos, tiveram o tempo gasto para determinar como o sistema deveria ser desenvolvido maior do que o tempo empregado no desenvolvimento do programa e testes. Esta insatisfação é compreensível, pois tais abordagens eram voltadas para sistemas críticos, ou seja, sistemas em que as falhas poderiam resultar em perdas econômicas significativas, danos físicos ou ameaças à vida humana. Estes sistemas críticos eram desenvolvidos por equipes grandes que algumas vezes trabalhavam para empresas diferentes, separadas geograficamente, e trabalhavam no software durante vários anos (SOMMERVILLE, 2008, p. 262).

Apesar de existir diversos métodos ágeis cada qual propondo um processo diferente, todos compartilham um conjunto de princípios, denominada Manifesto Ágil, declaração que fundamentam o desenvolvimento ágil de software, conforme ilustra a Figura 6.

Figura 6 - Manifesto Ágil

Fonte: http://www.agilemanifesto.org/iso/ptbr/

Sommerville (2008, p. 263) aponta vários problemas em relação ao desenvolvimento utilizando métodos ágeis, como a dependência existente da disposição e capacidade do cliente e a dificuldade em definir contrato para este tipo de desenvolvimento. Ainda ressalta o problema de contratos:

> [...] os métodos ágeis têm de contar com contratos em que o cliente paga pelo tempo necessário para o desenvolvimento do sistema, em vez de pelo desenvolvimento de um conjunto específico de requisitos. Desde que tudo

> esteja bem, isso beneficia tanto o cliente quanto o desenvolvedor. Contudo, se surgirem problemas pode haver disputas difíceis sobre quem tem a culpa quem deve pagar pelo tempo extra e recursos necessários para resolver os problemas (SOMMERVILLE, 2008, p. 263).

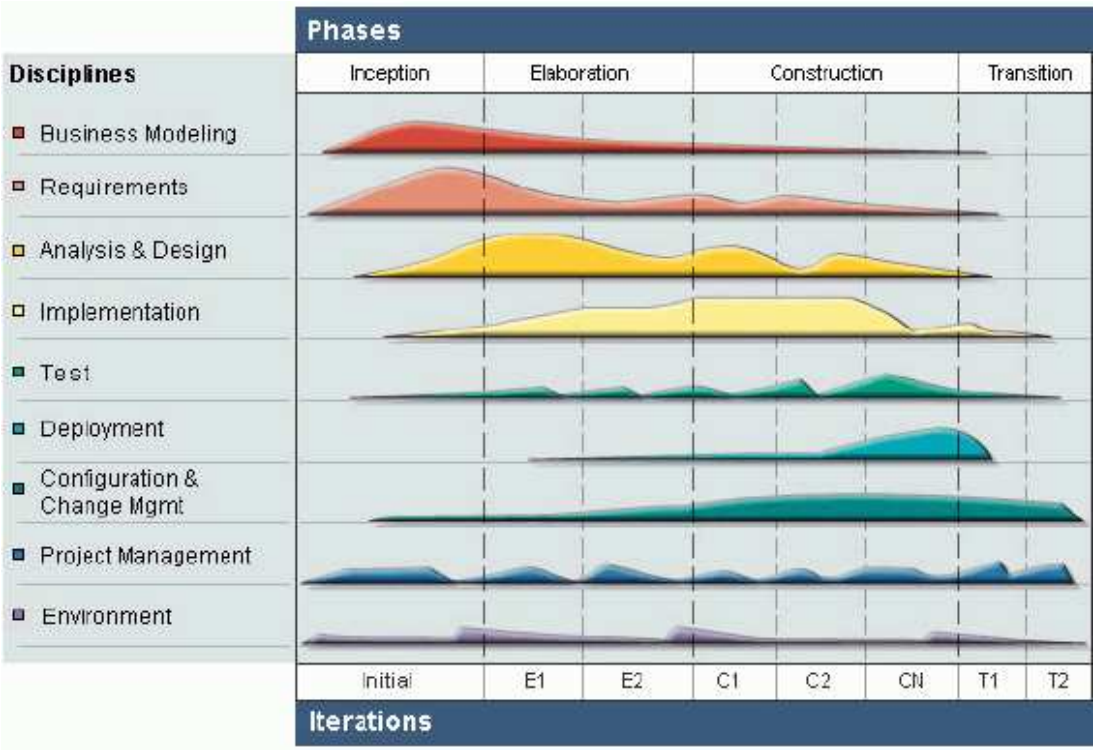
3.3 Rational Unified Process

O *Rational Unified Process* (Processo Unificado Rational), ou RUP, é um modelo híbrido de processo, que traz elementos de todos os modelos genéricos de processo, derivado do trabalho sobre a UML (*Unified Modeling Language*) (SOMMERVILLE, 2008).

O desenvolvimento é iterativo e incremental, e seu ciclo de vida contém quatro fases, abordando nove atividades, ou *workflows.*

Em cada fase do ciclo de vida, pode existir mais de uma iteração, sendo que em cada iteração há um enfoque maior em determinado *workflow*. Na Figura 7 é ilustrado o enfoque dos *workflows* em cada fase.

Figura 7 - Disciplinas e Fases



Fonte: http://www.ibm.com/developerworks/webservices/library/ws-soa-term2

As fases do RUP são:

- *Concepção:* Define o escopo do projeto
- *Elaboração:* Detalha os requisitos e a arquitetura
- *Construção:* Desenvolve o sistema
- *Transição:* Implanta o sistema

Os *workflows* são atividades que ocorrem durante o processo de desenvolvimento. Existem seis *workflows* de processo principais e três *workflows* de apoio principais (SOMMERVILLE, 2008, p. 55):

- *Modelagem de negócios*: Modelagem dos processos de negócios utilizando casos de uso de negócios;
- *Requisitos:* Identificação dos atores do sistema e desenvolvimento de casos de uso para modelagem de requisitos;
- *Análise de negócios:* Criação de modelo de projeto utilizando modelos de arquitetura, modelos de componente, modelos de objeto e modelos de seqüência;
- *Implementação:* Implementação e estruturação dos componentes de sistema em subsistemas de implementação;
- *Teste:* Processo iterativo realizando em conjunto com a implementação;
- *Implantação:* Criação, distribuição e instalação de uma versão do produto;
- *Gerenciamento de configuração e mudanças:* Gerencia as mudanças do sistema;
- *Gerenciamento de projetos:* Gerencia o desenvolvimento do sistema;
- *Ambiente:* Disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

O RUP descreve ainda boas práticas de engenharia de software recomendadas para uso em desenvolvimento de sistemas. Segue as seis melhores práticas fundamentais (SOMMERVILLE, 2008):

- *Desenvolver o software iterativamente:* Desenvolver e entregar as características de maior prioridade no processo de desenvolvimento, planejando os incrementos de software com base nas prioridades do cliente.
- *Gerenciar requisitos:* Documentar e acompanhar alterações de requisitos do cliente, analisando o impacto das mudanças antes de aceitá-las.
- *Usar arquiteturas baseadas em componentes:* Estruturar a arquitetura de sistema com componentes.

- *Modelar o software visualmente:* Abstrair o sistema e representá-la utilizando blocos de construção gráfica, utilizando UML.
- *Verificar a qualidade do software:* Garantir que o software atenda aos padrões de qualidade.
- *Controlar as mudanças de software:* Utilizando um sistema de gerenciamento de mudanças e procedimentos e ferramentas de gerenciamento de configuração, gerenciar as mudanças do software.

Sommerville (2008) afirma que diagramas como a Figura 7 dificultam a compreensão, pois combinam perspectivas estáticas e dinâmicas. Porém, para se ter uma visão do ciclo de vida como um todo, a sua utilização torna-se necessária.

O RUP é um processo pesado e aplicável a grandes equipes de desenvolvimento e a grandes projetos. Porém, o fato de ser amplamente customizável, torna possível a adaptação para projetos de qualquer escala.

Inicialmente, o software do projeto proposto seria desenvolvido utilizando o RUP, porém, por ser um processo complexo e trabalhoso, certamente o tempo despendido para adquirir conhecimentos para utilizá-lo seria maior que o tempo para desenvolver um software de pequeno porte. Por este motivo, foi decidido utilizar o Open UP, um processo ágil que mantém as boas práticas contidas no RUP.

3.4 Open UP

O Open UP (Processo Unificado Aberto) é uma metodologia ágil de desenvolvimento de software desenvolvida pela parceria IBM-Eclipse. Desenvolvida inicialmente com o intuito de ser uma versão ágil do RUP, está em conformidade acordo com o Manifesto do Desenvolvimento de Software Ágil e possui uma abordagem iterativa e incremental.

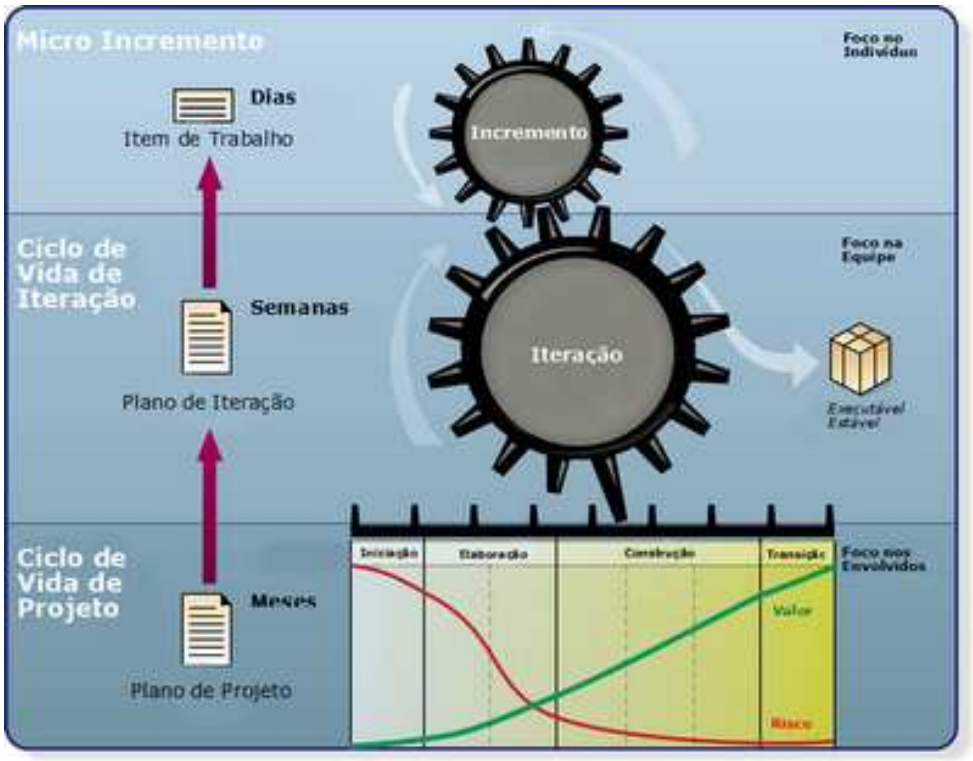
Santos (2009), diz que o Open UP é “Mínimo, Completo e Extensível, valorizando a colaboração entre a equipe e os benefícios aos interessados ao invés da formalidade e entregáveis desnecessários”.

O Open UP é baseado nas principais características do RUP, porém, livre de ferramentas e de baixo formalismo que pode ser estendido a uma variada gama de tipos de projetos e não apenas desenvolvimento de software (MEIRA, 2010).

Apesar de ter sua origem a partir do RUP, o Open UP apresenta uma quantidade bem menor de produtos de trabalho, papéis e tarefas. Outra diferença é a introdução do conceito de micro incrementos (SANTOS, 2009).

O desenvolvimento é estruturado em três camadas distintas, como ilustra a Figura 8:

Figura 8 - Desenvolvimento estruturado em três camadas



Fonte: http://open2up.blogspot.com/

As três camadas de desenvolvimento são:

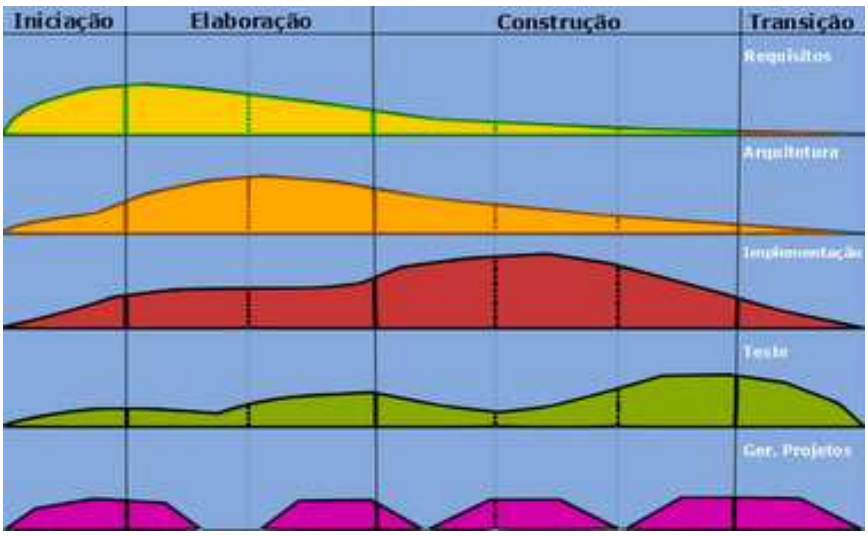
Ciclo de Vida de Projeto: Trata do processo de desenvolvimento como um todo. Dividida em quatro fases, enfatizando em cada fase determinados processos. Ao final de cada fase é gerado um conjunto de atividades e artefatos. As fases são:

- *Iniciação:* Análise de negócios e análise de requisitos, dando uma ênfase menor a arquitetura e implementação;
- *Elaboração:* Análise arquitetural da solução proposta;
- *Construção:* Implementação, teste e integração;
- *Transição:* Implantação do *release*, teste beta, reconfiguração necessária do sistema, treinamento do usuário e conversão dos dados legados.

Ciclo de Vida de Iteração: As atividades principais são divididas em subatividades (iteração). Cada iteração deve resultar em um *Build* (executável), que deve ser exaustivamente

testado. As disciplinas (*workflows)* tratadas são: Requisitos, Arquitetura, Implementação, Teste e Gerência de Projetos, como mostra a Figura 9.

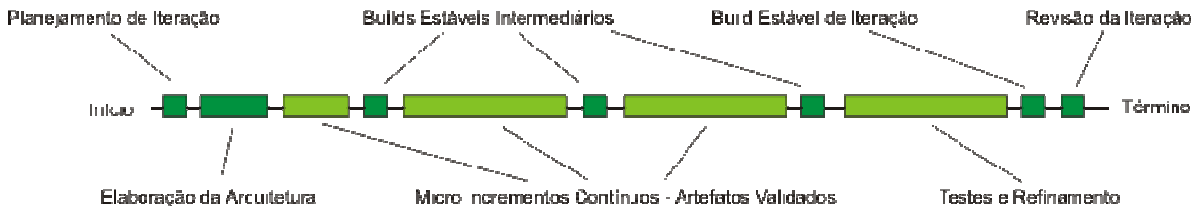
Figura 9 - Distribuição das disciplinas pelas fases



Fonte: http://open2up.blogspot.com/

Ciclo de Vida de Micro Incremento: Partilha as atividades da equipe de desenvolvimento em pequenas unidades, onde cada unidade se encerra com a entrega de um artefato de valor para a equipe. Provêm um *feedback* rápido, direcionando as decisões tomadas ao término de cada iteração. A Figura 10 apresenta o ciclo de vida de micro incremento.

Figura 10 - Ciclo de Vida de Micro Incremento



Fonte: http://open2up.blogspot.com/

CAPÍTULO 4 – METODOLOGIA

Neste capítulo serão apresentados os métodos, conceitos e ferramentas utilizados no desenvolvimento do software educativo deste projeto.

4.1 Orientação a Objeto

Utilizando um processo de desenvolvimento de software incremental, como é o caso do Open Up, a utilização das práticas de Orientação a Objeto ocorre obrigatoriamente, pois somente desta forma é possível desenvolver funcionalidades independentemente.

O projeto foi desenvolvido utilizando práticas de Orientação a Objeto (OO), ou seja, análise, projeto e modelagem orientados a objetos.

A OO organiza os problemas e suas soluções como um conjunto de objetos distintos que incluem a estrutura e comportamento dos dados, ou seja, o sistema é representado por objetos que possuem características (atributos) e ações (métodos) (PFLEEGER, 2004, p. 210).

Este paradigma de desenvolvimento possui várias vantagens, como o aumento da reutilização de código (neste caso, objetos), facilidade de manutenção devido a sua organização, e segurança de dados provida pelo encapsulamento.

A desvantagem da modelagem orientada a objetos é sua complexidade, que exige uma boa abstração na criação dos objetos, e mais tempo do desenvolvedor.

4.2 C++ Builder

O software foi codificado em C++, por existir certo conhecimento na linguagem e por oferecer flexibilidade ao programar em baixo nível, na manipulação de bytes.

Para a geração de códigos e criação da interface gráfica foi utilizado o C++ Builder 2010, IDE (*Integrated Development Environment*) desenvolvida pela CodeGear, divisão da Borland. Em 2008 a CodeGear foi comprada pela empresa Embarcadero.

O C++ Builder possui um ambiente de desenvolvimento de interface, além de conexão nativa com Banco de Dado e um bom depurador (*debugger*). Outra característica é a implementação de "__property", não presente na linguagem ANSI C++. Esta declaração permite a criação de propriedade, que pode conter uma função *getter* (leitura), *setter* (escrita)

ou ambos. Esta propriedade facilita o encapsulamento dos objetos e melhora a legibilidade do código, pois permite a leitura e escrita por meio de um mesmo nome.

4.3 Firebird

Para o armazenamento de dados, foi utilizado o Sistema Gerenciador de Banco de Dados (SGBD) Firebird na versão 2.5.

O Firebird é gratuito, e foi criado baseado no código do Interbase, SGBD proprietário desenvolvido pela Borland. O Interbase, criado com propósitos acadêmicos, liberou o código em 2000, na sua versão 6.0. Porém, posteriormente voltou a ter licença proprietária.

Por se tratar de desenvolvimento de um software educativo, seria inadequado utilizar um SGBD proprietário, uma vez que aumentaria o custo do produto final, dificultando o acesso do interressados. Outra vantagem é que o Firebird não depende muito do hardware, podendo ser instalado em máquinas antigas, ou de baixo desempenho. Por estes motivos, o Firebird é o SGBD ideal para este projeto.

4.4 Padrões de Projeto

Padrões de projeto, ou *Design Patterns*, são soluções encontradas para problemas comuns. Estes padrões não é uma especificação detalhada, sendo uma descrição de conhecimento e experiência acumulados, numa solução comprovada para um problema comum (SOMMERVILLE, 2008, p. 279).

No software desenvolvido neste projeto, foi utilizado o padrões de projeto *Singleton*.

O *Singleton Pattern* especifica que uma classe pode ser instanciada apenas uma vez, retornando a instância quando esta já existir. Sua utilização no software desenvolvido surgiu primeiramente para melhorar o desempenho, criando a classe de conexão ao banco de dados que implementa o *Singleton Pattern*, pois esta só poderia ser instanciada uma única vez. Porém, durante o desenvolvimento, foram surgindo problemas de necessidade de variáveis globais. Novamente utilizando o *Singleton Pattern*, foi possível criar classes que mantivessem os valores dos atributos durante a execução do software.

4.5 Tecnologia MIDI

A tecnologia MIDI (Musical Instrument Digital Interface), ou Interface Digital para Instrumentos Musicais é um protocolo que permite a comunicação entre instrumentos musicais (teclados e sintetizadores) e equipamentos eletrônicos, inclusive o computador. O padrão MIDI é composto por linguagem de comunicação, hardware, interfaces para transmissão e recepção de informação e o formato em que isso é feito.

O MIDI provê arquivos de extensão “mid”, e estes permitem a reprodução em qualquer equipamento que suporte o padrão. Diferentemente de outros formatos de multimídia (WAV, MP3), um arquivo MIDI não contém o áudio propriamente dito, e sim, as instruções para produzi-lo.

4.6 StarUML

Para a modelagem de diagramas do software desenvolvido neste projeto foi utilizado o StarUML, na versão 5.0.2.1570.

O StarUML é um software gratuito de modelagem de diagramas baseada na UML, possuindo interface intuitiva e leve, traz funcionalidades como geração de código e engenharia reversa, ou seja, a partir de um produto final obter parte de seu processo de desenvolvimento.

Diferentemente de vários outros softwares de modelagem UML, a StarUML não necessida da *Java Runtime Environment* (JRE), ou Ambiente de Tempo de Execução Java.

4.7 Imagens e Ícones

Para a geração de imagens utilizadas no software e monografia, foi utilizado o Encore na versão 5 e o CorelDraw na versão X3.

O Encore, desenvolvido pela GVOX, permite a criação de partituras, conversão de partitura para MIDI, e conversão de MIDI para partitura.

O CorelDraw, desenvolvido pela Corel Corporation, é um complexo software de criação de gráficos vetoriais, que permite a exportação em vários formatos.

Os ícones foram criados utilizando o software IcoFX, na versão 1.6.4. Este software, apesar da distribuição ser *Freeware*, possui recursos que muitos editores proprietários não oferecem como a importação de imagem e geração automática de ícones em vários tamanhos.

4.8 HelpNDoc

HelpNDoc é um software de edição e compilação de sistemas de ajuda (*help system*).

Para a geração do sistema de ajuda do software e manual foi utilizado o HelpNDoc na versão 2.8.0.53 *Freeware Edition*.

Apesar de ser um software gratuito, o HelpNDoc possui diversas vantagens em relação a vários outros editores, tanto gratuitos quanto proprietários. Além de fácil customização, seu compilador gera o sistema de ajuda e manual em formato “doc” (Microsoft Word) e “pdf” (Adobe System).

Em contrapartida, inclui notas de rodapé em todo o conteúdo, com informações do software HelpNDoc.

4.9 InnoSetup

Para a criação do pacote de instalação do software desenvolvido neste projeto foi utilizado o utilitário Inno Setup.

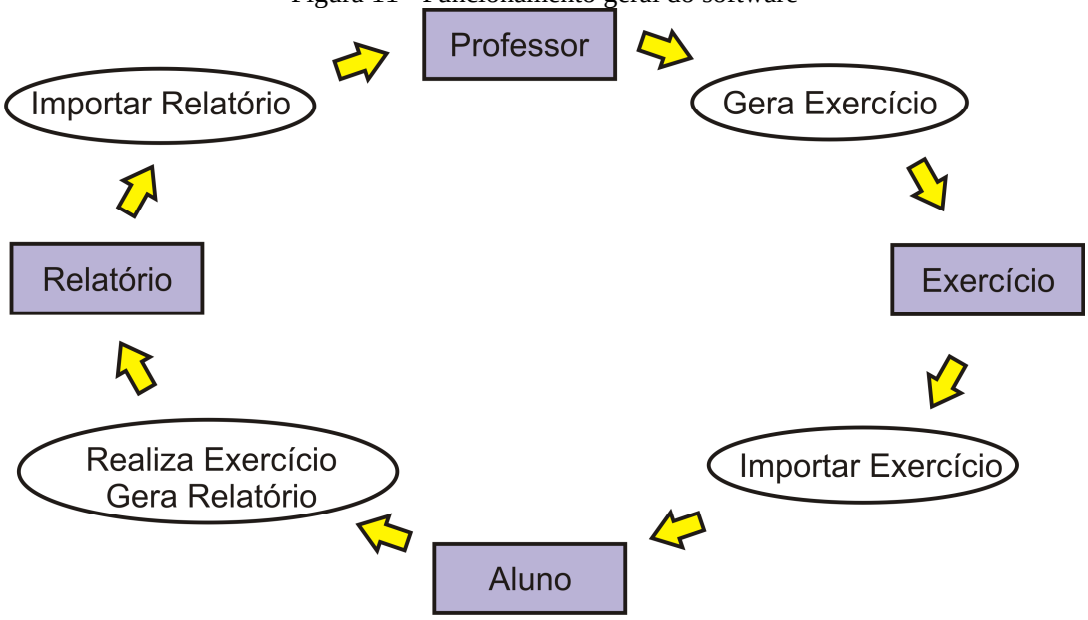
Este programa dispõe de interface intuitiva, existindo a opção de criar um instalador com o *wizard* (passo-a-passo). Pode também ser customizado, determinando o destino de cada arquivo a ser instalado, bem como incluir outro instalador dentro do próprio pacote.

CAPÍTULO 5 – IMPLEMENTAÇÃO DO SOFTWARE

O software educativo desenvolvido neste projeto, denominado *Zettai Onkan*, que em japonês significa “afinação perfeita”, tem como objetivo o treinamento em percepção musical.

O treinamento consiste em exercícios de transcrição de frases musicais para partituras. Estes exercícios são criados pelo professor, e ao serem respondidos pelo aluno, retornará um relatório de desempenho ao professor, conforme é ilustrado na Figura 11.

Figura 11 - Funcionamento geral do software



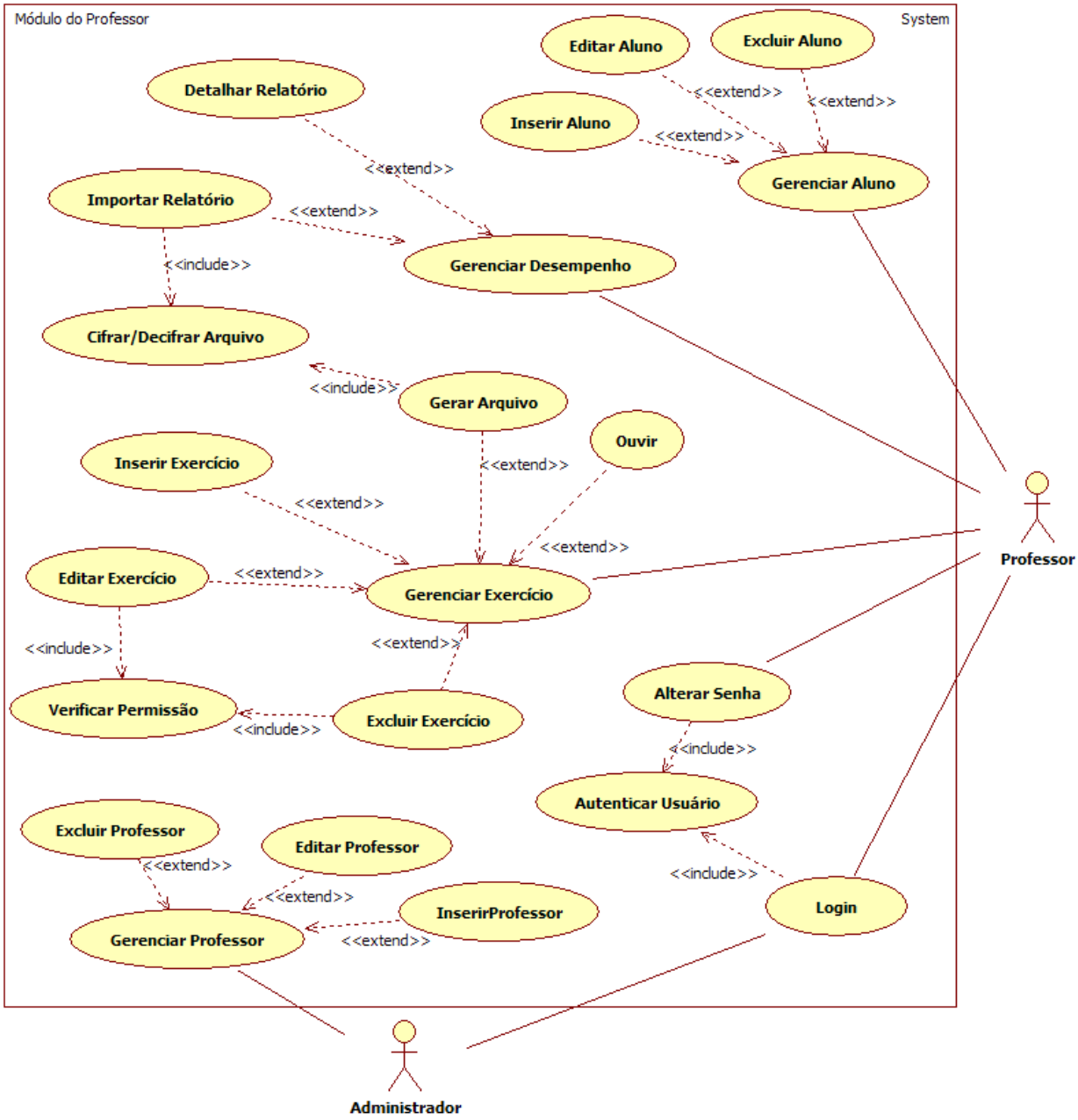
Fonte: Autoria Própria

No desenvolvimento do software, na disciplina de requisitos, foram gerados diagramas de casos de uso apresentados na Figura 12 e Figura 13.

Como verificamos nos diagramas de caso de uso, o *Zettai Onkan* é composto por dois módulos: módulo do professor e módulo do aluno.

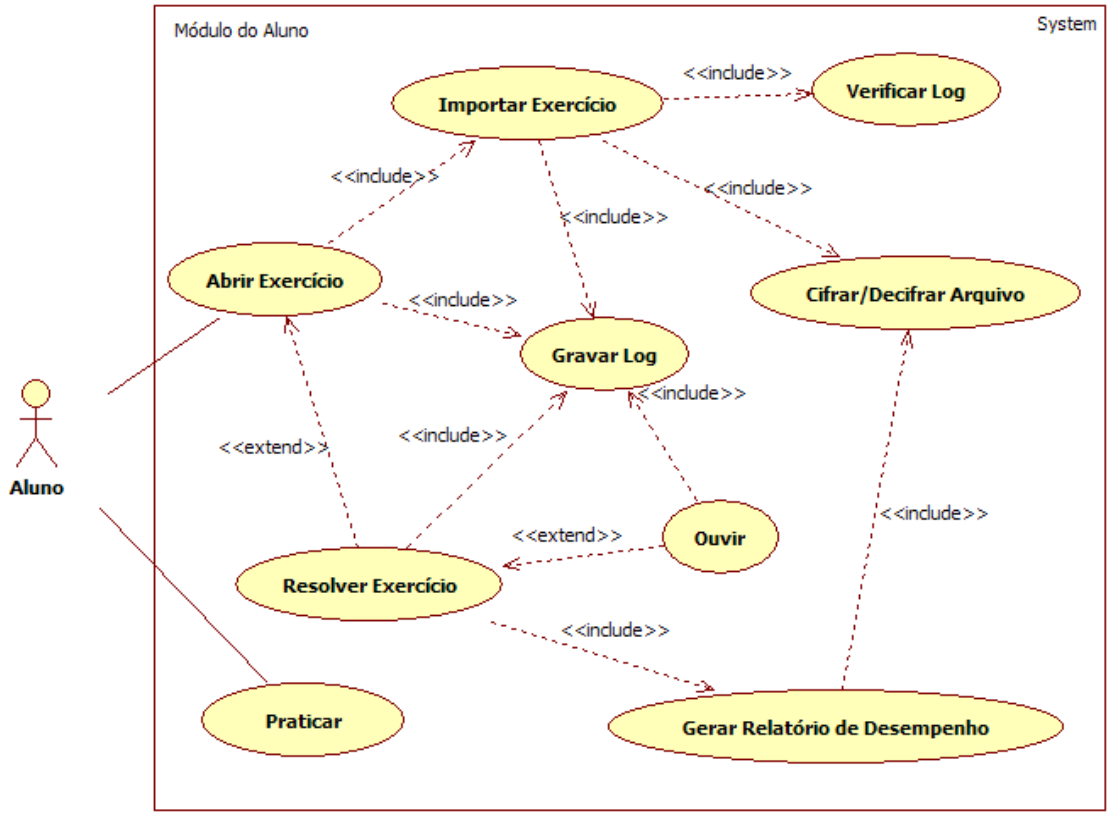
O motivo de criar dois módulos é aumentar a portabilidade do módulo do aluno, que não necessita de banco de dados e não precisa de um controle de acesso, podendo ser utilizado por qualquer pessoa que possua o módulo e um exercício para resolver.

Figura 12 - Caso de Uso - Módulo do Professor



Fonte: Autoria própria

Figura 13- Caso de Uso - Módulo do Aluno



Fonte: Autoria própria

5.1 Módulo do professor

O módulo do professor permite a criação de exercícios, cadastro de alunos e acompanhamento de desempenho do aluno. Possui banco de dados para armazenamento de exercícios e dados dos alunos.

5.1.1 Controle de acesso

No módulo do professor, ao iniciar o programa é exibida a tela de *login*, onde o professor deve entrar com o nome de usuário e senha na tela de *login*, conforme Figura 14. O cadastro do professor, só pode ser realizado através do *login* do administrador, usuário fixo pré-programado no software.

Figura 14 – Zettai Onkan - Login

Fonte: Autoria própria

A tela de *login* trata apenas as interações com o usuário. Todas as regras de acesso são controladas pela classe Porteiro, cujo *header* está ilustrado na Figura 15. Esta classe implementa o padrão de projeto *Singleton* e armazena dados do professor que está no sistema, tornando possível a verificação a qualquer momento do professor que está utilizando o software. Para a validação de usuário, a classe Porteiro utiliza a classe DAOProfessor, responsável por todas as operações da classe Professor com o banco de dados.

Na classe Porteiro, é utilizada a declaração __property, que torna possível o acesso às funções *get* e *set* através de um mesmo nome. Outro benefício de utilizar __property é a manutenção de código, pois em caso de alteração de acesso a um atributo público para utilização de função *get*, por exemplo, não necessitaria alterar todas as classes que utilizam este atributo.

Figura 15 – Header da classe Porteiro

```
#ifndef PorteiroH
#define PorteiroH
//---------------------------------------------------------------------------

#include "Conector.h"
#include "DAOProfessor.h"
#include "Professor.h"
#include "Configuracoes.h"

class Porteiro
{
    public:
        static Porteiro* getInstance();  //Singleton pattern
        bool PassarCartao(UnicodeString Usuario, UnicodeString Senha);
        __property Professor* ProfessorLogado = {read = FProfessor};
        __property UnicodeString Mensagem = {read = FMensagem};
        __property UnicodeString UsuarioMaster = {write = setUsuarioMaster};
        __property UnicodeString SenhaMaster = {write = setSenhaMaster};
        __property UnicodeString CaminhoDB = {write = setCaminhoDB, read = FCaminhoDB};
    protected:
        Porteiro();

    private:
        static Porteiro* _instance;
        UnicodeString FUsuarioMaster;
        UnicodeString FSenhaMaster;
        UnicodeString FMensagem;
        UnicodeString FCaminhoDB;
        Professor *FProfessor;

        void __fastcall setCaminhoDB(UnicodeString caminho);
        void __fastcall setUsuarioMaster(UnicodeString Usuario);
        void __fastcall setSenhaMaster(UnicodeString Senha);

};
#endif
```

Fonte: Autoria própria

5.1.2 Mensagens MIDI

Para gerar o áudio no *Zettai Onkan*, foi utilizado o módulo mmsystem.h, que oferece a interface para controle da unidade de multimídia do sistema operacional Windows 32-bit. Por esta razão, é provável que haja incompatibilidade com outros sistemas operacionais.

As mensagens MIDI, que são enviadas para o módulo mmsystem, são formadas por 32 bits. Porém, como a inserção de dados na mensagem é feita *byte* a *byte*, é necessário criar a união, ilustrada a Figura 16.

O primeiro byte da mensagem é de *status*, os dois seguintes são de dados, e o último byte não é utilizado.

Figura 16 - Mensagem Midi

```
union msg
{
    public:
        unsigned long word;
        unsigned char dado[4];
};
```

Fonte: Autoria própria

O byte de *status* é formado por dois valores hexadecimais. O primeiro valor hexadecimal especifica o tipo de mensagem. No software desenvolvido neste projeto foram utilizadas as mensagens do tipo *Note On* (nota pressionada), especificado pelo hexadecimal 9, *Note Off (*nota liberada), hexadecimal 8 e *Program Change* (alteração de instrumento), hexadecimal C. O segundo valor hexadecimal especifica para qual canal será enviada a mensagem. O protocolo MIDI suporta o envio de mensagens sobre 16 canais independentes, ou seja, é possível ouvir 16 instrumentos ou sons simultaneamente. No software desenvolvido neste projeto foi utilizado o canal 0 para melodia e o canal A (10) para sons de ritmo.

Para centralizar as operações com mensagens MIDI, foi desenvolvida a classe Miji, que implementa o padrão *Singleton* para manter as configurações armazenadas durante a execução do software, como volume e instrumento.

Para executar uma nota musical isoladamente, a classe Miji implementa o método playNota, descrita na Figura 17.

Figura 17 - Método playNota

```
void __fastcall Miji::playNota(int nota)
{
    msg som;
    FUltimaNota = nota;
    som.dado[0] = 0x90;
    som.dado[1] = nota;
    som.dado[2] = FVolumeMusica;
    midiOutShortMsg(device, som.word);
}
```

Fonte: Autoria própria

Mensagens do tipo *Note On* precisam de dois bytes de dados, sendo um descrevendo a altura da nota musical que será tocada e o outro a sua intensidade. Tanto a altura quanto a intensidade são representadas por sete bits, ou seja, 128 valores possíveis que variam de 0 a 127.

Percebe-se que nesta mensagem não há informação em relação à duração da nota a ser executada. Por esta razão, é necessário parar a execução de toda nota que for executada. Para interromper uma nota em execução, é utilizada mensagem do tipo *Note Off*. A implementação do método para parar uma nota na classe Miji está ilustrada na Figura 18.

Figura 18 - Método stopNota

```
void __fastcall Miji::stopNota(int nota)
{
    msg som ;
    som.dado[0] = 0x80;
    som.dado[1] = nota == 0? FUltimaNota: nota;
    midiOutShortMsg(device, som.word);
}
```

Fonte: Autoria própria

Para a execução de uma seqüência de notas foi implementada o método playCompasso, ilustrada na Figura 19. Este método se baseia em um *loop* que executa as notas musicais em seqüência, incrementando o tempo corrido para controlar a duração de cada nota, ou seja, verificando o tempo corrido é enviada a mensagem *Note On* ou *Note Off*. O controle do tempo de execução é realizado utilizando a função Sleep() da API do Windows. A função Sleep() suspende a execução da *thread* corrente durante o tempo determinado em milissegundos. Conforme está descrito no código da Figura 19, a cada *loop* o Windows aguarda 19 milissegundos para voltar a executar a *thread*.

Na música em geral, costuma-se utilizar o tempo (velocidade de execução de uma música) máximo em 208 batidas por minuto (BPM). Considerando que a menor duração possível no *Zettai Onkan* é a semicolcheia, que equivale a ¼ da unidade de tempo (semínima), conclui-se que a menor duração de notas equivale a (60 x 1/208 x 1/4) segundos, ou seja, aproximadamente 72 x 10^{-3} segundos. Por este motivo, cada *loop* necessita ser em um intervalo menor que 72 milissegundos. Logicamente, quanto menor o tempo de espera em cada *loop* maior a precisão, porém, existe um atraso devido ao tempo necessário para suspender e voltar a executar a *thread.* Após vários testes, verificou-se que em média ocorria o atraso de 1 milissegundo para cada chamada na função Sleep(). Levando-se estes fatores em conta, foi utilizado o tempo de 19 milissegundos na função Sleep() equivalendo a 20 milissegundos incrementados em cada *loop*.

Figura 19 - Método playCompasso

```
void __fastcall Miji::playCompasso(Compasso *compasso)
{
    msg Clicks;
    msg Notas;
    int TempoCorrido = 0;
    int SemicolcheiaMili;
    int PosicaoSemicolcheia;
    SemicolcheiaMili = TempoEmMili/4;
    Clicks.dado[2] = FVolumeClick;
    Notas.dado[2] = FVolumeMusica;

    while (true)
    {
        if (TempoCorrido % SemicolcheiaMili == 0) //Tempo de Semicolcheia
        {
            PosicaoSemicolcheia = TempoCorrido/SemicolcheiaMili;
            if (PosicaoSemicolcheia == (compasso->Formula+2)*4) //Fim de Compasso
            {
                //Silenciar canais
                Notas.dado[0] = 0x80;
                midiOutShortMsg(device, Notas.word);
                Clicks.dado[0] = 0x8A;
                midiOutShortMsg(device, Clicks.word);
                break;
            }
            else if (compasso->getNota(PosicaoSemicolcheia) != NULL)
            {
                Notas.dado[0] = 0x80;        //Silencia nota anterior
                midiOutShortMsg(device, Notas.word);

                if (compasso->getNota(PosicaoSemicolcheia)->Positiva)
                {
                    Notas.dado[0] = 0x90;
                    Notas.dado[1] = compasso->getNota(PosicaoSemicolcheia)->Tom;
                    midiOutShortMsg(device, Notas.word);
                }
            }
            if (TempoCorrido%TempoEmMili == 0)          // Clicks
            {
                Clicks.dado[0] = 0x8A;
                midiOutShortMsg(device, Clicks.word);
                Clicks.dado[0] = 0x9A;
                if ((TempoCorrido/TempoEmMili) % (compasso->Formula+2) == 0 )
                    Clicks.dado[1] = 69;
                else
                    Clicks.dado[1] = 65;
                midiOutShortMsg(device, Clicks.word);
            }
        }
        Application->ProcessMessages();

        Sleep(19);
        TempoCorrido+=20;

        if (!Continuar)
        {
            Clicks.dado[0] = 0x8A;
            midiOutShortMsg(device, Clicks.word);
            Notas.dado[0] = 0x80;
            midiOutShortMsg(device, Notas.word);
            break;
        }
    }
}
```

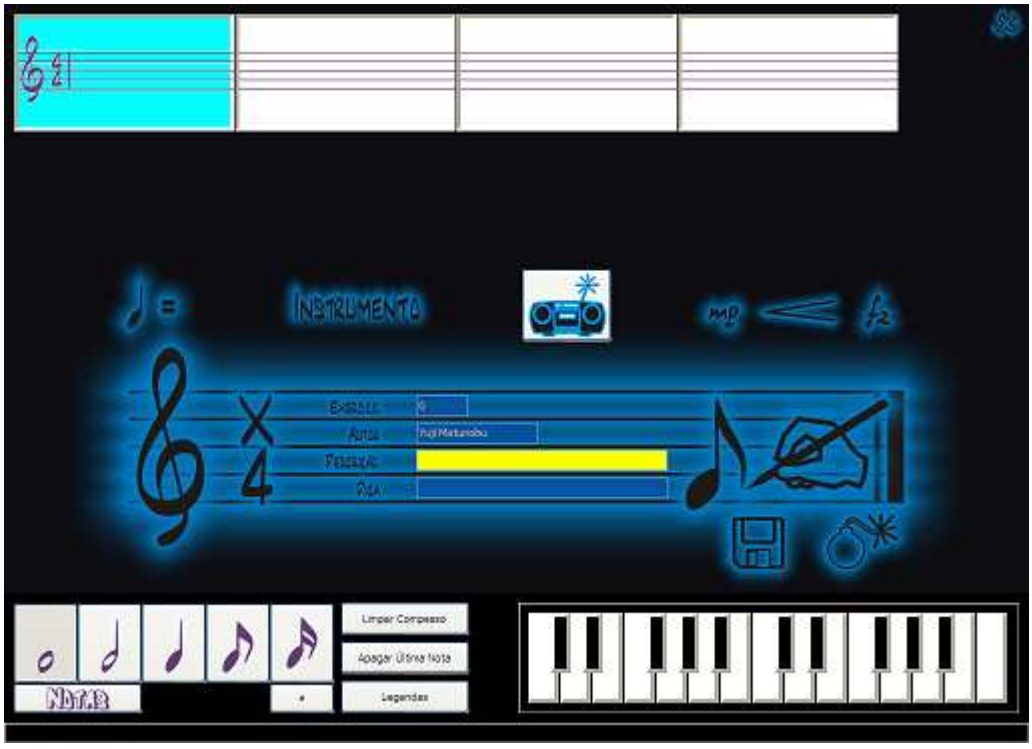
Fonte: Autoria própria

Existem diversos componentes prontos para execução de arquivos midi. Porém não permitem a alteração do tempo (velocidade de execução) nem de instrumento ou volume. Outro problema é que utilizando tais componentes, necessitaria converter as notas musicais inseridas pelo professor em um arquivo midi a cada vez que o professor desejasse ouvir antes de concluir a edição do exercício, o que certamente diminuiria o desempenho.

5.1.3 Editor de exercício

A tela de edição de exercício, ilustrada na Figura 20, é a mais complexa do software. É nesta tela que o professor irá criar os exercícios de percepção musical. Nesta tela, quando o professor insere uma nova nota musical no exercício, a partitura é redesenhada e o áudio desta nota é executado.

Figura 20 - Zettai Onkan - Editor de Exercícios



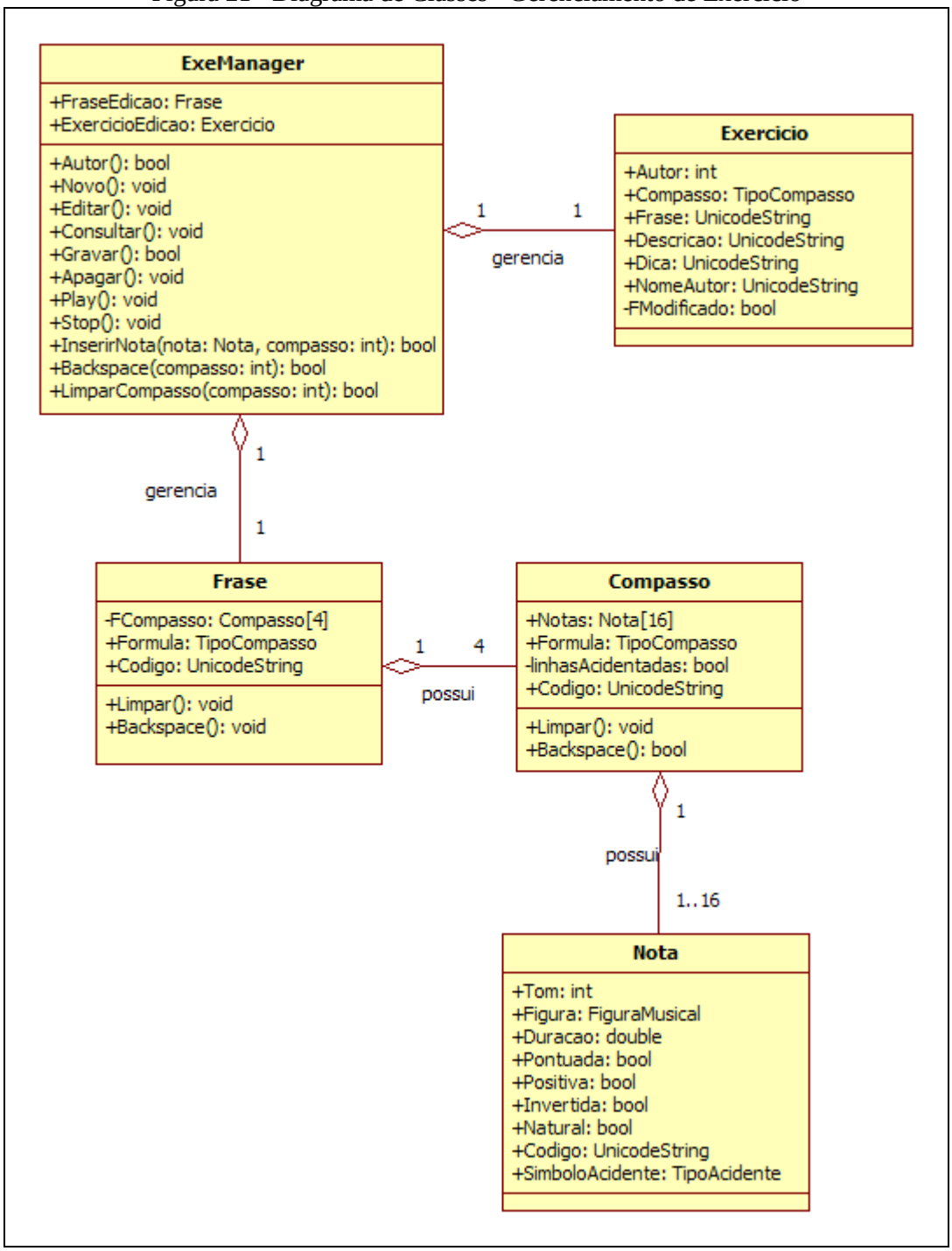
Fonte: Autoria própria

O gerenciamento dos exercícios é realizado pela classe ExeManager, que controla a persistência dos exercícios através da classe DAOExercício. Além disso, a classe ExeManager

possui em seus atributos uma agregação de um objeto da classe Frase, que armazena o exercício recuperado do banco de dados, ou do exercício em edição pelo professor.

A classe Frase contém uma agregação de quatro objetos da classe Compasso, e este uma agregação de até 16 objetos da classe Nota, conforme diagrama de classes da Figura 21.

Figura 21 - Diagrama de Classes - Gerenciamento de Exercício



Fonte: Autoria própria

5.1.4 Gerar arquivo de exercício

Através da tela de gerar arquivos, ilustrada na Figura 22, o professor pode exportar um exercício cadastrado no banco de dados. Esta exportação é necessária para que o aluno possa realizar o exercício sem a necessidade de possuir o banco de dados.

Figura 22 - Zettai Onkan - Gerar Arquivo para Aluno

Fonte: Autoria própria

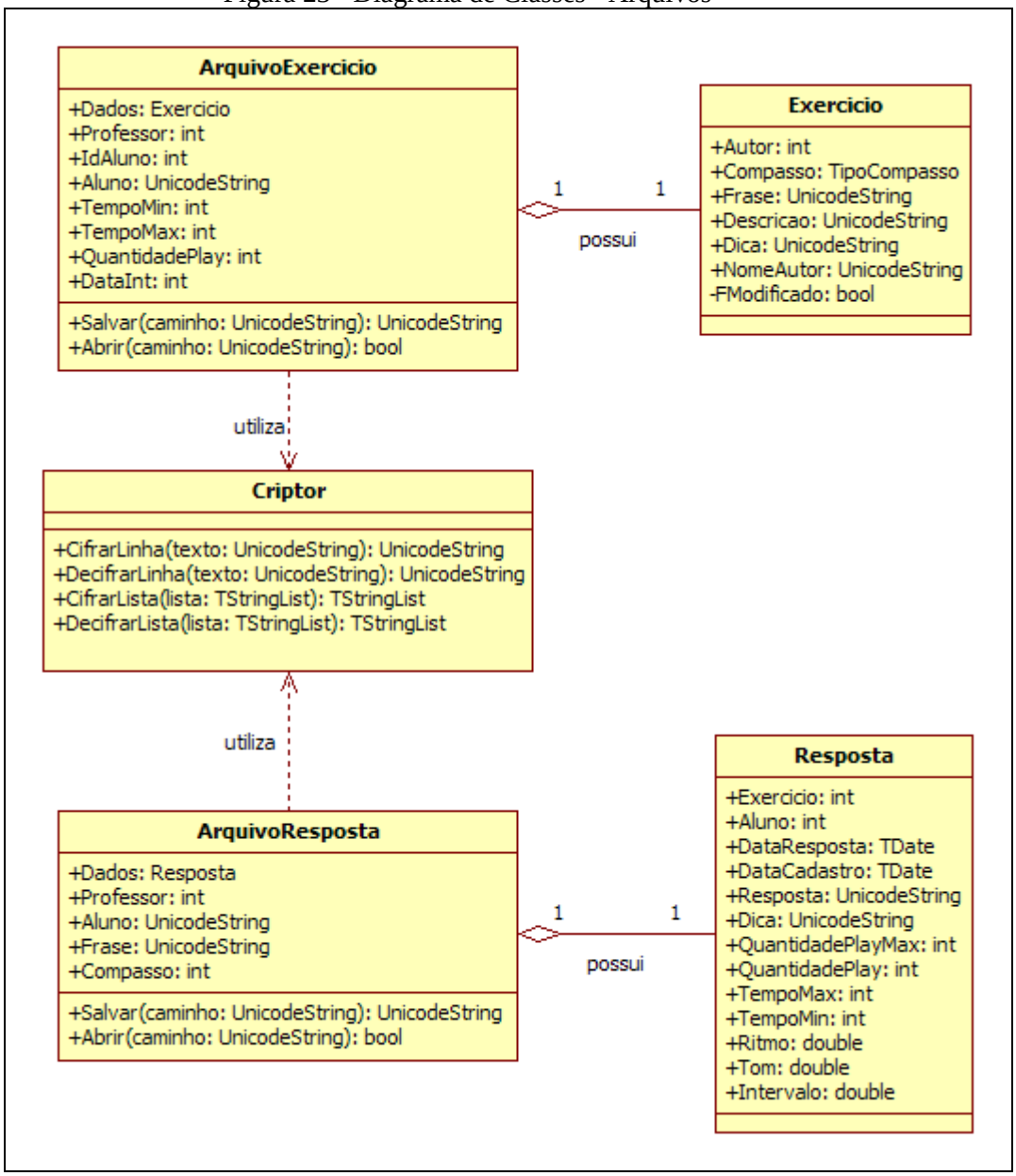
Para a conversão de arquivo para objeto e vice-versa, foi desenvolvida a classe ArquivoExercicio, cuja finalidade é abrir e salvar arquivos de exercício.

A classe ArquivoExercicio possui uma agregação da classe Exercicio, além de informações adicionais que o professor pode inserir, como limite de execução do áudio e restrição de alteração de tempo.

A indicação do aluno para qual o exercício está sendo gerado é obrigatória, pois a resposta gerada pelo aluno conterá esta informação para que possa ser vinculado com o cadastro do aluno no momento da importação da resposta.

O arquivo é gerado em formato texto, porém, seus dados são cifrados. Para as funções de cifrar e decifrar arquivos a classe ArquivoExercicio utiliza a classe Criptor, conforme ilustra o diagrama de classes da Figura 23.

Figura 23 - Diagrama de Classes - Arquivos



Fonte: Autoria própria

A classe Criptor inverte a ordem das linhas do arquivo e cifra cada linha utilizando algoritmo de substituição, cujo método está descrito na Figura 24. Este algoritmo, além substituir cada caractere por outra de acordo com a quantidade total de caracteres na linha, ele também inverte a ordem dos caracteres.

Apesar de ser um algoritmo simples de cifração, evita que o aluno altere as restrições impostas pelo professor, e garante a originalidade da resposta do aluno.

Figura 24 - Método CifrarLinha

```
UnicodeString __fastcall Criptor::CifrarLinha(UnicodeString texto)
{
    int i, tamanho, cod, diferenca;
    UnicodeString toxte;
    char letra;
    tamanho = texto.Length();
    toxte = texto;

    diferenca = tamanho;
    diferenca %= 95;
    for (i = 1; i <= tamanho; i++)
    {
        letra = texto[tamanho+1-i];
        cod = letra;
        cod += diferenca;
        if (cod > 126)
        {
            cod = cod - 95;
        }
        letra = char(cod);
        toxte[i] = letra;
    }
    return toxte;
}
```

Fonte: Autoria própria

5.1.5 Importar arquivo de resposta

Para importar as respostas do aluno para o banco de dados é utilizada a classe ArquivoResposta, que possui em seus atributos uma agregação de um objeto da classe Resposta. A classe resposta contém informações sobre o desempenho do aluno na realização do exercício.

Da mesma forma que a classe ArquivoExercício, a classe ArquivoResposta utiliza a classe Criptor para cifrar e decifrar os arquivos, conforme o ilustra diagrama de classes da Figura 23.

5.2 Módulo do aluno

O módulo do aluno permite executar exercícios enviados pelo professor, gerando o respectivo relatório de desempenho após o término. Não precisa ser instalado, podendo ser

executado de qualquer mídia que permita gravação, como dispositivos de memória flash (pendrives).

Este módulo é relativamente simples, possuindo apenas a funcionalidade de realizar o exercício e praticar. O objetivo da funcionalidade de praticar é deixar o aluno habituado com a interface de edição de partitura, pois para resolver os exercícios será através da mesma interface.

5.2.1 Realizar exercício

Para realizar um exercício, o aluno deve inicialmente importar um arquivo de exercício gerado pelo professor. A interface inicial do módulo do aluno é simples, com poucos botões auto-explicativos, ilustrada na Figura 25.

Figura 25 - Zettai Onkan - Tela Inicial do Módulo do Aluno

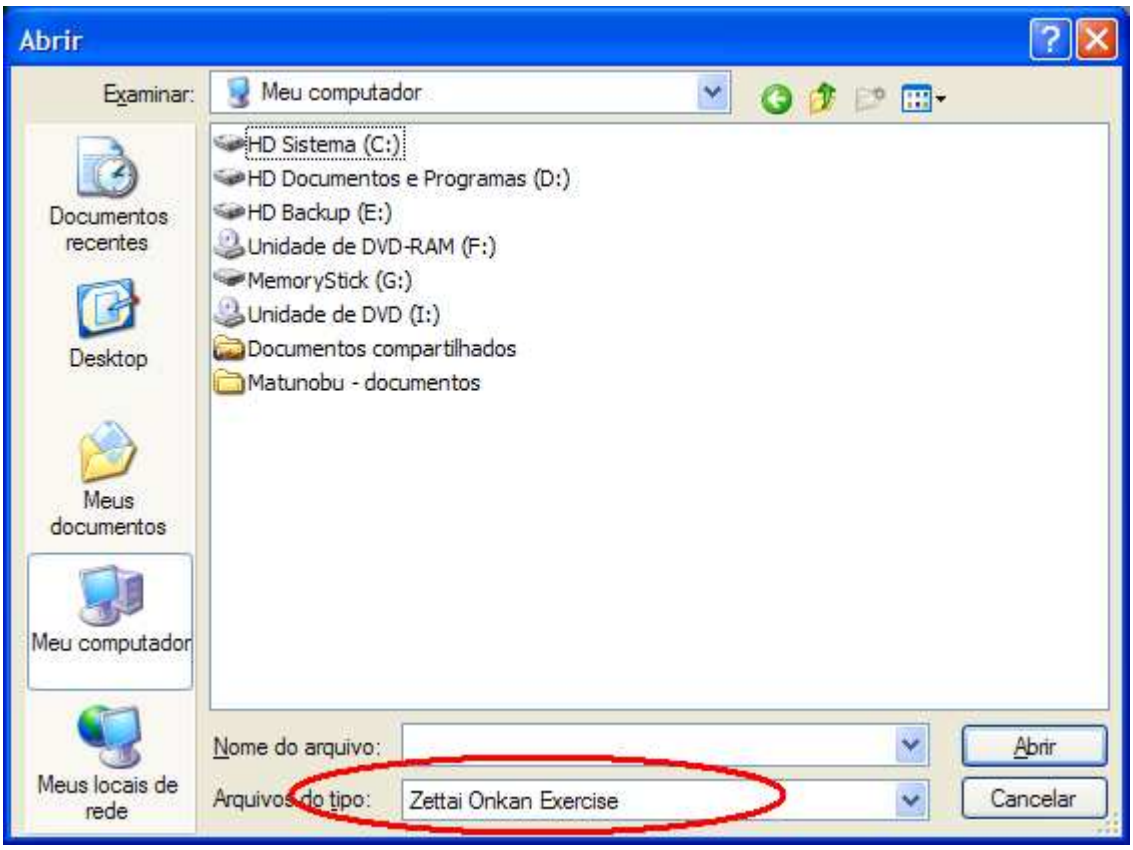
Fonte: Autoria própria

Ao selecionar a opção de abrir exercício, será exibida a caixa de seleção de arquivo que filtra apenas os arquivos suportados pelo programa, conforme ilustra a Figura 26. Como o filtro é baseado na extensão do arquivo, após a seleção, o software verifica o cabeçalho do arquivo para confirmar a compatibilidade do mesmo.

O módulo do aluno, apesar de não utilizar banco de dados, armazena informações necessárias para que as funcionalidades operem corretamente. Para esta persistência, é

utilizado um *log* de dados, armazenados em formato texto, a fim de registrar eventos realizados pelo aluno.

Figura 26 - Zettai Onkan - Abrir Exercício



Fonte: Autoria própria

Uma das informações armazenadas é a criação do relatório de desempenho, ao final de um exercício. Esta informação é verifica no momento da importação de arquivo, evitando que o aluno realize várias vezes o mesmo exercício até obter um desempenho alto, tornando o relatório de desempenho inválido. Considerando que os arquivo de exercícios são nominais, ou seja, possui em seus dados o aluno a quem se destina o exercício, é possível que vários alunos realizem o mesmo exercício, porém, não o mesmo arquivo de exercício.

Outra informação armazenada é a quantidade de vezes que o aluno já ouviu o exercício. O exercício se baseia em ouvir o exercício e transcrever em forma de partitura, e o professor pode limitar a quantidade de vezes permitida para a execução do áudio. Portanto, mesmo que o aluno interrompa a resolução do exercício, ao importar novamente o mesmo arquivo a informação da quantidade de vezes em que o aluno ouviu o exercício está persistido.

Na tela de resolução de exercícios, ilustrada na Figura 27, quando o aluno acionar o botão de ouvir a frase musical, é verificada se existe algum gravador de áudio em execução. Esta verificação ocorre através do nome de processos em execução no Windows, portanto, somente são verificados processos cujo nome é conhecido. Foram incluídos nesta verificação o gravador de áudio do Windows e mais três softwares populares de gravação de áudio (Audacity, Free Sound Recorder e EXP Audio Edit). O objetivo desta verificação é dificultar que o aluno armazene o áudio do exercício e execute ilimitadamente a frase musical.

Figura 27 - Zettai Onkan - Responder Exercício

Fonte: Autoria própria

5.2.2 Cálculo de desempenho

Para calcular o desempenho do aluno, foi criada a classe AnalisadorFrases que implementa métodos de calcular a igualdade de tons, ritmos e intervalos.

Considerando uma frase musical uma seqüência de notas e silêncios, para cada método foi utilizado uma generalização destas notas, para que o erro cometido em uma percepção não influencie em outra. Estas generalizações foram necessárias pois as notas são analisadas em seqüência, e em determinados casos, a nota equivalente pode não estar na mesma posição.

Na análise da percepção do tom a duração das notas não é considerada. Porém, para desconsiderar a duração, em caso de duas ou mais notas de silêncio consecutivas, é necessário considerar como uma nota apenas de silêncio, para que o tom de uma nota não seja comparado com uma pausa não identificada.

Na análise da percepção rítmica, o tom das notas é desconsiderado e as pausas em seqüência são unidas em uma única nota e suas durações somadas.

Na análise da percepção de intervalos a duração das notas não é considerada e todas as pausas são excluídas, formando uma seqüência apenas de notas positivas. Relembrando que intervalo é a diferença entre alturas de duas notas, a comparação é realizada utilizando o valor absoluto da diferença de uma nota com a sua anterior. Por exemplo, se a seqüência correta das notas for Ré-Mi-Fá, e o aluno responder Lá-Si-Dó, a percepção de intervalo do aluno é 100%, pois o intervalo entre Ré e Mi equivale a 1 tom, e entre Lá e Si também. O mesmo ocorre entre Mi e Fá, e entre Si e Dó, onde o intervalo equivale a 1 semitom.

CONCLUSÕES

Um software educativo de exercício e prática em nenhum momento propiciará a qualidade e dinâmica que um bom professor pode propiciar. No caso das Artes em especial, como a música, em que existe a sensibilidade e musicalidade dos indivíduos, o computador está longe de possuir capacidade de substituir o professor. Porém, o professor não pode estar a todo momento junto de seu aluno, com a exceção de família de músicos.

Com o auxílio de um software para o estudo de música, o professor pode suprir esta falta de tempo, podendo se concentrar em atividades em que a presença do professor é indispensável, como a correção de técnicas práticas.

Este projeto teve como objetivo o desenvolvimento de um software educativo para treinamento em percepção musical. Utilizando o processo de desenvolvimento de software Open UP, foi possível desenvolver o software de forma ágil sem perder o controle das atividades desenvolvidas. Além disso, por permitir customização no processo, muitas atividades voltadas para o desenvolvimento em equipe puderam ser descartadas.

A utilização do Open UP possibilitou também a alteração nos requisitos do software sem causar grandes preocupações, pois seu desenvolvimento incremental, que gerou um executável a cada nova funcionalidade inserida, possibilitava o retorno imediato e, conseqüentemente, correção de erros e alterações antes que este causasse outros problemas.

O software desenvolvido neste projeto possui interface amigável, com imagens intuitivas e teclas de atalho para que tanto o professor quanto o aluno não sinta dificuldade na acessibilidade à funcionalidade desejada.

Para um projeto futuro é interessante desenvolver uma interface para que o software possa ser executado na internet, por meio de um navegador. Desta forma, os arquivos seriam armazenados no servidor da página, eliminando a necessidade de troca de arquivos entre professor e alunos.

No desenvolvimento de um software educativo, é importante visar o baixo custo do produto. Porém, não apenas o custo de seu desenvolvimento, mas também nos requisitos, como o sistema operacional. Portanto, é interessante também, para um projeto futuro, criar versões para sistemas operacionais gratuitos, para que ocorra uma maior disseminação do software.

REFERÊNCIAS

CARDOSO, B.; MASCARENHAS, M. **Curso completo de teoria musical e solfejo** 1. Vol. 8. ed. Rio de Janeiro: Irmãos Vitale S. A., 1973.

DAMIAN, C. M. O ouvido absoluto e o ouvido relativo: vantagens e desvantagens dentro da educação musical. **Revista Arte-Online** v. 2, nov.1999/fev. 2000. Disponível em: < http://www.ceart.udesc.br/Revista_Arte_Online/Volumes/artclaudia.htm>. Acesso em: 01 set. 2010.

DENARDI, C. **A educação musical e o uso do software educativo-musical.** [S.l: S.n., s.d]. Disponível em: < http://www.editoraopet.com.br/main.asp?Team={189EE9BA-E2ED-42D8-BD9C-7C84F102EC9E}>. Acesso em: 11 fev. 2010.

FISCHER, M. C. B. O. **Estudo de requisitos para um software educativo de apoio ao ensino da introdução à computação**. 2001. Dissertação (Mestrado em Ciência da Computação) - USP, São Paulo, 2001.

GIRAFFA, L. M. M. **Uma arquitetura de tutor utilizando estados mentais**. 1999. Tese (Doutorado em Ciência da Computação) - UFRGS, Porto Alegre, 1999. Disponível em: < http://hdl.handle.net/10183/17620>. Acesso em: 11 fev. 2010.

GOLDEMBERG, R.; OTUTUMI, C. Análise de conteúdo segundo Bardin: procedimento metodológico utilizado na pesquisa sobre a situação atual da percepção musical nos cursos de graduação em música do Brasil. In: Simpósio de Cognição e Artes Musicais, 4., 2008, São Paulo. **Anais do SIMCAM4.** São Paulo: Paulistana, 2008**.** Disponível em: < http://www.fflch.usp.br/dl/simcam4/downloads_anais/SIMCAM4_Ricardo_Goldemberg_e_Cristiane_Otutumi.pdf>. Acesso em: 01 set. 2010.

HOLST, I. **ABC da música**. 2. ed. São Paulo: Martins Fontes, 1998.

MED, B. **Teoria da música** 3. ed. Brasília: Musimed, 1986.

MEIRA, F. L. **Open UP:** processo unificado aberto. [S.l: S.n., 2010]. Disponível em: <http://open2up.blogspot.com>. Acesso em: 03 mai. 2010.

MEIRELLES, F. S. **Informática:** novas aplicações com microcomputadores. 2. ed. aum. e atual. São Paulo, Pearson, 2004.

MILETTO, E. M. et al. Educação musical auxiliada por computador: algumas considerações e experiências. Porto Alegre, 2004. Disponível em: <http://www.cinted.ufrgs.br/renote/mar2004/artigos/09-educacao_musical.pdf>. Acesso em: 02 nov. 2010.

PFLEEGER, S. L. **Engenharia de software:** teoria e pratica. 2. ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, Roger S. **Engenharia de software.** 5. ed. Rio de Janeiro: McGraw-Hill, 2002.

SANTOS, Sandra S. **OpenUP:** um processo ágil. [S.l: S.n., 2009]. Disponível em: <http://www.ibm.com/developerworks/br/rational/local/open_up/index.html>. Acesso em: 03 mai. 2010.

SOMMERVILLE, Ian. **Engenharia de software.** 8. ed. São Paulo: Pearson Addison - Wesley, 2007.

STERNBERG, R. J. **Psicologia cognitiva**. Porto Alegre: Artmed, 2000.

VALENTE, J. A. **Diferentes usos do computador na educação**. In J. A. Valente (org.), *Computadores e Conhecimento: repensando a educação* (pp.1-23). Campinas, SP: Gráfica da UNICAMP, 1993.

VANZELLA, P.; OLIVEIRA, M. G. M; WERKE, M. Incidência e categorização de ouvido absoluto em estudantes de música da Universidade de Brasília. In: Simpósio de Cognição e Artes Musicais, 4., 2008, São Paulo. **Anais do SIMCAM4.** São Paulo: Paulistana, 2008**.** Disponível em: <http://www.fflch.usp.br/dl/simcam4/downloads_anais/SIMCAM4_ PVanzella_MGOliveira_MWerke.pdf>. Acesso em: 01 set. 2010.

APÊNDICE A – Implementação da classe AnalisadorFrase

A seguir, será apresentada a implementação da classe AnalisadorFrase, responsável por gerar a pontuação em cada uma das três percepções analisadas: tom, ritmo e intervalo. Seu método construtor recebe duas frases musicais como parâmetro, que serão utilizadas nas três análises.

```
//---------------------------------------------------------------------------
#pragma hdrstop
#include "AnalisadorFrases.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
AnalisadorFrases::AnalisadorFrases(Frase* Pergunta, Frase* Resposta)
{
   FrPergunta = new Frase();
   FrResposta = new Frase();
   FrPergunta->Codigo = Pergunta->Codigo;
   FrResposta->Codigo = Resposta->Codigo;
   Formula = Pergunta->Formula;
}
//---------------------------------------------------------------------------
double __fastcall AnalisadorFrases::getIgualdadeTom()
{
   bool UltimaSilencioP = false;
   bool UltimaSilencioR = false;
   Nota *notaP, *notaR;
   double Porcentagem;
   int TonsP, TonsR, TonsA, TonsCertos;
   TonsP = TonsR = 0;

   Nota* FPergunta[64];
   Nota* FResposta[64];

   for (int c = 0; c < 4; c++)
   {
      for(int n = 0; n < (Formula+2)*4; n++)
      {
         notaP = FrPergunta->getCompasso(c)->getNota(n);
         notaR = FrResposta->getCompasso(c)->getNota(n);
         if (notaP != NULL)
         {
            if (!notaP->Positiva)
            {
               if (!UltimaSilencioP)
               {
                  FPergunta[TonsP] = new Nota();
                  notaP->CopiarPara(FPergunta[TonsP]);
                  TonsP++;
                  UltimaSilencioP = true;
               }
            }
            else
            {
               FPergunta[TonsP] = new Nota();
               notaP->CopiarPara(FPergunta[TonsP]);
```

```
                TonsP++;
            }
        }
        if (notaR != NULL)
        {
            if (!notaR->Positiva)
            {
                if (!UltimaSilencioR)
                {
                    FResposta[TonsR] = new Nota();
                    notaR->CopiarPara(FResposta[TonsR]);
                    TonsR++;
                    UltimaSilencioR = true;
                }
            }
            else
            {
                FResposta[TonsR] = new Nota();
                notaR->CopiarPara(FResposta[TonsR]);
                TonsR++;
            }
        }
    }
  }
  TonsA = TonsP;
  TonsCertos = 0;
  if (TonsR < TonsP) TonsA = TonsR;
  for (int i = 0; i < TonsA; i++)
  {
    if (FPergunta[i]->Tom == FResposta[i]->Tom)
      TonsCertos++;
  }
  Porcentagem = (double)TonsCertos/(double)TonsA *100;
  return Porcentagem;
}
//---------------------------------------------------------------------------
double __fastcall AnalisadorFrases::getIgualdadeIntervalo()
{
  Nota *notaP, *notaR;
  double Porcentagem;
  int TonsP, TonsR, Intervalos, IntervalosCertos;
  int TomP, TomR;
  TonsP = TonsR = 0;

  Nota* FPergunta[64];
  Nota* FResposta[64];

  for (int c = 0; c < 4; c++)
  {
    for(int n = 0; n < (Formula+2)*4; n++)
    {
      notaP = FrPergunta->getCompasso(c)->getNota(n);
      notaR = FrResposta->getCompasso(c)->getNota(n);
      if (notaP != NULL)
      {
        if (notaP->Positiva)
        {
          FPergunta[TonsP] = new Nota();
          notaP->CopiarPara(FPergunta[TonsP]);
          TonsP++;
```

```
            }
        }
        if (notaR != NULL)
        {
            if (notaR->Positiva)
            {
                FResposta[TonsR] = new Nota();
                notaR->CopiarPara(FResposta[TonsR]);
                TonsR++;
            }
        }
    }
}

Intervalos = TonsP - 1;
if (TonsR < TonsP) Intervalos = TonsR - 1;
TomP = TomR = 0;
IntervalosCertos = 0;
for (int i = 0; i <= Intervalos; i++)
{
    if (i == 0)
    {
        TomP = FPergunta[i]->Tom;
        TomR = FResposta[i]->Tom;
    }
    else
    {
        if (abs(FPergunta[i]->Tom - TomP) == abs(FResposta[i]->Tom - TomR))
            IntervalosCertos++;
        TomP = FPergunta[i]->Tom;
        TomR = FResposta[i]->Tom;
    }

}
Porcentagem = (double)IntervalosCertos/(double)Intervalos *100;
return Porcentagem;
}
//---------------------------------------------------------------------------
double __fastcall AnalisadorFrases::getIgualdadeRitmo()
{
    bool UltimaSilencioP = false;
    bool UltimaSilencioR = false;
    Nota *notaP, *notaR;
    double Porcentagem;
    int TonsP, TonsR, Tons, RitmoCerto;

    TonsP = TonsR = 0;

    double FPergunta[64];
    double FResposta[64];
    for (int i = 0; i < 64; i++)
    {
        FPergunta[i] = 0;
        FResposta[i] = 0;
    }

    for (int c = 0; c < 4; c++)
    {
        for(int n = 0; n < (Formula+2)*4; n++)
        {
```

```
            notaP = FrPergunta->getCompasso(c)->getNota(n);
            notaR = FrResposta->getCompasso(c)->getNota(n);
            if (notaP != NULL)
            {
               if (!notaP->Positiva)
               {
                  if (UltimaSilencioP)
                  {
                     FPergunta[TonsP-1] += notaP->Duracao;
                  }
                  else
                  {
                     FPergunta[TonsP] = notaP->Duracao;
                     TonsP++;
                     UltimaSilencioP = true;
                  }
               }
               else
               {
                  FPergunta[TonsP] = notaP->Duracao;
                  TonsP++;
                  UltimaSilencioP = false;
               }
            }
            if (notaR != NULL)
            {
               if (!notaR->Positiva)
               {
                  if (UltimaSilencioR)
                  {
                     FResposta[TonsR-1] += notaR->Duracao;
                  }
                  else
                  {
                     FResposta[TonsR] = notaR->Duracao;
                     TonsR++;
                     UltimaSilencioR = true;
                  }
               }
               else
               {
                  FResposta[TonsR] = notaR->Duracao;
                  TonsR++;
                  UltimaSilencioR = false;
               }
            }
         }
      }
      Tons = TonsP;
      if (TonsR < TonsP) Tons = TonsR;
      RitmoCerto = 0;
      for (int i = 0; i < Tons; i++)
      {
         if (FPergunta[i] == FResposta[i])
            RitmoCerto++;
      }
      Porcentagem = (double)RitmoCerto/(double)Tons *100;
      return Porcentagem;
   }
```

APÊNDICE B – Manual do software

ZETTAI ONKAN HELP SYSTEM

INTRODUÇÃO

Bem-Vindos ao Zettai Onkan!

- **O que é o Zettai Onkan?**

Zettai Onkan é um software educativo desenvolvido para alunos e professores de música, cuja finalidade é melhorar a percepção musical através de exercícios.

Este software não tem como objetivo utilizar o horário em aula, e sim, permitir a execução de exercício de percepção em qualquer outro ambiente.

Para tanto, o software é composto por dois módulos:

- Módulo do professor: Permite a criação de exercícios, cadastro de alunos e acompanhamento de desempenho do aluno. Possui banco de dados para armazenamento de exercícios e dados dos alunos.
- Módulo do aluno: Permite executar exercícios enviados pelo professor, gerando assim o respectivo relatório de desempenho.

- **Como funciona?**

O professor cria exercícios, frases musicais em quatro compassos, que ficarão armazenados no banco de dados. Estes exercícios ficarão compartilhados com outros professores, caso existam.

Quando o professor desejar aplicar um exercício, deverá gerar um arquivo a partir do exercício armazenado no banco de dados.

Este arquivo gerado, poderá então ser executado pelo aluno, utilizando o módulo do aluno, em qualquer computador que suporte a tecnologia MIDI.

Assim que o aluno conclui o exercício, será gerado o arquivo de desempenho, para que possa ser cadastrado no computador do professor, afim de ter o controle do desempenho do aluno.

- **O que significa "Zettai Onkan"?**

É um termo em japonês, utilizado como sinônimo de "Ouvido Absoluto".

Traduzindo ao "pé da letra" tem o significado de "Afinação Perfeita".

INSTALAÇÃO

Para instalar o módulo do professor, siga os seguintes passos:

1 - Execute o instalador do Zettai Onkan.

2 - Escolha o diretório para instalar o programa.

3 - Siga os passos até a finalização.

LOGIN

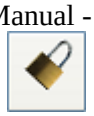
Ao iniciar o programa, será exibida a tela de *login* de acordo com a Figura 28:

Figura 28 - Manual - Tela de Login

Fonte: Autoria própria

Preencha os campos e clique no botão ilustrado na Figura 29.

Figura 29 - Manual - Botão Login



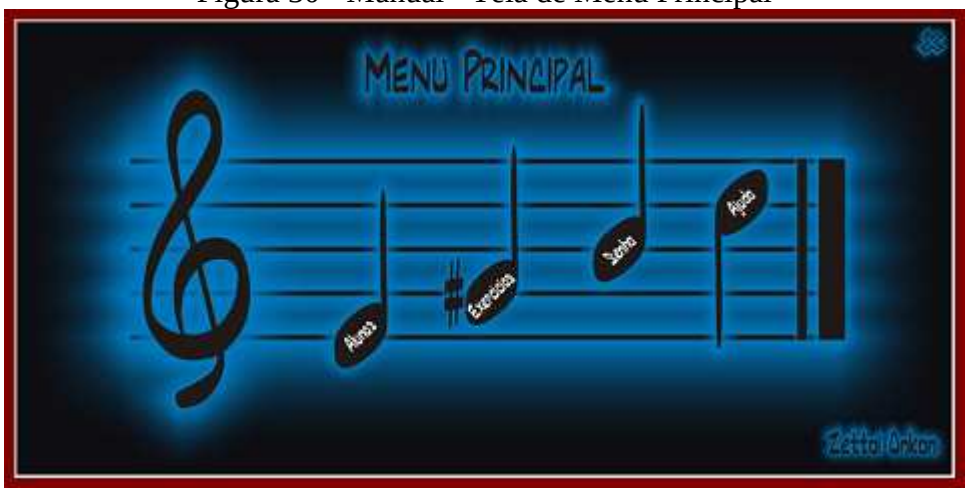
Fonte: Autoria própria

Caso não possua o nome de usuário e senha, fale com o administrador.

MENU PRINCIPAL

Após realizar o *Login*, será exibida a tela de menu principal, ilustrada na Figura 30.

Figura 30 - Manual - Tela de Menu Principal



Fonte: Autoria própria

É através do menu principal que o professor tem acesso às funcionalidades do programa.

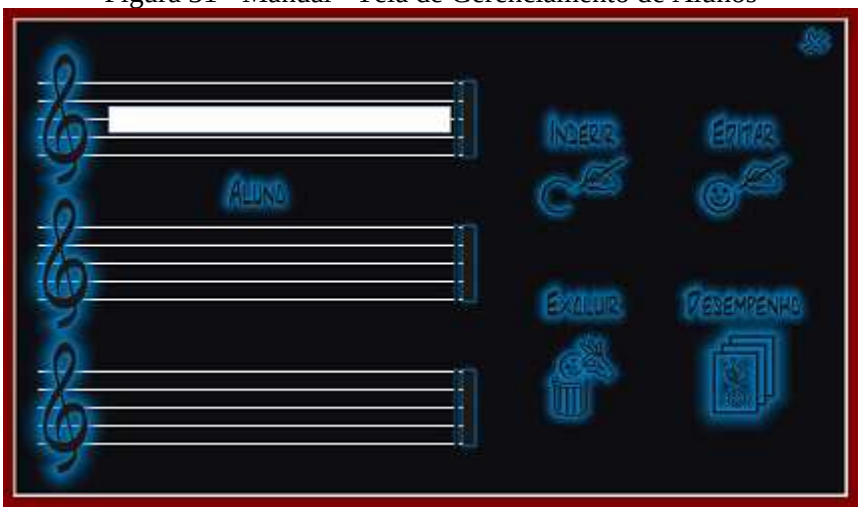
Existem os seguintes menus:

- Alunos - Acessa o banco de dados e localiza os alunos do professor que realizou o *login*. Informações sobre alunos de outros professores não são exibidos.
- Exercícios - Acessa o banco de dados e lista todos os exercícios cadastrados no sistema.
- Senha - Opção para alterar a senha.
- Ajuda - Abre este sistema de ajuda.

GERENCIAR ALUNOS

Na tela de menu principal, ao clicar em Alunos (ou tecla de atalho "A"), será exibida a tela de gerenciamento de aluno, ilustrada na Figura 31:

Figura 31 - Manual - Tela de Gerenciamento de Alunos



Fonte: Autoria própria

Nesta tela, o professor pode gerenciar os seus alunos.

Caso tenha alunos cadastrados, é exibida a relação em forma de lista. Para facilitar a busca, ao pressionar uma letra, os dados são filtrados. Para cancelar o filtro, basta deixar o campo de filtro em branco.

As opções são:

- Inserir - Abre um novo cadastro de aluno.
- Editar - Edita o aluno selecionado na grade de alunos.
- Excluir - Exclui o aluno selecionado na grade de alunos.
- Desempenho - Consulta os relatórios do aluno armazenados no banco de dados.

GERENCIAR RELATÓRIOS

Na tela de gerenciamento de alunos, se houver aluno selecionado, ao clicar em "Desempenho", será exibida a tela ilustrada na Figura 32.

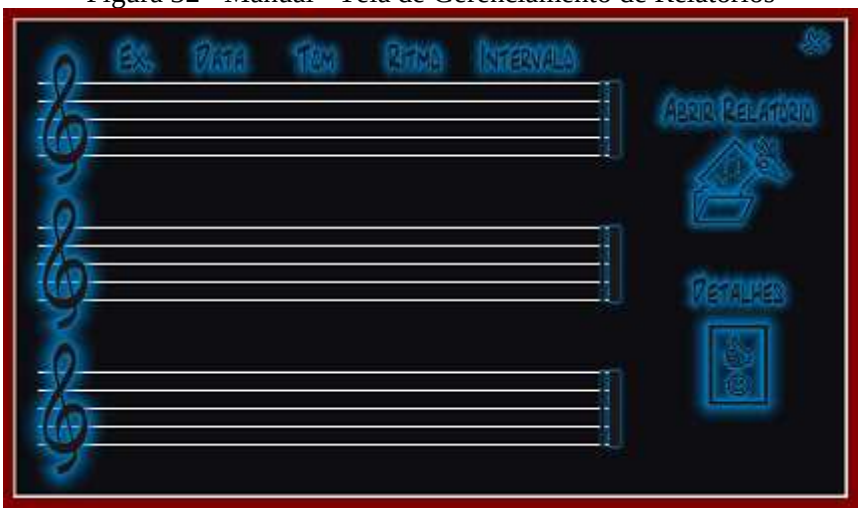
Nesta tela, o professor pode visualizar todos os exercícios realizados pelo aluno.

As opções são:

- Abrir Relatório - Após a realização de um exercício, o aluno irá levar o relatório ao professor. É por meio deste menu que o professor poderá abrir o arquivo de relatório, para armazenar no banco de dados e visualizar o desempenho do aluno.

- Detalhes - Exibe informações detalhadas sobre o relatório selecionado.

Figura 32 - Manual - Tela de Gerenciamento de Relatórios



Fonte: Autoria própria

INSERIR ALUNO

Na tela de gerenciamento de alunos, ao clicar em "Inserir", será exibida a tela de cadastro de aluno, ilustrada na Figura 33.

Figura 33 - Manual - Tela de Cadastro de Aluno

Fonte: Autoria própria

Para inserir um novo aluno, basta preencher o campo "Nome" e clicar no botão “Gravar”, ilustrado na Figura 34.

Figura 34 - Manual - Botão Gravar

Fonte: Autoria própria

Para cancelar, clique no botão “Cancelar”, ilustrado na Figura 35.

Figura 35 - Manual - Botão Cancelar

Fonte: Autoria própria

EDITAR ALUNO

Na tela de gerenciamento de alunos, se houver aluno selecionado, ao clicar em "Editar", será exibida a tela de cadastro de aluno, ilustrada na Figura 33.

Após realizar as alterações necessárias clique no botão ilustrado na Figura 34.

Para cancelar, clique no botão ilustrado na Figura 35.

EXCLUIR ALUNO

Na tela de gerenciamento de alunos, se houver aluno selecionado, ao clicar em "Excluir", será exibida uma caixa de diálogo, para confirmação da exclusão.

Ao excluir um aluno, os relatórios pertencentes a este aluno não poderão ser acessados.

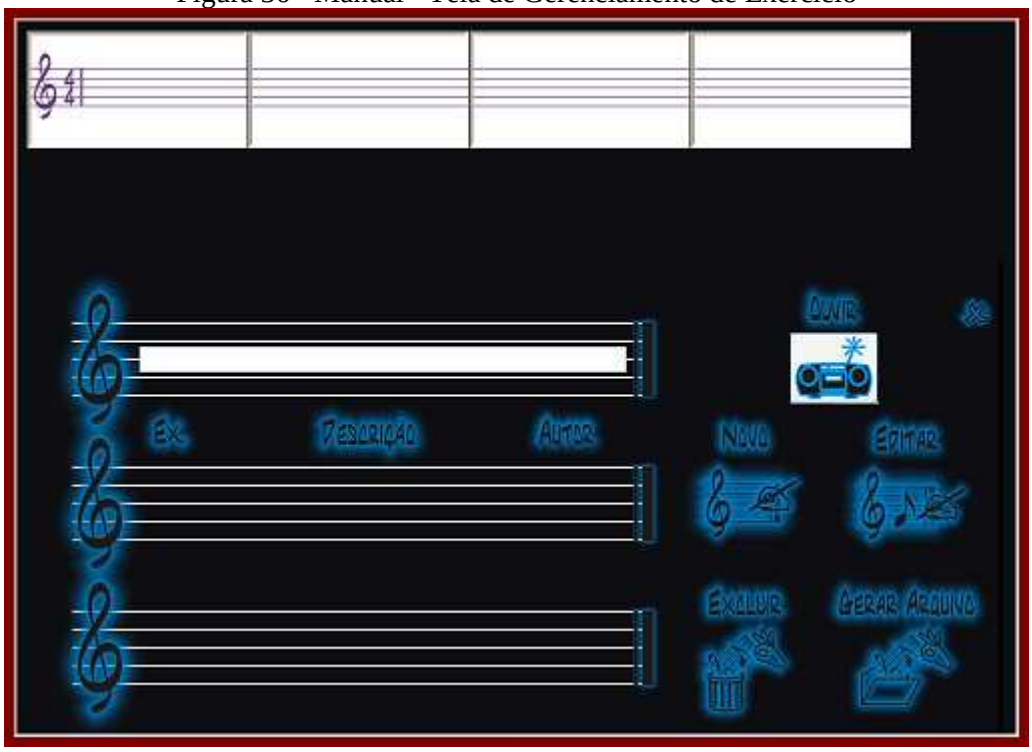
GERENCIAR EXERCÍCIOS

Na tela de menu principal, ao clicar em “Exercícios” (ou tecla de atalho "E"), será exibida a tela de gerenciamento de exercícios, ilustrada na Figura 36.

Nesta tela, o professor pode gerenciar os exercícios criados por ele, bem como visualizar e gerar arquivos de exercícios de outros professores.

Caso tenha exercícios cadastrados, é exibida a relação em forma de lista. Para facilitar a busca, ao pressionar uma letra, os dados são filtrados. Para cancelar o filtro, basta deixar o campo de filtro em branco.

Figura 36 - Manual - Tela de Gerenciamento de Exercício



Fonte: Autoria própria

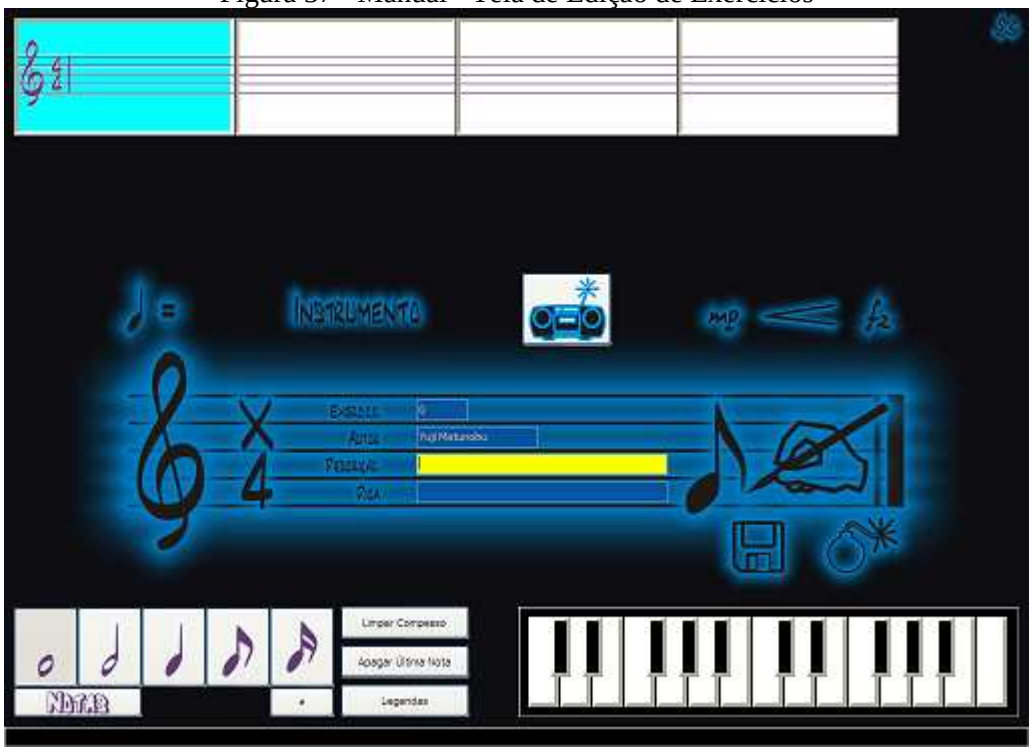
As opções são:

- ❖ Ouvir - O áudio do exercício selecionado é executado.
- ❖ Novo - Abre um novo cadastro de exercício.
- ❖ Editar - Caso o professor seja o autor do exercício selecionado, abre a tela de edição de exercícios.
- ❖ Excluir - Caso o professor seja o autor do exercício selecionado, exclui o exercício.
- ❖ Gerar Arquivo - Gera um arquivo para que o exercício possa ser executado pelo aluno.

EDIÇÃO DE EXERCÍCIOS

Na tela de gerenciamento de exercícios, ao clicar em "Novo" ou, se houver exercício selecionado clicar em "Editar" (neste caso se o professor for o autor do exercício), será exibida a tela de edição de exercícios, ilustrada na Figura 37.

Figura 37 - Manual - Tela de Edição de Exercícios



Fonte: Autoria própria

É por meio desta tela que o professor irá criar os exercícios a serem aplicados aos alunos.

A tela pode estar em dois modos de edição:

- Edição de dados - Para editar a "Descrição" e a "Dica" do exercício;
- Edição de partitura - Para editar o exercício propriamente dito.

Para alternar entre os modos de edição clique no botão ilustrado na Figura 38 ou utilize a tecla de atalho Ctrl + Tab.

Figura 38 - Manual - Botão Alternar Modos

Fonte: Autoria própria

Para definir a fórmula de compasso, clique no botão ilustrado na Figura 39. Será exibida um menu para escolher entre as possíveis fórmulas: Binário Simples, Ternário Simples e Quaternário Simples.

Figura 39 - Manual - Botão Selecionar Fórmula

Fonte: Autoria própria

O tempo, instrumento e volume podem ser acessados pelos botões ilustrados em Figura 40, Figura 41 e Figura 42 respectivamente. Estas informações não são armazenadas junto com o exercício.

Figura 40 - Manual - Botão Alterar Tempo

Fonte: Autoria própria

Figura 41 - Manual - Botão Selecionar Instrumento

Fonte: Autoria própria

Figura 42 - Manual - Botão Alterar Volume

Fonte: Autoria própria

Para ouvir o exercício atual, clique no botão ilustrado na Figura 43.

Figura 43 - Manual - Botão Ouvir

Fonte: Autoria própria

Para salvar o exercício novo, ou alterações de um exercício existente, clique no botão ilustrado na Figura 34.

Para descartar as alterações, clique no botão ilustrado na Figura 35.

EDITAR DADOS

Na tela de edição de exercícios, ao clicar no botão ilustrado na Figura 38 ou pressionar as teclas Ctrl + Tab, alternará entre os modos de edição.

Quando o campo "Descrição" ou "Dica" estiver AMARELO, está no modo de edição de dados, permitindo que se altere a descrição e a dica do exercício.

A descrição não é exibida ao aluno, sendo uma informação apenas para o professor saber de que exercício se trata.

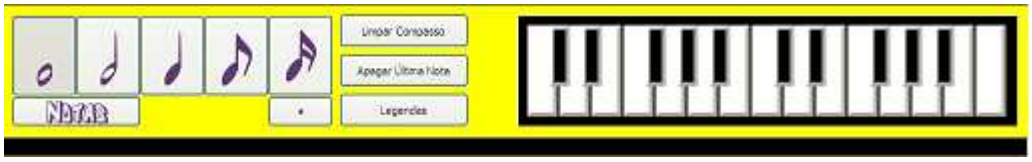
A dica é exibida ao aluno, e mesmo sendo armazenada no banco de dados, para cada arquivo que vier a ser gerado, poderá ser usado uma dica diferente.

EDITAR FRASE MUSICAL

Na tela de edição de exercícios, ao clicar no botão ilustrado na Figura 38 ou pressionar as teclas Ctrl + Tab, alternará entre os modos de edição.

Quando o painel de ferramentas estiver na cor AMARELA, conforme ilustra a Figura 44, indica que está no modo de edição da frase musical:

Figura 44 - Manual - Painel de Ferramentas



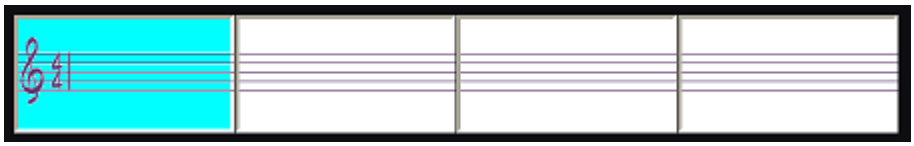
Fonte: Autoria própria

Para inserir uma nota musical, é necessário definir 3 pontos básicos:

- Compasso
- Figura Musical (duração)
- Nota Musical (altura)

Quando definimos o compasso, estamos selecionando em qual compasso será inserida a nota. Para isto basta clicar sobre o compasso desejado ou navegar utilizando as setas direcionais. O compasso selecionado ficará realçado na cor azul, conforme ilustra a Figura 45.

Figura 45 - Manual - Partitura



Fonte: Autoria própria

A Figura Musical é inicialmente definida para Semibreve. Note que o botão com a figura de semibreve aparece pressionada. Quando a fórmula de compasso é Ternária ou Binária, este botão é ocultado e por padrão a figura de Mínima permanece pressionada. Para selecionar a figura musical, basta clicar no botão correspondente desejado, ou utilizar a tecla de atalho 1, 2, 3, 4 ou 5 para Semibreve, Mínima, Semínima, Colcheia e Semicolcheia respectivamente. Para inserir notas pontuadas é necessário deixar o botão ilustrado na Figura 46.

Figura 46 - Manual - Botão Pontuada

Fonte: Autoria própria

Para alternar entre nota musical ou notas de pausa, clique no botão ilustrado na ou utilize a tecla de atalho Tab.

Figura 47 - Manual - Botão Alternar Positivo/Negativo

Fonte: Autoria própria

A nota musical deve ser sempre a última a ser definida, pois ao selecionarmos a nota, esta já é inserida no compasso com a figura determinada. Para selecionar a nota, clique na

nota desejada no teclado ilustrado na Figura 48, ou utilize teclas de atalho: A para dó, S para ré, D para mí, e assim por diante.

Figura 48 - Manual - Teclado

Fonte: Autoria própria

Para visualizar as teclas de atalho correspondentes, clique em "Legendas" ou pressione a barra de espaço.

GERAR ARQUIVO

Na tela de edição de exercícios, se houver exercício selecionado, ao clicar em “Gerar Arquivo” será exibida a tela de exportação de arquivo, ilustrada na Figura 49.

Figura 49 - Manual - Tela de Exportação de Arquivo

Fonte: Autoria própria

Selecione o aluno para qual o exercício está sendo gerado e preencha os campos tempo mínimo e tempo máximo para estipular o quanto o aluno poderá alterar o tempo. Caso eles sejam iguais, o tempo será fixado e o aluno não poderá alterar.

Determine quantas vezes o aluno poderá ouvir a frase musical para responder o exercício. Caso o valor seja 0 (ZERO), não haverá limite, e o aluno poderá ouvir quantas vezes julgar necessário.

O texto escrito no campo "DICA" será exibido ao aluno quando este for executar o exercício.

Clique no botão ilustrado na Figura 34 e selecione o diretório em que deseja salvar o arquivo.

ALTERAR SENHA

Na tela de menu principal, ao clicar em Senha (ou tecla de atalho "S"), será exibida a tela de alteração de senha, ilustrada na Figura 50.

Figura 50 - Manual - Tela de Alteração de Senha

Fonte: Autoria própria

Nesta tela, o professor pode alterar a sua senha.

Todo professor ao ser cadastrado, a senha gerada é o próprio nome de usuário, porém, em letras minúsculas.

É aconselhável que altere a senha na primeira utilização, para evitar uso indevido de dados de seus alunos.

Para alterar a senha, basta preencher os campos conforme indicado e clicar no botão ilustrado na Figura 34.