UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

RAFAEL PELLE

DESENVOLVIMENTO DE UM ANALISADOR DE CÓDIGO PARA SNAP! VOLTADO AO ENSINO DE COMPUTAÇÃO NA EDUCAÇÃO BÁSICA

FLORIANÓPOLIS
2018

RAFAEL PELLE

DESENVOLVIMENTO DE UM ANALISADOR DE CÓDIGO PARA SNAP! VOLTADO AO ENSINO DE COMPUTAÇÃO NA EDUCAÇÃO BÁSICA

Trabalho de Conclusão do Curso de Graduação em Ciências da Computação, do Departamento de Informática e Estatística, do Centro Tecnológico da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Orientadora: Prof.ª Dr.ª rer. nat. Christiane Gresse von Wangenheim, PMP

FLORIANÓPOLIS
2018

RAFAEL PELLE

DESENVOLVIMENTO DE UM ANALISADOR DE CÓDIGO PARA SNAP! VOLTADO AO ENSINO DE COMPUTAÇÃO NA EDUCAÇÃO BÁSICA

Trabalho de conclusão de curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharelado em Ciências da Computação.

Orientadora:

_____

Prof.ª Dr.ª rer. nat. Christiane Gresse von Wangenheim, PMP
Universidade Federal de Santa Catarina

Banca Examinadora:

_____

Prof. Dr. Jean Carlo Rossa Hauck
Universidade Federal de Santa Catarina

_____

Nathalia da Cruz Alves
Universidade Federal de Santa Catarina

FLORIANÓPOLIS
2018

RESUMO

O desenvolvimento tecnológico dos últimos anos tornou a computação indispensável no cotidiano da população, tanto na carreira profissional quanto na vida pessoal. Um dos grandes desafios da área é a formação de profissionais qualificados para atender as necessidades, e em quantidade para suprir a demanda, do mercado de trabalho. Por sua atual importância deve-se começar a ensinar computação já na Educação Básica. O ensino de computação na Educação Básica, utilizando ambientes de desenvolvimento como Snap! auxilia os alunos a aprender competências básicas de programação e pensamento computacional. Uma das principais etapas dentro de uma unidade instrucional de ensino de computação é a avaliação da aprendizagem do aluno, por exemplo, analisando o código desenvolvido no Snap!, uma linguagem de programação visual baseada em blocos. Tipicamente esta tarefa requer um esforço considerável, além da necessidade da capacitação do professor do Ensino Básico. Nesse contexto, o objetivo deste trabalho é o desenvolvimento de uma ferramenta web para automatizar o processo de avaliação da aprendizagem por meio da análise do código de projetos desenvolvidos com a linguagem de programação Snap! a ser utilizada em unidades instrucionais para ensinar a computação no Ensino Básico. A partir do levantamento do estado da arte é concebido um modelo conceitual de análise e avaliação automatizada. A partir desse modelo conceitual são identificados os requisitos a uma ferramenta web, que é modelada, implementada e testada. Os resultados do trabalho são avaliados por meio da aplicação da ferramenta no contexto educacional. Com isto espera-se uma facilitação na parte da avaliação da aprendizagem dos alunos, aumentando as chances de o ensino de computação ser adotado mais amplamente nas escolas.

Palavras-chave: Computação, Snap!, Análise de código, Unidade Instrucional, Ensino Básico.

LISTA DE FIGURAS

Figura 1 - Comparação entre o número de concluintes de cursos de português, matemática e informática (INEP, 2010 a 2014) 11
Figura 2 - Definição do termo competências (DEPARTMENT OF DEFENSE HANDBOOK, 1988) 15
Figura 3 - Estrutura sistemática do ensino 16
Figura 4 - Modelo ADDIE (BRANCH, 2009) 17
Figura 5 - Procedimentos comuns da fase de avaliação 19
Figura 6 - Os níveis do currículo CSTA/ACM K-12 (CSTA, 2011) 20
Figura 7 - Exemplo de arquitetura de uma ferramenta de análise estática (JONES, 2013) 24
Figura 8 - Exemplo de um roteiro Snap! 29
Figura 9 - Interface gráfica do Snap! (SNAP!, 2013) 31
Figura 10 - Opções do menu "File" do Snap! 32
Figura 11 - Definindo o nome do projeto Snap! 32
Figura 12 - Roteiro no arquivo XML 33
Figura 13 - Áudio no arquivo XML 33
Figura 14 - Imagem no arquivo XML 34
Figura 15 - Analisando um projeto no Scrape 38
Figura 16 - Analisando um projeto com Dr.Scratch 39
Figura 17 - Analisando um projeto no Ninja Code Village 40
Figura 18 - Analisando um projeto na ferramenta λ 41
Figura 19 - Processo de análise e avaliação 46
Figura 20 - Processo de análise de um projeto Snap! 47
Figura 21 - Ninjas do CodeMaster 50
Figura 22 – Processo de avaliação de um projeto Snap! 51
Figura 23 - Diagrama de Casos de Uso 55
Figura 24 – Protótipo da página inicial 58
Figura 25 - Protótipo da página “aluno” 59
Figura 26 – Protótipo da tela “resultado para aluno” 60
Figura 27 - Protótipo da tela “login para professores” 60
Figura 28 - Protótipo da tela “página principal para professores” 61
Figura 29 – Protótipo da tela “Cadastrar professor” 61
Figura 30 - Protótipo da tela "resultado para professor" 62
Figura 31 - Diagrama de componentes 63
Figura 32 - Diagrama de Classes do serviço REST 64
Figura 33 - Diagrama de Classes do serviço REST para Snap! 65
Figura 34 - Diagrama de Sequência do método gradeProject 67
Figura 35 - Diagrama de Classes simplificado do módulo de apresentação 69
Figura 36 - Diagrama de Classes do caso de uso "USC02 - Cadastrar professor" 70
Figura 37 - Diagrama de Sequência modelo do upload de múltiplos projetos 71
Figura 38 - Diagrama Entidade Relacionamento do Banco de Dados 72
Figura 39 - Usuários que participaram da avaliação 83
Figura 40 - Avaliação realizada pela ferramenta CodeMaster 89

LISTA DE TABELAS

Tabela 1 - Objetivos de aprendizagem para o nível 2 (CSTA, 2011)....22
Tabela 2 - Rubrica de avaliação do Dr.Scratch (MORENO & ROBLES, 2015) 26
Tabela 3 - Principais elementos da linguagem Snap!....28
Tabela 4 - Categorias de blocos da linguagem Snap! 30
Tabela 5 - Regiões destacadas na interface gráfica....32
Tabela 6 - Critérios de inclusão e exclusão do estudo de mapeamento....35
Tabela 7 - Termos de busca....35
Tabela 8 - Strings de busca....36
Tabela 9 - Resumo das buscas 36
Tabela 10 - Ferramentas identificadas por meio do estudo de mapeamento 37
Tabela 11 - Comparativo entre as ferramentas identificadas....42
Tabela 12 - Comparativo entre as ferramentas identificadas....42
Tabela 13 - Dimensões-chave de avaliação....48
Tabela 14 - Rubrica de avaliação da ferramenta CodeMaster (Snap!)....48
Tabela 15 - Tipos de operadores....49
Tabela 16 - Mapeamento entre nota e nível de expertise....51
Tabela 17 - Requisitos funcionais....52
Tabela 18 - Requisitos não funcionais....54
Tabela 19 – Atores do sistema 55
Tabela 20 - Casos de teste....76
Tabela 21 - Especificações da máquina 79
Tabela 22 - Resultados obtidos nos testes de desempenho 79
Tabela 23 - Fatores de qualidade a serem avaliados 80
Tabela 24 - Projetos selecionados para avaliação 84
Tabela 25 - Avaliação da utilidade da ferramenta CodeMaster 87
Tabela 26 - Avaliação da adequação funcional da ferramenta CodeMaster....88
Tabela 27 - Avaliação do desempenho da ferramenta CodeMaster....88
Tabela 28 - Resultado da aplicação do questionário SUS....88
Tabela 29 - Comparação entre os resultados das avaliações CodeMaster e avaliações manuais 90

LISTA DE ABREVIATURAS

ACM – Association for Computing Machinery

ADDIE – Analyze, Design, Develop, Implement, Evaluate

BYOB – Build Your Own Blocks

CnE – Computação na Escola

CSTA – Computer Science Teachers Association

GQS – Grupo de Qualidade de Software

IEEE – Instituto de Engenheiros Eletricistas e Eletrônicos

INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira

JSON – JavaScript Object Notation

JSP – JavaServer Pages

MEC – Ministério da Educação

MIT – Massachusetts Institute of Technology

NCV – Ninja Code Village

PCN – Parâmetros Curriculares Nacionais

REST – REpresentational State Transfer

SUS – System Usability Scale

TAM – Technology Acceptance Model

TI – Tecnologia da Informação

XML – eXtensible Markup Language

Sumário

1. INTRODUÇÃO 10
1.1 CONTEXTUALIZAÇÃO 10
1.2 OBJETIVOS 12
1.3 METODOLOGIA DE PESQUISA 13
1.4 ESTRUTURA DO DOCUMENTO 14

2. FUNDAMENTAÇÃO TEÓRICA 15
2.1 APRENDIZAGEM E ENSINO 15
2.2 ENSINO DE COMPUTAÇÃO NO ENSINO BÁSICO 19
2.3 ANÁLISE DE CÓDIGO 24
2.3.1 ANÁLISE ESTÁTICA DE CÓDIGO NO CONTEXTO EDUCACIONAL 25
2.4 AMBIENTE E LINGUAGEM DE PROGRAMAÇÃO SNAP! 28

3. ESTADO DA ARTE 34
3.1 DEFINIÇÃO DO PROTOCOLO DE ESTUDO 34
3.2 EXECUÇÃO DA BUSCA 36
3.3 EXTRAÇÃO E ANÁLISE DOS DADOS 37
3.3.1 SCRAPE 37
3.3.2 HAIRBALL 38
3.3.3 DR.SCRATCH 39
3.3.4 NINJA CODE VILLAGE 40
3.4 DISCUSSÃO 42
3.4.1 AMEAÇAS À VALIDADE 44

4. DESENVOLVIMENTO DA CODEMASTER – SNAP! 45
4.1 MODELO CONCEITUAL 45
4.2 PROCESSO DE ANÁLISE E AVALIAÇÃO 46
4.3 ANÁLISE DOS REQUISITOS 52
4.4 CASOS DE USO 55
4.5 PROTOTIPAÇÃO DE TELAS 58
4.6 MODELAGEM E IMPLEMENTAÇÃO 62
4.6.1 ARQUITETURA DO MÓDULO DE AVALIAÇÃO 64
4.6.2 ARQUITETURA DO MÓDULO DE APRESENTAÇÃO 68
4.7 EXPANDINDO O CODEMASTER 73
4.7.1 ADICIONANDO NOVO CRITÉRIO DE AVALIAÇÃO 73
4.7.2 ADICIONANDO NOVO ANALISADOR DE CÓDIGO 74
4.8 TESTES 75
4.8.1 CASOS DE TESTE 76
4.8.2 TESTES DE DESEMPENHO 78

5. AVALIAÇÃO 80
5.1 DEFINIÇÃO DA AVALIAÇÃO 80
5.1.1 DEFINIÇÃO DA AVALIAÇÃO COM USUÁRIOS 82
5.1.2 DEFINIÇÃO DA AVALIAÇÃO DA CORRETUDE 82
5.2 EXECUÇÃO DA AVALIAÇÃO 83
5.2.1 EXECUÇÃO DA AVALIAÇÃO COM USUÁRIOS 83
5.2.2 EXECUÇÃO DA AVALIAÇÃO DA CORRETUDE 84
5.3 ANÁLISE DOS DADOS 86
5.3.1 ANÁLISE DOS DADOS DAS AVALIAÇÕES COM USUÁRIOS 86
5.3.2 ANÁLISE DOS DADOS DA AVALIAÇÃO DA CORRETUDE 89
5.3.3 PONTOS FORTES 91
5.3.4 SUGESTÕES DE MELHORIA 91
5.3.5 AMEAÇAS À VALIDADE 91
6. CONCLUSÃO 93
REFERÊNCIAS 95
ANEXO A – OS BLOCOS DISPONÍVEIS NO SNAP! (VERSÃO 4.0) 99
ANEXO B – AVALIAÇÃO COM OS USUÁRIOS 102

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Ao longo das últimas décadas, computadores transformaram profundamente, de várias formas, o mundo e o modo de trabalho. Como resultado, as ciências da computação e as tecnologias que elas habilitam estão no centro da economia e no modo como se vive (CSTA, 2011). Computadores têm um impacto enorme no modo como vivemos, pensamos e agimos. De fato, acredita-se que a verdadeira revolução do computador não vai acontecer até que todos consigam entender a tecnologia suficientemente bem para usá-la de modo verdadeiramente inovador (CSTA, 2011). Para ser um cidadão bem instruído em um mundo intensamente informatizado e estar preparado para careiras no século XXI, os estudantes têm que ter um entendimento claro dos princípios e práticas da computação (CSTA, 2011).

Apesar da importância da computação e ótimas perspectivas de carreira, os cursos de graduação na área de Tecnologia da Informação (TI) ainda têm baixa concorrência por vagas e altas taxas de desistência. Em 2010, a evasão nos cursos da área de TI foi de 87% em todo o Brasil (BRASSCOM, 2012). É inevitável que isso se reflita no mercado de trabalho. O déficit de 408 mil profissionais de TI estimado para 2022 pode significar uma perda de receita de R$ 167 bilhões entre 2010 e 2020 para o setor (SOFTEX, 2014). Assim, existe uma necessidade urgente de melhorar o nível de compreensão pública de da computação como um campo acadêmico e profissional (CSTA, 2011).

Um dos objetivos do ensino dos conceitos de computação no Ensino Básico é popularizá-la e motivar os alunos a seguirem carreira na área, melhorando assim a situação atual do mercado de trabalho. O ensino destes princípios e práticas tem benefícios que vão além de uma formação profissional. Estudantes de computação aprendem raciocínio lógico, pensamento algorítmico, concepção e resolução de problemas estruturados, todos esses conceitos e habilidades são valiosos muito além da sala de aula de ciências da computação (CSTA, 2011). Estudar computação pode preparar um estudante para uma carreira em muitas áreas, tanto dentro quanto fora da computação (CSTA, 2011).

Uma alternativa para ensinar computação na Educação Básica é o uso de linguagens de programação visual, baseadas em blocos como a linguagem Snap! (SNAP!; BERKELEY, 2013). Com Snap! é possível criar jogos e animações, usando laços de repetição, execução condicional, eventos, operadores matemáticos e *booleanos*, listas entre outros recursos. Snap! é uma linguagem de programação amplamente convidativa para crianças e adultos e também uma plataforma para estudar computação (SNAP!; BERKELEY, 2013).

Embora existam ambientes de programação como Snap!, ainda é raro ensinar computação no Ensino Básico. Uma das razões é a falta de professores de computação no Ensino Básico, já que o número de concluintes em cursos de licenciatura em computação é muito menor do que em cursos de formação de professores de matérias específicas (INEP, 2010 a 2014). A Figura 1 compara o número de concluintes de cursos de formação de professor de português e matemática com número de concluintes de cursos de licenciatura em computação do ensino superior no período de 2010 a 2014.

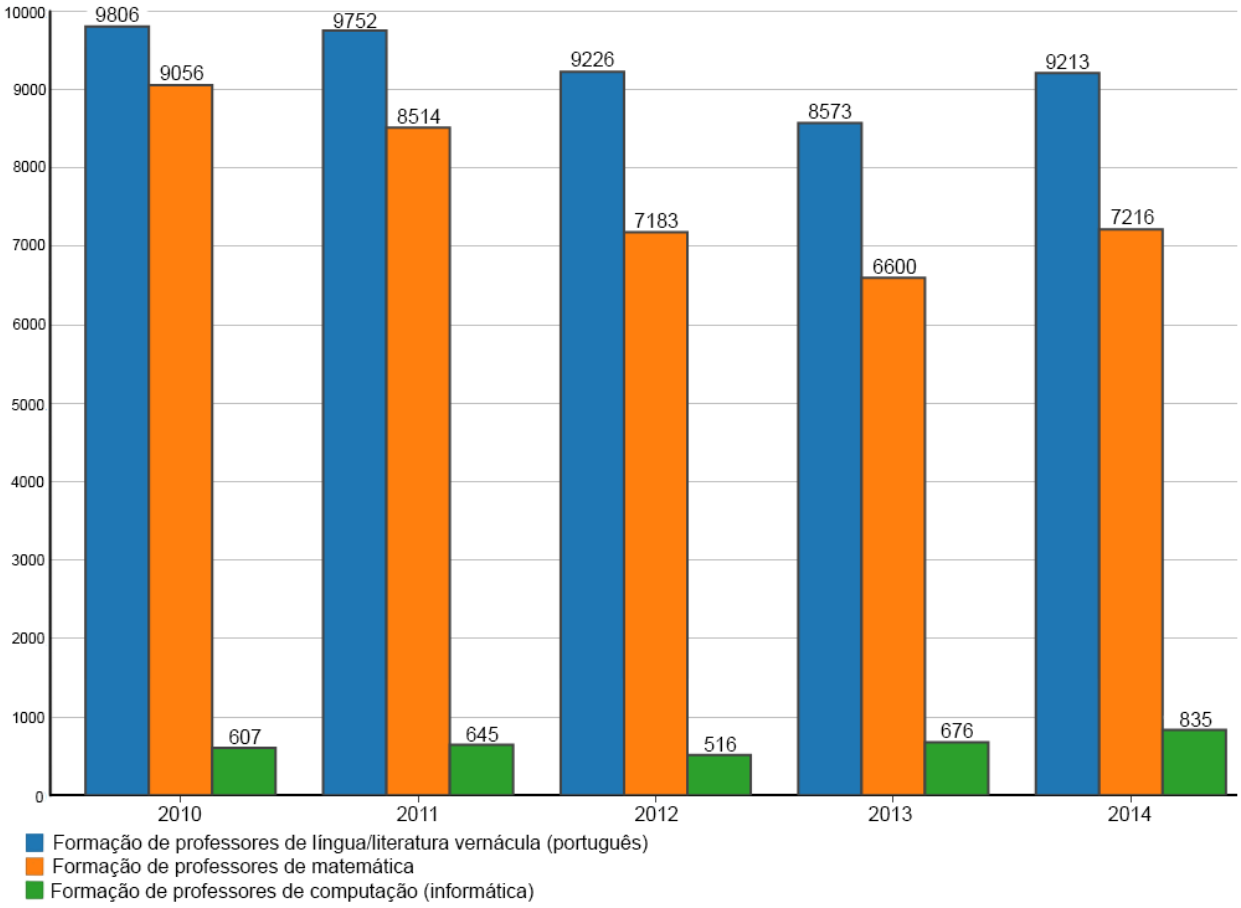


Figura 1 - Comparação entre o número de concluintes de cursos de português, matemática e informática (INEP, 2010 a 2014)

Essa escassez de professores capacitados na área no Ensino Básico, é uma das razões que dificulta inserir o ensino de computação nas escolas. Desta maneira, o ensino de computação em escolas depende da aplicação de unidades instrucionais interdisciplinares pré-definidas, por professores do Ensino Básico de outras áreas, por exemplo, professores de história, mesmo sem necessariamente ter uma formação formal de computação. Outra questão que dificulta a inserção do ensino de computação é a falta de tempo desses professores para preparação e acompanhamento das aulas, além das suas outras responsabilidades. Também se observa dificuldade especificamente na avaliação da aprendizagem dos alunos, parte essencial do ensino. Sem ter recebido um

treinamento formal, corrigir e avaliar os trabalhos desenvolvidos pelos alunos se torna uma tarefa árdua, além de despender muito tempo se for realizada manualmente.

A avaliação de um trabalho desenvolvido por um aluno pode ser realizada por meio da análise do código produzido. Análise de código é o processo de extrair informação sobre um programa através do código fonte ou artefatos derivados do código fonte, por exemplo, *Java byte code,* utilizando ferramentas automatizadas (BINKLEY, 2007).

Nesse contexto, este trabalho visa facilitar a avaliação da aprendizagem dos alunos em relação a projetos de programação, desenvolvendo uma ferramenta web para automatizar o processo de avaliação da aprendizagem por meio da análise do código de projetos desenvolvidos com a linguagem de programação Snap!, a ser utilizada em unidades instrucionais no ensino de computação no Ensino Básico. Espera-se que a ferramenta aumente as chances de o ensino de computação ser adotado mais amplamente nas escolas.

1.2 OBJETIVOS

Objetivo geral

O objetivo deste trabalho é o desenvolvimento de uma ferramenta web para analisar e avaliar automaticamente código desenvolvido com a linguagem de programação para iniciantes Snap!, a ser utilizada em unidades instrucionais no ensino de computação no Ensino Básico.

Objetivos específicos

O1. Analisar a fundamentação teórica sobre aprendizagem, ensino de computação no Ensino Básico, a ferramenta de desenvolvimento Snap! e análise de código.

O2. Analisar o estado da arte sobre ferramentas para análise de código, utilizadas no ensino de computação no Ensino Básico.

O3. Desenvolver uma ferramenta web para análise e avaliação automatizada de projetos Snap!.

O4. Aplicar e avaliar a ferramenta criada em relação a corretude e aplicabilidade.

Premissas e restrições

O trabalho é realizado de acordo com o regulamento vigente do Departamento de Informática e Estatística (INE – UFSC) em relação aos Trabalhos de Conclusão de Curso.

A ferramenta desenvolvida tem como foco a análise e avaliação de código, somente de projetos desenvolvidos com a linguagem de programação Snap! (versão 4.0) no ensino de computação no Ensino Básico.

O presente trabalho é desenvolvido em paralelo com o desenvolvimento do Trabalho de Conclusão de Curso do aluno Matheus Faustino Demetrio (DEMETRIO, 2017), que por sua vez tem como objetivo, também, criar uma ferramenta web para a análise de código, porém, para outra linguagem de programação: App Inventor. Deste modo, o esforço de desenvolvimento da parte estrutural de servidor e interface gráfica do sistema é dividida, permitindo criar uma estrutura única mais robusta, funcional e expansível. O desenvolvimento do projeto da interface gráfica foi auxiliado pelos alunos Heliziane Barbosa e Luiz Felipe Azevedo, bolsistas da iniciativa Computação na Escola do GQS/InCod/INE/UFSC.

1.3 METODOLOGIA DE PESQUISA

A metodologia de pesquisa utilizada neste trabalho é dividida em quatro etapas.

Etapa 1 – Fundamentação teórica

A fundamentação teórica será apresentada após estudo, análise e síntese dos conceitos principais e a teoria referente aos temas a serem abordados neste trabalho. Nesta etapa são realizadas as seguintes atividades:

A1.1 – Análise teórica sobre ensino e aprendizagem, especificamente o ensino de computação no Ensino Básico.

A1.2 – Análise teórica sobre o ambiente de desenvolvimento Snap!.

A1.3 – Análise teórica sobre análise de código.

Etapa 2 – Estado da arte

Nesta etapa é realizado um estudo de mapeamento (PETERSEN, et al., 2008) para identificar e analisar analisadores de código de linguagens de programação visual (baseadas em blocos) atualmente sendo utilizados. Esta etapa é dividida nas seguintes atividades:

A2.1 – Definição do protocolo de estudo.

A2.2 – Execução da busca por analisadores de código.

A2.3 – Extração e análise de informações relevantes.

Etapa 3 – Desenvolvimento

Nesta etapa é desenvolvida uma ferramenta *web* para análise e avaliação automatizada de projetos Snap! seguindo um processo de engenharia de software iterativo incremental (WAZLAWICK, 2013). Esta etapa é dividida nas seguintes atividades:

A3.1 – Análise de requisitos.

A3.2 – Design de interface

A3.3 – Modelagem da arquitetura do sistema.

A3.4 – Modelagem detalhada e implementação.

A3.5 – Testes do sistema.

Etapa 4 – Aplicação e avaliação

Nesta etapa a ferramenta desenvolvida é aplicada, avaliada sob o aspecto da corretude, comparando os resultados gerados com uma análise manual do código, e avaliada sob o aspecto da aplicabilidade na prática educacional. Esta etapa é dividida nas seguintes atividades:

A4.1 – Avaliação da corretude.

A4.1.1 – Definição da avaliação da corretude.

A4.1.2 – Execução da avaliação da corretude.

A4.1.3 – Análise dos resultados.

A4.2 – Avaliação da aplicabilidade

A4.2.1 – Definição da avaliação da aplicabilidade.

A4.2.3 – Execução da avaliação da aplicabilidade.

A4.1.4 – Análise dos resultados.

A4.3 – Discussão dos resultados.

1.4 ESTRUTURA DO DOCUMENTO

Na seção 2 deste trabalho são abordados os conceitos da base do ensino e aprendizagem em geral e focados no ensino de computação. Além dos conceitos de análise e avaliação no contexto do Ensino Básico.

Na seção 3 é levantado o estado da arte sobre analisadores e avaliadores automatizados de código voltados a linguagens de programação visuais para o ensino de computação no Ensino Básico.

Na seção 4 é apresentado o desenvolvimento de uma ferramenta de análise e

avaliação de código voltado a projetos desenvolvidos com Snap!. Nesta seção é apresentado os passos da implementação, bem como as técnicas e tecnologias utilizadas para a criação da ferramenta.

Na seção 5 são abordadas as avaliações de corretude da ferramenta bem como as avaliações de utilidade, funcionalidade, desempenho e usabilidade medidas por meio de testes com usuários da área do ensino de computação.

Na seção 6 são apresentadas as conclusões. São verificados se os objetivos propostos foram atendidos, qual a contribuição da ferramenta para a sociedade e propostas de futuros trabalhos.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados conceitos sobre a teoria de ensino e aprendizagem, design instrucional e avaliação da aprendizagem. São abordados tópicos sobre ensino de computação no Ensino Básico. É apresentada uma visão geral sobre a linguagem de programação Snap!. A teoria sobre análise de código também será abordada.

2.1 APRENDIZAGEM E ENSINO

Formalmente, a aprendizagem é definida como “Processo por meio do qual uma nova competência é incorporada à estrutura cognitiva do indivíduo” (MICHAELIS, 2016). Essas competências podem ser entendidas como a união entre conhecimentos, habilidades e atitudes (Figura 2).

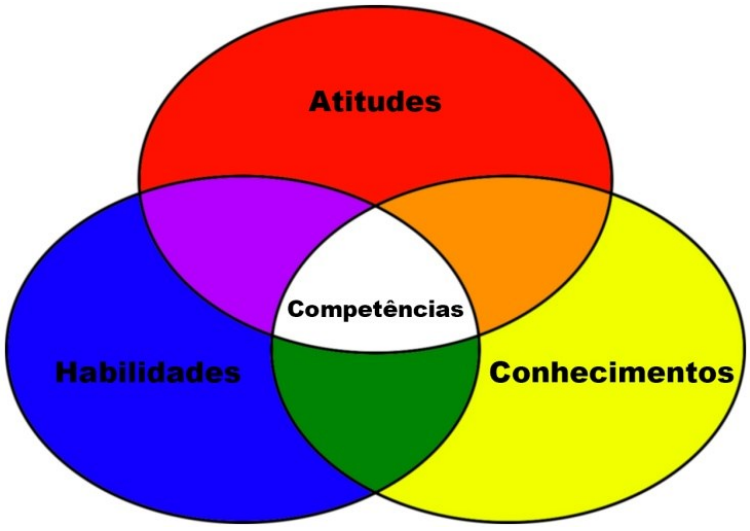


Figura 2 - Definição do termo competências (DEPARTMENT OF DEFENSE HANDBOOK, 1988)

Novas competências podem ser adquiridas por meio de estudo, experiência, raciocínio e observação e pode ser transmitida de forma sistemática. Neste contexto, o ensino sistematiza a aprendizagem de novas competências. (MICHAELIS, 2016).

O ensino pode ser, então, definido como uma atividade sistemática de interação entre seres, interação esta que se configura numa ação exercida sobre o sujeito ou grupo de sujeitos visando provocar neles mudanças tão eficazes que os tornem elementos ativos desta própria ação exercida (LIBÂNEO, 2009). Presume-se, a interligação de três elementos (Figura 3): um agente (professor, alguém, um grupo, etc.), uma competência transmitida (conhecimentos, atitudes e habilidades) e um educando (aluno, grupo de alunos, uma geração) (LIBÂNEO, 2009).

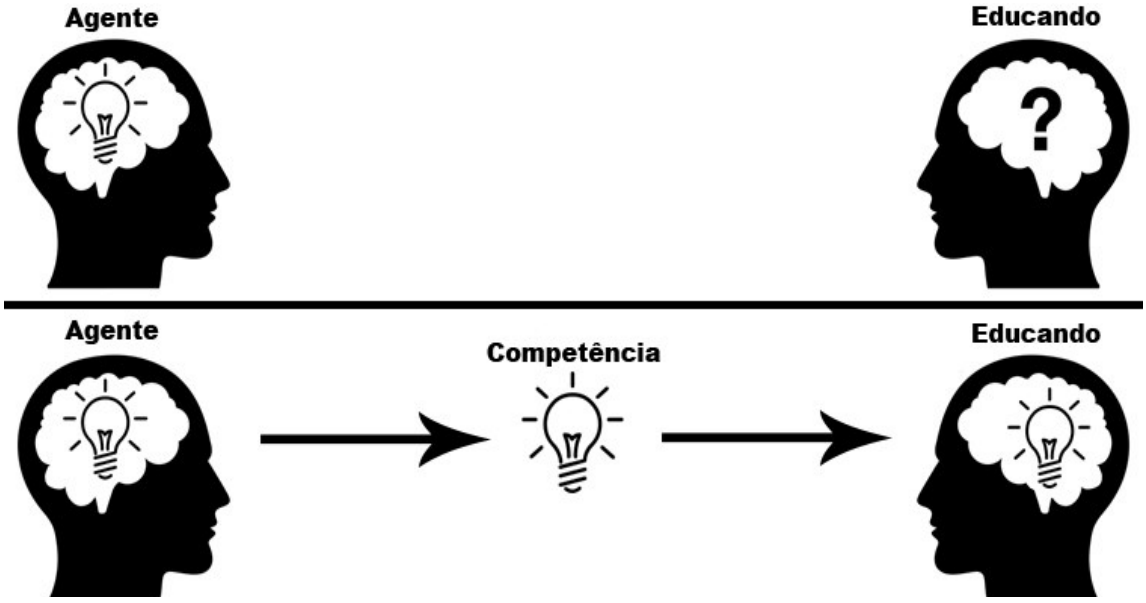


Figura 3 - Estrutura sistemática do ensino

Existem diversas formas de ensino, entre elas o ensino formal e o ensino informal. O ensino formal é aquele desenvolvido nas escolas, com conteúdo previamente demarcado, o informal é aquele que os indivíduos aprendem durante seu processo de socialização (na família, bairro, clube, amigos, etc.), carregado de valores e culturas próprias (GOHN, 2006).

O foco do presente trabalho é o ensino formal, que é intencional e planejado, acontece em instituições de ensino regulamentadas, por um agente (professor) que aplica unidades instrucionais com o objetivo de transmitir competências para seus educandos. Essas unidades instrucionais podem ser uma aula, uma oficina, um curso, ou até mesmo um jogo educacional. Uma unidade instrucional deve possuir uma declaração descrevendo as competências desejáveis como resultado do processo de ensino (objetivos de

aprendizagem) bem definidas. Para que o ensino atinja os objetivos de aprendizagem as unidades instrucionais precisam ser desenvolvidas sistematicamente adotando abordagens de design instrucional.

Design Instrucional

Design Instrucional é se refere à engenharia pedagógica (BASQUE, 2010) e trata-se de uma metodologia para fazer com que a transmissão de competências seja eficiente, efetiva e motivadora (MERRILL et al., 1996). O Design Instrucional corresponde à: ação intencional e sistemática de ensino, que envolve o planejamento, o desenvolvimento e a utilização de métodos, técnicas, atividades, materiais, eventos e produtos instrucionais em situações de ensino específicas, a fim de facilitar a aprendizagem humana a partir dos princípios de aprendizagem e instrução conhecidos (FILATRO, 2004). Design instrucional é um processo iterativo para planejar os objetivos de desempenho, selecionar estratégias instrucionais, escolher mídia, selecionar e/ou criar materiais e aplicar avaliação (BRANCH, 2009).

Um dos principais modelos de design instrucional é o modelo **ADDIE**[1] (BRANCH, 2009).

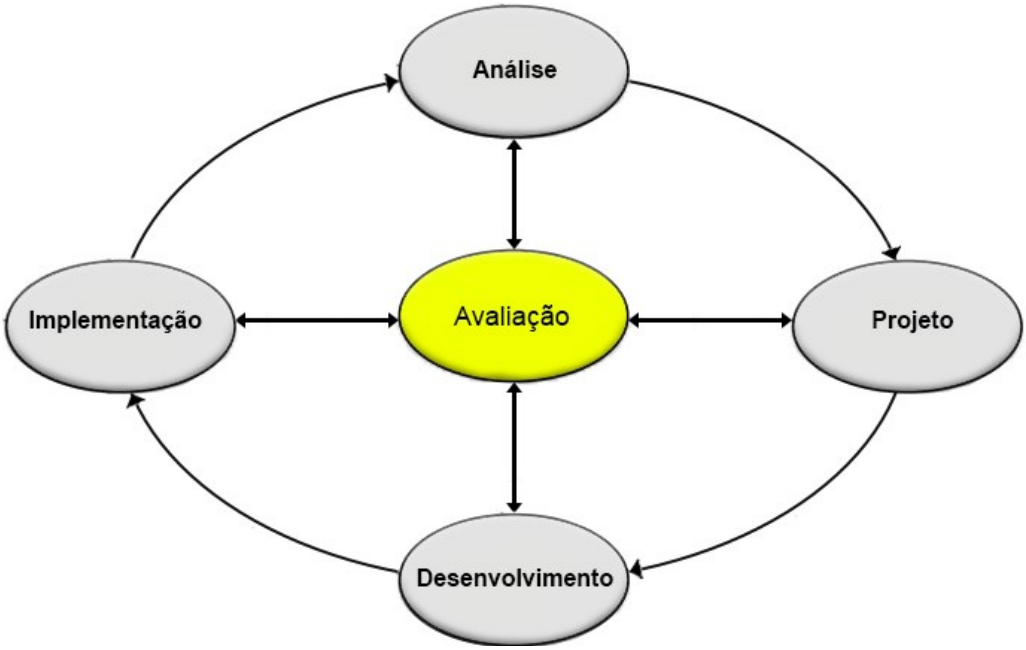


Figura 4 - Modelo ADDIE (BRANCH, 2009)

O modelo ADDIE é composto de 5 fases (BRANCH, 2009):

[1] O nome ADDIE é um acrônimo para as palavras: ***A**nalyze, **D**esign, **D**evelop, **I**mplement e **E**valuate* (Análise, Projeto, Desenvolvimento, Implementação e Avaliação) que representam cada uma das cinco fases do modelo.

- **Análise:** esta fase foca na análise de componentes que serão utilizados para orientar o projeto de desenvolvimento da unidade instrucional. Devem ser analisadas as características do público-alvo, estudado o nível de habilidade que cada aprendiz apresenta ter, que conhecimentos ele possui e que conhecimentos ele deve ter após a aplicação da unidade instrucional. Também é analisado o contexto em que se insere a unidade instrucional e quais são as limitações de recursos humanos, técnicos, financeiro e de tempo.
- **Projeto:** nesta fase é feita a especificação detalhada do objetivo geral da unidade instrucional e de todos os objetivos de aprendizagem. Também são escolhidos quais recursos serão utilizados para o desenvolvimento da unidade instrucional, selecionados quais conteúdos deverão ser abordados e que métodos instrucionais serão utilizados. São definidos quais os tipos de atividades que serão realizadas (colaborativas, interativas ou individuais) e quanto tempo será atribuído para cada uma delas. Também é feito o sequenciamento que definirá a ordem em que serão apresentados os conteúdos, a escolha de quais ferramentas serão utilizadas na medição de desempenho, os tipos de testes que serão feitos e a seleção da mídia de aprendizagem.
- **Desenvolvimento:** esta fase destina-se à criação dos materiais didáticos a serem utilizados durante a unidade. Envolve o desenvolvimento dos materiais projetados na fase anterior, podendo ser *slides*, folhas de exercícios, rubricas, guias, *worksheet*, entre outros. São utilizadas várias ferramentas (papel, caneta, processadores de texto, editor de gráfico, software de programação, etc.).
- **Implementação:** é a fase em que a unidade instrucional desenvolvida é aplicada na prática para direcionar os alunos à aprendizagem. Nesta fase são testados todos os materiais para determinar se estes são funcionais e apropriados para a audiência analisada na primeira fase.
- **Avaliação:** esta fase serve para avaliar as diferentes variáveis como qualidade, eficiência e eficácia da unidade instrucional com o intuito de melhorar e/ou tomar uma decisão sobre a sua adoção. É tipicamente avaliado se a unidade instrucional é efetiva, isto é, se melhora a aprendizagem, motiva os aprendizes, etc. Nesta fase determina-se de que modo será feita a coleta de dados e as medições para a avaliação da unidade instrucional.

Como o foco do presente trabalho é a avaliação da aprendizagem do aluno, é detalhada mais especificamente a avaliação da aprendizagem.

Avaliação da aprendizagem

A avaliação da aprendizagem é uma tarefa fundamental no processo de ensino. Ela deve acompanhar todas as etapas do processo de ensino e aprendizagem. É por meio da avaliação da aprendizagem que é possível determinar se os alunos atingiram os objetivos de aprendizagem. A avaliação no design instrucional tem foco na medição da capacidade do aluno de realizar a sua nova competência (BRANCH, 2009).

A avaliação pode ser feita de forma formativa ou somativa. A avaliação formativa deve ser interpretada como todas as atividades empreendidas por professores e/ou por alunos, que forneçam *feedback* para modificar as atividades de aprendizagem e ensino (BLACK; WILIAM, 1998). É utilizada durante todo o processo de ensino e aprendizagem, avalia o nível de entrada do aluno e avalia se o aluno está progredindo adequadamente para atingir os objetivos de aprendizagem. A avaliação somativa é feita ao final de uma unidade instrucional. É projetada para avaliar a aprendizagem de um aluno em relação aos objetivos de aprendizagem da unidade instrucional, com o propósito de dar nota, certificação ou avaliação do progresso (BLACK; WILIAM, 1996). A avaliação somativa é comumente aplicada usando provas, trabalhos práticos, exercícios ou apresentações.

Para ser feita de forma válida, a avaliação da aprendizagem precisa ser feita de forma sistemática seguindo um processo. A Figura 5 apresenta os procedimentos comuns da fase de avaliação segundo o modelo ADDIE (BRANCH, 2009).

Figura 5 - Procedimentos comuns da fase de avaliação

2.2 ENSINO DE COMPUTAÇÃO NO ENSINO BÁSICO

O Ministério da Educação (MEC) disponibiliza os Parâmetros Curriculares Nacionais, os quais foram criados para possibilitar uma base comum aos currículos nacionais do Ensino Básico. Para o Ensino Fundamental II (6º ao 9º ano), atualmente não é previsto, nos Parâmetros Curriculares Nacionais, nenhum conteúdo referente à área de Computação (PCN, 2015). Também não existe atualmente, no Brasil, um currículo nacional para o ensino de computação no Ensino Básico.

Porém, existem no mundo diversos currículos de referência para o ensino de computação no Ensino Básico, entre eles um dos mais reconhecidos é o CSTA/ACM K-12 (CSTA, 2011). Conforme esse framework de currículo de referência os objetivos de qualquer curso da computação, no Ensino Básico, devem ser (CSTA, 2011):

- Introduzir os conceitos fundamentais da computação para todos os estudantes, a começar no Ensino Fundamental;
- Apresentar a computação no nível do ensino secundário de uma maneira que possa ser tanto acessível quanto digna de crédito no currículo escolar;
- Incentivar as escolas a oferecerem cursos de ciência da computação adicionais de nível secundário, o que permitirá aos estudantes interessados a possibilidade de estudar aspectos da ciência da computação com mais profundidade e prepará-los para a entrada no mercado de trabalho ou faculdade;
- Aumentar a disponibilidade de aprendizado da ciência da computação para todos os alunos, especialmente aqueles que pertencem às minorias.

Para o ensino da computação no Ensino Básico, as diretrizes CSTA K-12 são baseadas em um modelo com três níveis, por faixas etárias. O nível 1 define os objetivos de aprendizagem para os estudantes do Ensino Básico até o quinto ano. O nível 2 fornece os objetivos de aprendizagem para estudantes do sexto ao nono ano do Ensino Fundamental. O nível 3 fornece os objetivos de aprendizagem para os alunos do ensino médio. O foco deste trabalho é o nível 2.

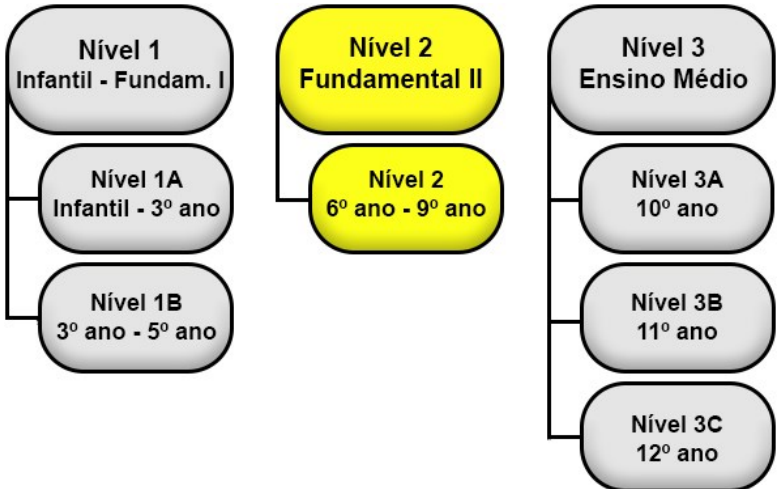


Figura 6 - Os níveis do currículo CSTA/ACM K-12 (CSTA, 2011)

Conforme este framework de currículo cinco áreas essenciais devem ser abordadas no ensino de computação no Ensino Básico (CSTA, 2011):

- **Programação:** trata-se de uma parte essencial da computação, é a competência de criar programas de software. Alunos devem aprender a projetar, desenvolver e publicar produtos (*websites*, aplicações móveis, animações e jogos) utilizando recursos tecnológicos. Eles devem compreender o que são algoritmos e qual a sua aplicação prática. Como parte da prática, também devem implementar software utilizando uma linguagem de programação.
- **Pensamento computacional:** é uma abordagem para a resolução de problemas de forma que pode ser implementada com um computador. Os alunos tornam-se não somente utilizadores de ferramentas, mas construtores de ferramentas. Os alunos fazem uso de um conjunto de conceitos, tais como abstração, recursão e iteração, para processar e analisar dados e criar artefatos reais e virtuais. O pensamento computacional é uma metodologia de resolução de problemas que pode ser automatizada, transferida e aplicada para diferentes áreas. A relevância do pensamento computacional se dá pela facilidade de aplicá-lo a qualquer outro tipo de raciocínio.
- **Colaboração:** a computação é uma disciplina intrinsecamente colaborativa. Portanto, é importante que sejam desenvolvidas habilidades de colaboração, tais como trabalho em equipe, crítica construtiva e comunicação eficaz.
- **Computadores e dispositivos de comunicação:** os alunos devem compreender os elementos do computador moderno e de dispositivos e redes de comunicação. Os alunos devem usar a terminologia apropriada e precisa quando se comunicam acerca de tecnologia.
- **Impactos éticos globais e na comunidade:** princípios de privacidade, segurança de rede, licenças de software e direitos autorais devem ser ensinados a fim de preparar os alunos a se tornarem cidadãos responsáveis no mundo moderno. Os alunos também devem ser capazes de avaliar a confiabilidade e a precisão das informações na Internet. É essencial que os alunos compreendam o impacto dos computadores na comunicação internacional e identificar até que ponto os problemas de acesso impactam nossas vidas.

Especificamente no nível 2, o foco do presente trabalho, os alunos devem começar a usar o pensamento computacional como uma ferramenta para a resolução de problemas. Eles começam a apreciar a ubiquidade da computação e as formas na qual a computação

facilita a comunicação e colaboração. Os alunos devem começar a experimentar o pensamento computacional como um meio de abordar questões relevantes, não apenas para eles, mas para o mundo em torno deles. As experiências de aprendizagem criadas devem ser relevantes para os alunos e promover a sua percepção de si mesmos como solucionadores de problemas proativos e capacitados. Elas devem ser projetadas com foco na aprendizagem e exploração ativa. Elas podem ser ensinadas em disciplinas explícitas de ciência da computação ou incorporados em outras áreas curriculares, tais como ciências sociais, línguas, matemática e ciência (CSTA, 2011). A Tabela 1 apresenta os objetivos para o nível 2 (6º ano ao 9º ano).

Tabela 1 - Objetivos de aprendizagem para o nível 2 (CSTA, 2011)

Pensamento Computacional
[O1] Usar os passos básicos de algoritmos para a resolução de problemas ao projetar soluções (p.ex., declaração e exploração do problema, examinação de exemplos, design, implementação de uma solução, testes, avaliação). **[O2]** Descrever o processo de paralelização na forma que se refere à resolução de problemas. **[O3]** Definir um algoritmo, como sendo uma sequência de instruções que podem ser processadas por um computador. **[O4]** Avaliar formas em que algoritmos diferentes podem ser utilizados para resolver o mesmo problema. **[O5]** Dramatizar algoritmos de busca e ordenação. **[O6]** Descrever e analisar uma sequência de instruções a ser seguida (p.ex., descrever o comportamento de um personagem em um videogame, dirigido por regras e algoritmos). **[O7]** Representar dados em maneiras diferentes, incluindo texto, sons, imagens e números. **[O8]** Usar representações visuais de estados de problema, estruturas, e dados (p.ex., gráficos, tabelas, diagramas de rede, fluxogramas). **[O9]** Interagir com modelos específicos de conteúdo e simulações (p.ex., ecossistemas, epidemias, dinâmica molecular) para apoiar a aprendizagem e pesquisa. **[O10]** Avaliar que tipos de problemas podem ser resolvidos usando modelagem e simulação. **[O11]** Analisar o grau em que um modelo de computador representa, com precisão, o mundo real. **[O12]** Fazer uso da abstração para decompor um problema em subproblemas. **[O13]** Compreender a noção de hierarquia e abstração em computação, incluindo linguagens de alto-nível, tradução (p. ex., interpretar o mesmo problema de diferentes modos), conjunto de instruções, e circuitos lógicos. **[O14]** Examinar conexões entre elementos da matemática e ciência da computação, incluindo números binários, lógica, conjuntos e funções. **[O15]** Fornecer exemplos de aplicações interdisciplinares do pensamento computacional.
Colaboração
[O16] Aplicar ferramentas e periféricos de produtividade/multimídia para colaboração em grupo e para apoiar a aprendizagem ao longo do currículo. **[O17]** Colaborativamente criar, desenvolver, publicar e apresentar produtos (p.ex., vídeos, podcasts, sites), utilizando recursos tecnológicos que demonstram e comunicam conceitos do currículo. **[O18]** Colaborar com colegas, especialistas e outros utilizando práticas colaborativas como programação em pares, trabalho em equipes de projeto, e participação em atividades de aprendizagem ativa em grupo. **[O19]** Exibir disposições necessárias para colaboração: fornecer feedback útil e integrante, compreender e aceitar múltiplas perspectivas, socialização.
Programação
[O20] Selecionar ferramentas e recursos tecnológicos apropriados para realizar tarefas variadas e resolver problemas. **[O21]** Usar uma variedade de ferramentas e periféricos de multimídia para apoiar a produtividade e aprendizagem pessoal durante todo o currículo.

[O22] Conceber, desenvolver, publicar e apresentar produtos (p.ex., páginas web, aplicações móveis, animações) usando recursos de tecnologia que demonstram e comunicam os conceitos do currículo. **[O23]** Demonstrar uma compreensão de algoritmos e a sua aplicação prática. **[O24]** Implementar soluções de problema utilizando uma linguagem de programação, incluindo: o comportamento de laços (sequências de instruções que se repetem), instruções condicionais, lógica, expressões, variáveis e funções. **[O25]** Demonstrar boas práticas na segurança da informação pessoal, usando senhas, encriptação e transações seguras. **[O26]** Identificar carreiras interdisciplinares que são abrangidas pela ciência da computação. **[O27]** Demonstrar receptiva disposição para resolver e programar problemas indeterminados (p.ex. conforto com complexidade, persistência, brainstorming, adaptabilidade, paciência, tendência a mexer, criatividade, aceitação de mudanças). **[O28]** Coletar e analisar dados que correspondem à saída de múltiplas execuções de um programa de computador.
Computadores e dispositivos de comunicação
[O29] Reconhecer que os computadores são equipamentos que executam programas. **[O30]** Identificar uma variedade de dispositivos eletrônicos que contêm processadores computacionais. **[O31]** Demonstrar compreensão sobre a relação entre hardware e software. **[O32]** Usar terminologia adequada ao desenvolvimento e, precisa na comunicação sobre tecnologia. **[O33]** Aplicar estratégias para identificar e resolver problemas de rotina de hardware que ocorrem no uso de computador diariamente. **[O34]** Descrever os principais componentes e funções de sistemas de computadores e redes. **[O35]** Descrever o que distingue os seres humanos de máquinas, dando um enfoque na inteligência humana contra a inteligência de máquina e, formas que podemos nos comunicar. **[O36]** Descrever maneiras em que os computadores usam modelos de comportamento inteligente (p.ex., movimento de robô, fala e compreensão da linguagem e, visão computacional).
Impactos éticos, globais e na comunidade
[O37] Apresentar comportamentos legais e éticos no uso de informação e tecnologia e, discutir as consequências do uso indevido. **[O38]** Demonstrar conhecimento das mudanças nas tecnologias de informação ao longo do tempo e os efeitos destas mudanças na educação, no local de trabalho e na sociedade. **[O39]** Analisar os impactos positivos e negativos da computação na cultura humana. **[O40]** Avaliar a precisão, relevância, adequação, abrangência, e viés de fontes de informação eletrônicas referentes a problemas do mundo real. **[O41]** Avaliar a precisão, relevância, adequação, abrangência, e viés de fontes de informação eletrônicas referentes a problemas do mundo real. **[O42]** Discutir como a distribuição desigual de recursos de computação em uma economia global levanta questões de equidade, acesso e poder.

Computação no Ensino Básico é muitas vezes ensinado por meio de unidades instrucionais que ensinam a programação de código (WING, 2006). Neste contexto, o aluno pode ser instruído a realizar exercícios de programação predefinidos, como por exemplo completar um trecho de código faltante. Geralmente esses exercícios tem apenas uma resposta correta e, portanto, a análise e avaliação do mesmo refere-se apenas a uma comparação com um gabarito. Outros exercícios focam na criação de soluções para problemas do mundo real, em que as soluções são artefatos de software, como por exemplo, jogos e animações. Essas atividades de computação estão ligadas a aprendizagem baseada em problemas, que são problemas complexos e abertos a várias possíveis soluções (LYE & KOH, 2014).

2.3 ANÁLISE DE CÓDIGO

Análise de código é o processo de extrair informação sobre um programa através do código fonte ou artefatos derivados do código fonte, por exemplo, *Java byte code,* utilizando ferramentas automatizadas (BINKLEY, 2007). A análise de código pode ser estática ou dinâmica. A análise dinâmica de código é feita por meio da execução do código (GOMES, 2009). Já a análise estática de código, foco do presente trabalho, é a análise de código realizada sem execução do código (GOMES, 2009). Os componentes típicos da análise de código são (BINKLEY, 2007):

- ***Parser:*** converte o código fonte em uma representação intermediária, mais adequada para ser analisada.
- **Representação intermediária:** abstração de um aspecto em particular do programa, em uma forma mais adequada para análise automatizada. Diversos tipos de representação intermediárias são utilizadas, sendo o grafo a mais comum.
- **Análise da representação intermediária:** Podem ser realizados diversos tipos de análise, incluindo a análise estática e a análise dinâmica.

Tipicamente, uma ferramenta de análise estática funciona escaneando um ou mais arquivos de código fonte e criando uma representação intermediária do código para analisá-la (Parser), como mostra a Figura 7 (JONES, 2013).

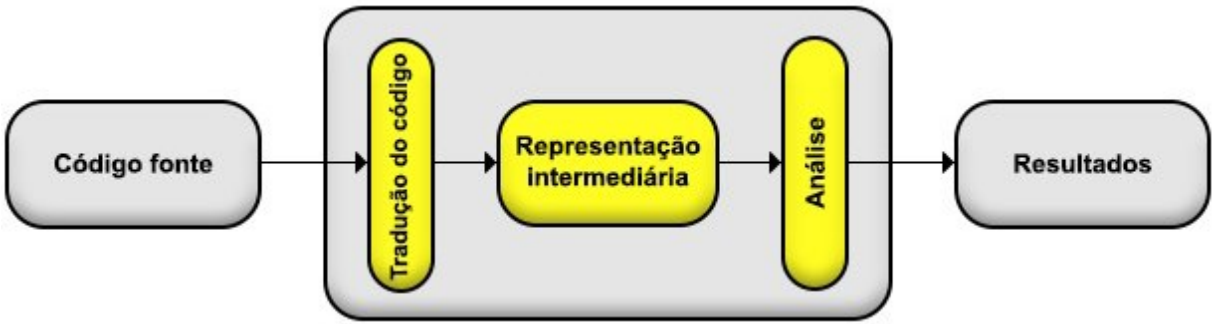


Figura 7 - Exemplo de arquitetura de uma ferramenta de análise estática (JONES, 2013)

Utilizando uma variedade de métodos, ferramentas de análise estática podem descobrir problemas como *memory leaks*, *buffer overflows* e até problemas de concorrência (JONES, 2013). Ferramentas de análise estática de código podem detectar diversos tipos de defeitos de software, por exemplo, defeitos lógicos ou funcionais (o software eventualmente computa valores incorretos) ou defeitos em tempo de execução (EMANUELSSON; NILSSON, 2008). Uma ferramenta de análise estática de código

também pode detectar vulnerabilidades sutis na segurança do software, que podem ser exploradas por ataques maliciosos (EMANUELSSON; NILSSON, 2008). Compiladores também usam análise estática de código para forçar, de forma consistente, que o programador utilize de forma correta a sintaxe da linguagem (GOMES, 2009). A análise estática de código é mais rápida, em comparação com, por exemplo, a revisão manual do código (GOMES, 2009). Isto significa que por meio da análise estática de código é possível avaliar programas com maior frequência (GOMES, 2009).

2.3.1 ANÁLISE ESTÁTICA DE CÓDIGO NO CONTEXTO EDUCACIONAL

No contexto educacional, as ferramentas de análise estática de código podem ser utilizadas para contribuir para a aprendizagem do aluno, permitindo a ele desenvolver soluções corretas para exercícios, como atividades práticas de programação, sem uma assistência intensa do professor (STRIEWE; GOEDICKE, 2014). Analisando o código em desenvolvimento, a ferramenta automaticamente orienta o aluno sobre problemas detectados. A orientação foca em dicas úteis, para código com soluções incorretas ou incompletas, que vão além de mensagens básicas como “Erro na linha X” (STRIEWE; GOEDICKE, 2014). Outro objetivo é a avaliação automatizada de atividades práticas de programação. A avaliação automatizada destina-se a ajudar os professores na tarefa de avaliar projetos desenvolvidos por alunos (STRIEWE; GOEDICKE, 2014). Substituir a avaliação manual dessas atividades práticas por avaliação automatizada pode também reduzir a carga de trabalho dos professores. Os objetivos têm em comum a geração de *feedback* significativo, de forma automatizada, baseado em uma análise do código fonte apresentado pelos alunos (STRIEWE; GOEDICKE, 2014).

Voltado ao foco do presente trabalho os aspectos importantes que, tipicamente, devem ser analisados neste contexto são (RASHKOVITS & LAVY, 2013) (MORENO & ROBLES, 2015):

- **Abstração e Modularização:** O código deve ser eficientemente organizado em classes e as classes devem ser organizadas em hierarquias. Cada classe representa um único conceito e tem todos os atributos e métodos necessários.
- **Paralelismo:** O código deve priorizar o paralelismo na execução de atividades não relacionadas ou que não dependem uma da outra para execução.
- **Sincronização:** O código deve conter componentes capazes de sincronizar a execução de vários elementos dinâmicos da aplicação.

- **Controle de fluxo:** As instruções, expressões e chamadas de métodos devem ser corretamente ordenadas. O uso de condicionais, laços e chamada de sub-rotinas deve ser eficiente.
- **Interatividade com usuário:** A aplicação deve ser interativa com o usuário. O usuário deve ser participante.
- **Representação de dados:** A aplicação deve representar e armazenar os dados coletados de forma eficiente, e de forma a garantir a consistência dos dados.

No foco do presente trabalho necessita-se então a medição destes aspectos com base em trabalhos práticos, que possuam várias possíveis soluções a serem desenvolvidas pelos alunos. Neste caso a análise e avaliação de código requer mais flexibilidade, pois, possivelmente existem diversas soluções corretas ao problema. Assim, se faz necessário um analisador de código que realize análise e avaliação mais flexível e livre do código.

Para a avaliação dos trabalhos práticos de programação podem ser adotadas rubricas. Uma rubrica é uma ferramenta de pontuação para avaliação de trabalhos de estudantes (JONSSON; SVINGBY, 2007). Uma rubrica inclui critérios para avaliação do desempenho, bem como padrões de satisfação desses critérios. A rubrica diz ao instrutor e ao aluno o que é considerado importante (JONSSON; SVINGBY, 2007).

Um exemplo de uma rubrica para avaliação de atividades práticas de programação com linguagens de programação visual baseada em blocos é apresentado na Tabela 2.

Tabela 2 - Rubrica de avaliação do Dr.Scratch (MORENO & ROBLES, 2015)

Critério	**Nível de performance**			
	0 pontos	**1 ponto**	**2 pontos**	**3 pontos**
Lógica	Não utilizou nenhum comando de lógica.	Utilização do comando “Se, então”.	Utilização do comando “Se, então; senão”.	Utilização de comandos de operações lógicas que combinam as condições.
Paralelismo	Não utilizou nenhum comando de paralelismo.	2 scripts iniciando com “bandeira verde”.	2 scripts com o comando “quando a tecla for pressionada” utilizando a mesma tecla ou 2 scripts com o comando “quando este ator for clicado” utilizando o mesmo ator.	2 scripts de recebimento de mensagens ou criação de clones ou 2 scripts de sensores ou 2 scripts de mudança de pano de fundo.
Interatividade com o usuário	Não utilizou nenhum comando de interatividade com o usuário.	Utilização do comando da “bandeira verde”.	Utilização de comandos de teclas/atores pressionadas, perguntas ao usuário, e botão do mouse.	Utilização de comandos de sensor de câmera e vídeo.

Representação de dados	Não utilizou nenhum comando de representação de dados.	Modificação de propriedades dos atores como coordenadas, tamanho e aparência.	Operações com variáveis.	Operações com listas.
Controle de fluxo	Não utilizou nenhum comando de controle de fluxo.	Programação de uma sequência de blocos.	Utilização dos comandos “repita x vezes” e “sempre”.	Utilização do comando “repita até que”.
Sincronização	Não utilizou nenhum comando de sincronização.	Utilização do comando “espere”.	Utilização de comandos de envio/recebimento de mensagens.	Utilização de comandos de “espere até” ou “quando o fundo muda para”.
Abstração	Não utilizou nenhum comando de abstração.	Programação de mais de 1 *script*.	Utilização e programação de funções por meio do comando “defina”.	Utilização de clones.

Feedback

Um *feedback* é uma técnica para orientar e estimular os alunos a refletirem sobre suas respostas, fornecendo informações que direcionem a mudança de sua maneira de pensar e agir, assim promovendo a sua aprendizagem (LIU, 2010). Este pode ser considerado uma resposta às ações do aluno, e pode ser elaborado seguindo diferentes abordagens, por exemplo: pela verificação da exatidão das respostas; ou sendo mais abrangente, provendo informações relacionadas ao conteúdo, como dicas, sugestões, exemplos, etc. (BLACK & WILIAN, 1998). O principal objetivo do *feedback* é auxiliar no desenvolvimento de competências sobre um determinado assunto, identificando nas respostas apresentadas pelo aluno, se estas já estão corretas e completas, e caso contrário, indica ao aluno os aspectos que ainda precisam ser corrigidos ou melhorados (RICHARDS & SCHIFFEL, 2005). Entre estas informações está a explicitação do desempenho do aluno e a análise das respostas apresentadas, identificando erros ou pontos de melhoria. A explicitação do desempenho avalia a exatidão ou a completude de uma resposta, comparando o comportamento apresentado pelo do aluno, com aquele esperado pelos objetivos de desempenho (JOHNSON & JOHNSON, 1993; AULD, BELFIORE, & SCHEELER, 2010).

Tipicamente, quando uma ferramenta de análise de código é utilizada para avaliar um único projeto, espera-se que o *feedback* seja instrucional. O resultado deve ser apresentado de forma detalhada, para que o aluno possa aperfeiçoar seu projeto. Por outro lado, se o objetivo é suportar o professor na avaliação de atividades de uma turma inteira, espera-se que o *feedback* seja voltado para gestão de turmas. Neste caso a ferramenta

também deve enviar o *feedback* individual, e detalhado, para os alunos. Uma proposta para realizar gestão de turmas de forma eficiente é que o professor receba todos os projetos desenvolvidos por meio de uma única plataforma, para que ele possa enviar todos os projetos para análise de uma só vez. A ferramenta de análise deve avaliar todos os projetos e apresentar todos os resultados para o professor, de forma sucinta.

No Brasil não existe um consenso e nem regra de como as notas devem ser dadas às provas e trabalhos práticos nas escolas. Cada estado ou município fica encarregado de definir qual padrão seguir e mesmo assim podem haver diferenças entre a rede estadual e municipal além de pública e privada. Como exemplo na cidade de Florianópolis - SC a RESOLUÇÃO CME Nº02/2011 Art. 7º diz que as notas podem ser em "[...]parecer descritivo que revele o diagnóstico do processo de aprendizagem" e em número de 1 a 10.

2.4 AMBIENTE E LINGUAGEM DE PROGRAMAÇÃO SNAP!

Uma alternativa para ensinar computação na Educação Básica é o uso de linguagens de programação visual baseada em blocos. Um exemplo é a linguagem Snap! (SNAP!; BERKELEY, 2013). Snap! (antigamente BYOB) é uma implementação estendida do Scratch, um ambiente de programação muito popular desenvolvido pelo *Lifelong Kindergarten Group* do MIT Media Lab (SNAP!, 2013). Com Snap! é possível criar jogos e animações, e visualizar sua execução. Para a criação destes pode-se usar laços de repetição, execução condicional, eventos, operadores matemáticos e *booleanos*, listas entre outros recursos.

Snap! versão 4.0 é compatível com dispositivos iOS, OS X, Windows e Linux sendo uma ferramenta *web*. Snap! é um software *open-source* disponível em: <http://snap.berkeley.edu/run> (SNAP!, 2013).

Elementos do Snap!

O Snap! utiliza elementos característicos de ambientes/linguagens de programação baseada em blocos. A Tabela 3 apresenta os principais elementos e suas respectivas descrições.

Tabela 3 - Principais elementos da linguagem Snap!

Elemento	Tradução (Português)	Descrição
Sprite	Ator	Um objeto individual que pode mudar a aparência de acordo com as fantasias, emitir sons e executar outras ações em um palco.
Palette	Categoria	Uma divisão entre blocos para agrupá-los de forma que faça sentido.

Script	Roteiro	Um ou mais blocos que realizam ações como movimentar, alterar aparência de um ator, etc.
Stage	Palco	Onde um ou mais atores executam um ou mais roteiros. Os atores utilizam sons e fantasias que estão no Palco.
Block	Bloco	Uma instrução que pode ou não fazer parte de um roteiro.
Costumes	Fantasias	Imagens que um ator ou palco pode assumir para alterar sua aparência. O usuário pode usar fantasias predefinidas ou criar suas próprias fantasias.
Sounds	Sons	Sons que um ator ou palco pode emitir.
Sprite Corral	Curral	Local onde ficam todos os atores e palcos do projeto.

Blocos de programação do Snap!

Um programa Snap! consiste de um ou mais roteiros. Cada roteiro é feito de um ou mais blocos de comando. A Figura 8 mostra um exemplo de roteiro de um programa Snap! que faz um ator se mover e desenhar um quadrado no palco, emitindo um som para cada aresta desenhada, quando a tecla “espaço” é pressionada.

Figura 8 - Exemplo de um roteiro Snap!

Os blocos apresentados na Figura 8 têm cores diferentes, cada cor correspondente a uma categoria. Todas as oito categorias de blocos são apresentadas na Tabela 4. A lista completa dos blocos disponíveis no Snap! é apresentada no Anexo A.

Tabela 4 - Categorias de blocos da linguagem Snap!

Categoria	Blocos
Movimento	Alteram a posição e ângulo dos atores. Exemplo: mova 50 passos.
Aparência	Alteram a visibilidade, fantasia, efeitos, tamanho, falas do ator, etc. Exemplo: diga “olá” por 2 segundos.
Som	Controlam quais sons devem ser reproduzidos, em que velocidade e volume, etc. Exemplo: toque o som “Latido”.
Caneta	Desenham o que um ator faz na tela, alteram tamanho e cor da caneta, etc. Exemplo: use a caneta.
Controle	Alteram o fluxo de execução como laços, condicionais, etc. Exemplo: quando a tecla “espaço” for pressionada.
Sensores	Detecção de entradas do usuário, sensores de cor, posição do mouse e dos atores, etc. Exemplo: pergunte “qual o seu nome?” e espere.
Operadores	Operadores matemáticos e booleanos. Exemplo: “x” < “y”.
Variáveis	Criar variáveis ou listas, incluindo listas de listas e operá-las. Exemplo: insira “x” na posição “y” de “z”.

Interface gráfica

A Figura 9 apresenta a interface gráfica do Snap! em que cinco regiões são destacadas. Essas são descritas na Tabela 5.

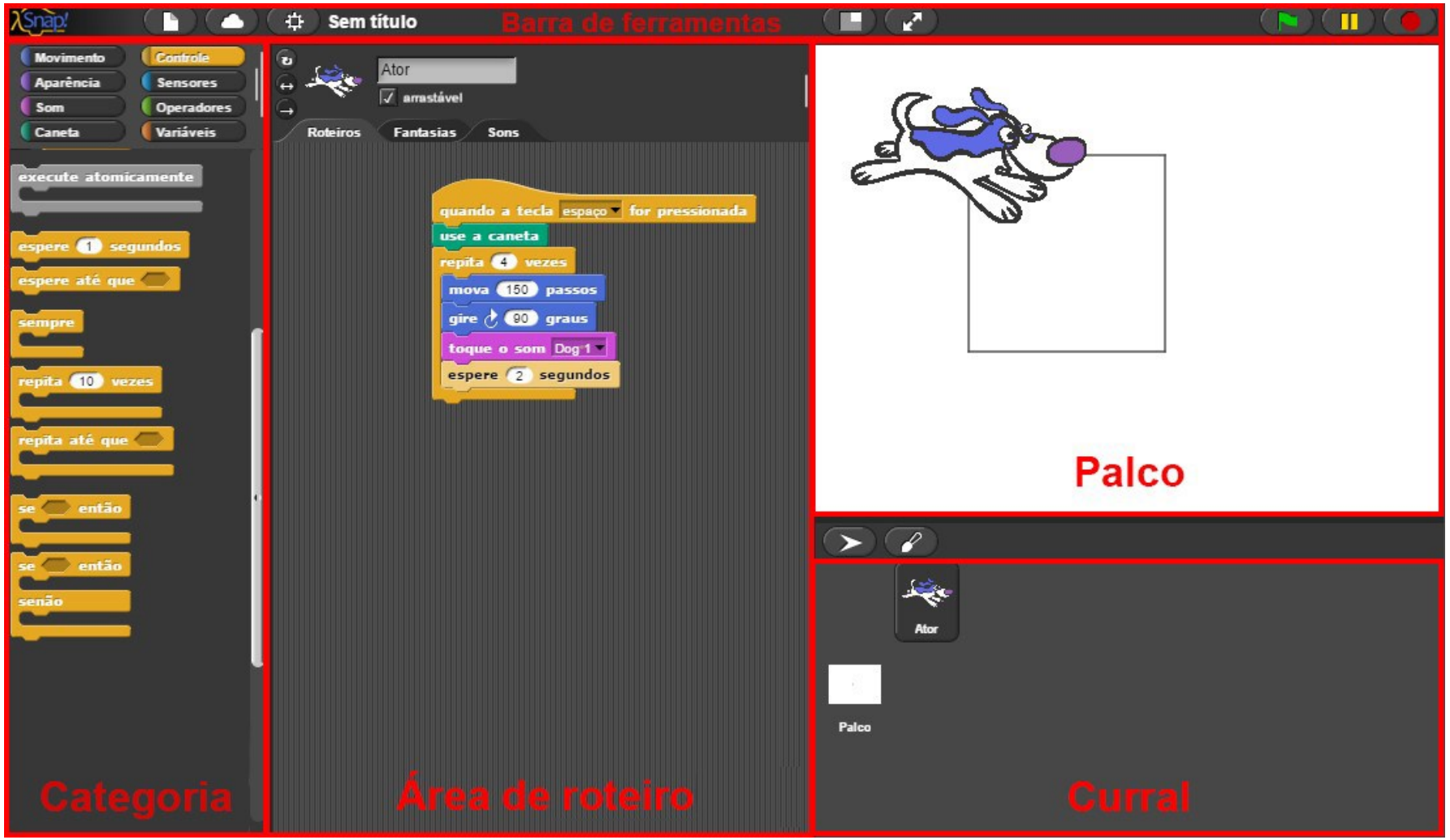


Figura 9 - Interface gráfica do Snap! (SNAP!, 2013)

Tabela 5 - Regiões destacadas na interface gráfica

Região destacada na interface do Snap!	Descrição
Barra de ferramentas	Está presente na parte superior da tela. É nesta região que se encontram os botões para abrir, salvar, importar, exportar um projeto. Também estão nessa região botões para outras funções, como selecionar um idioma, iniciar ou encerrar um roteiro, etc.
Categoria	Está presente no canto esquerdo da tela. É nesta região que se encontram os blocos de uma determinada categoria. Para selecionar uma entre as oito categorias o usuário deve clicar no botão correspondente.
Palco	Está presente no canto superior direito da tela. É nesta região que é possível visualizar os atores e suas ações (execução do roteiro). O usuário pode escolher entre visualização reduzida, visualização expandida ou visualização tela cheia.
Área de roteiro	Está presente na parte central da tela. É nesta região que se encontram os roteiros que serão executados. O usuário pode arrastar um bloco, de cada vez, de dentro da categoria para dentro da Área de roteiro para construir seu roteiro.
Curral	Está presente no canto inferior direito da tela. É nesta região que se encontram os atores e palcos do projeto. O usuário pode criar, duplicar e excluir atores e palcos.

Exportar/Importar projetos

Durante o desenvolvimento de um projeto Snap! o usuário pode exportar seu projeto, em formato XML, para salvá-lo em seu computador. Para exportar o seu projeto o usuário deve clicar no menu "*File*" e clicar na opção de menu "*Export project*" (Figura 10). Caso o projeto ainda não tenha um nome, essa informação será solicitada (Figura 11).

Figura 10 - Opções do menu "File" do Snap!

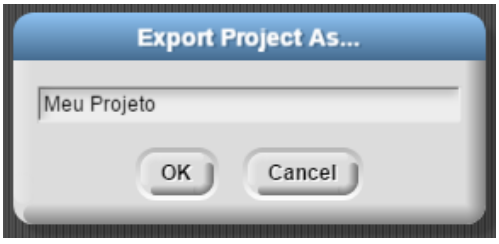


Figura 11 - Definindo o nome do projeto Snap!

O arquivo XML exportado contém diversas informações sobre o projeto. Esse arquivo é estruturado de forma que o ambiente de desenvolvimento Snap! consiga importá-lo novamente. Uma das informações salvas no arquivo são os roteiros utilizados no projeto (Figura 12). As propriedades de um roteiro também são salvas no arquivo XML, como posição, blocos e ordem dos blocos programados.

```
▼<scripts>
  ▼<script x="111" y="31">
    ▼<block s="receiveKey">
      ▼<l>
          <option>space</option>
        </l>
      </block>
      <block s="down"/>
    ▼<block s="doRepeat">
        <l>4</l>
      ▼<script>
        ▼<block s="forward">
            <l>150</l>
          </block>
        ▼<block s="turn">
            <l>90</l>
          </block>
        ▼<block s="playSound">
            <l>Dog 1</l>
          </block>
        ▼<block s="doWait">
            <l>2</l>
          </block>
        </script>
```

Figura 12 - Roteiro no arquivo XML

Outro tipo de informação que pode estar presente no arquivo XML do projeto são as mídias digitais, como sons e imagens. Mídias digitais podem ser armazenadas no projeto de duas formas, dependendo da forma como são importadas na etapa de desenvolvimento: endereço da mídia na internet ou codificadas em Base64. Base64 é um formato que possibilita o armazenamento de dados binários em forma de texto. A Figura 13 apresenta o armazenamento do endereço de um som em um arquivo XML.

```
▼<sounds>
  ▼<list id="10">
    ▼<item>
        <sound name="Dog 1" sound="http://snap.berkeley.edu/snapsource/Sounds/Dog1.wav" id="11"/>
      </item>
    </list>
  </sounds>
```

Figura 13 - Áudio no arquivo XML

A Figura 14 apresenta o armazenamento de uma imagem codificada em Base64 dentro do arquivo XML.

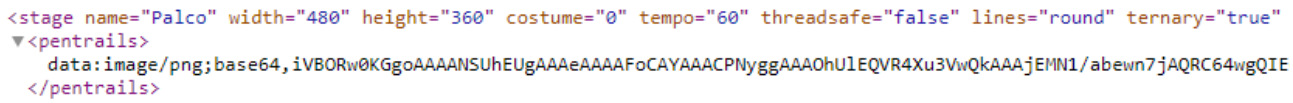


Figura 14 - Imagem no arquivo XML

3. ESTADO DA ARTE

Neste capítulo é apresentado o estado da arte em relação a analisadores estáticos de código de ambientes de programação visual (baseado em blocos), utilizados no contexto educacional. A análise do estado da arte é realizada por meio de um estudo de mapeamento (PETERSEN, et al., 2008).

3.1 DEFINIÇÃO DO PROTOCOLO DE ESTUDO

O estudo de mapeamento tem como objetivo identificar, avaliar e interpretar pesquisas disponíveis por meio de critérios de qualificação claros e reproduzíveis em relação ao tema deste trabalho. A questão de pesquisa primária para este estudo de mapeamento é: “Quais ferramentas existem para realizar análise estática de código, de ambientes de programação visual, utilizados no contexto educacional?”. Para responder esta pergunta são examinados artigos publicados nas seguintes bibliotecas e bases de dados digitais:

- IEEE Xplore (http://ieeexplore.ieee.org)
- ACM Digital Library (http://dl.acm.org)
- ScienceDirect (http://www.sciencedirect.com)
- Springer (http://link.springer.com)

Este estudo de mapeamento tem como foco os ambientes de programação visual baseado em blocos Snap!, Scratch, App Inventor e Blockly. Devido ao ambiente de programação Scratch, o primeiro desses, ter sido disponibilizado oficialmente em 2005, foram incluídos trabalhos publicados no período entre janeiro de 2005 e dezembro de 2016, escritos em Inglês ou Português.

Critérios de inclusão e exclusão

São definidos critérios de inclusão para determinar quais artigos são relevantes e critérios de exclusão para determinar quais artigos são irrelevantes no contexto desta pesquisa (Tabela 6).

Tabela 6 - Critérios de inclusão e exclusão do estudo de mapeamento

Critérios de inclusão	Critérios de exclusão
• Artigos sobre análise estática de código de linguagens de programação visual baseada em blocos (Scratch, Snap!, Blockly e App Inventor); e • Artigos sobre avaliação automatizada no contexto educacional.	• Artigos duplicados; • Artigos resumidos; e • Artigos sobre análise de código de linguagens de programação não visual. • Analisadores que realizem análise dinâmica de código. • Analisadores que realizem análise de projetos predeterminados.

Termos de busca

Com base na questão de pesquisa primária foram derivados os termos “Análise estática de código”, “Ambiente de programação visual” e “Educação”. Com base nos critérios de inclusão foram derivados os termos “Scratch”, “Snap!”, “Blockly”, “App Inventor” e “Avaliação”. A Tabela 7 apresenta todos os termos de busca, seus respectivos sinônimos e traduções para inglês.

Tabela 7 - Termos de busca

Termos	Sinônimos	Tradução (inglês)
Análise estática de código	Análise de código, análise estática	Static code analysis, code analysis, static analysis
Ambiente de programação visual	Programação visual, programação baseada em blocos	Visual programming, block programming
Educação	Ensino, aprendizagem	Education, teaching, learning
Avaliação	Classificação	Assessment, grading
Scratch	-	-
Snap!	-	-
Blockly	-	-
App Inventor	-	-

A Tabela 8 apresenta as *strings* de busca, definidas após a identificação dos termos de busca, sinônimos e traduções para inglês.

Tabela 8 - Strings de busca

Tipo	*String* de busca
String completa	(“static code analysis” OR “code analysis” OR “static analysis”) AND (“visual programming” OR “block programming”) AND (“education” OR “teaching” OR “learning”) AND (“assessment” OR “grading”) AND (“scratch” OR “snap!” OR “blockly” OR “app inventor”)
String reduzida	(“static code analysis” OR “code analysis” OR “static analysis”) AND (“scratch” OR “snap!” OR “blockly” OR “app inventor”)

3.2 EXECUÇÃO DA BUSCA

Todas as buscas foram realizadas no mês de Janeiro de 2017. Conforme apresentado na Tabela 9, as buscas retornaram no total 1680 resultados. Primeiramente, foi feita uma análise dos títulos e resumos dos 50 resultados de maior relevância de cada base de dados, para verificar se os mesmos estavam de acordo com os critérios de inclusão. Após esta análise, os artigos potencialmente relevantes para este trabalho foram lidos na íntegra. Ao final somente 4 artigos atenderam os critérios de inclusão.

Tabela 9 - Resumo das buscas

Base Digital	String de busca	Resultados encontrados	Resultados analisados	Artigos relevantes
IEEE Xplore	Completa	2	2	2
	Reduzida	91	50	0
ACM DL	Completa	25	25	2
	Reduzida	1169	50	0
ScienceDirect	Completa	39	39	0
	Reduzida	134	50	0
Springer	Completa	63	50	0
	Reduzida	158	50	0
Total		**1680**	**316**	**4**

A maior parte dos resultados analisados são referentes à análise dinâmica de código, análise de exercícios predeterminados ou análise de linguagens de programação textual. Estes não foram considerados relevantes ao presente trabalho.

Para aumentar a cobertura das publicações, a ferramenta Google Scholar também foi utilizada, por indexar um grande conjunto de dados de várias fontes diferentes (Haddaway et al., 2015). Por meio de buscas, utilizando as strings de busca, na ferramenta

Google Scholar mais uma ferramenta foi identificada. Para completar as informações apresentadas nos artigos encontrados na busca, também foi considerada literatura cinza, materiais adicionais, incluindo por exemplo trabalhos acadêmicos (BALL, 2016). A Tabela 10 apresenta as ferramentas identificadas por meio do estudo de mapeamento.

Tabela 10 - Ferramentas identificadas por meio do estudo de mapeamento

Ferramenta	Referência
λ - An Autograder for Snap!	(BALL, 2016)
Scrape	(WOLZ; HALLBERG; TAYLOR, 2011)
Hairball	(BOE; HILL; LEN; DRESCHLER; CONRAD; FRANKLIN, 2013).
Dr.Scratch	(MORENO-LEÓN; ROBLES, 2015).
Ninja Code Village	(OTA; MORIMOTO; KATO, 2016).

3.3 EXTRAÇÃO E ANÁLISE DOS DADOS

Foram extraídas informações referentes às ferramentas encontradas durante a execução da busca. Cada ferramenta é apresentada em detalhes nas subsecções a seguir.

3.3.1 SCRAPE

Scrape (WOLZ; HALLBERG; TAYLOR, 2011) é um conjunto de ferramentas gratuitas desenvolvidas para ajudar as pessoas a visualizar e entender padrões entre projetos desenvolvidos no ambiente de programação Scratch (Figura 15).

Scrape é uma ferramenta que serve para visualizar dados sobre projetos Scratch. Com Scrape é possível visualizar dados individuais sobre um projeto ou grupo de projetos. Também é possível comparar os projetos uns com os outros. Scrape é útil para responder questões como:

- Quantos projetos usam desvio condicional?
- Quantos desvios condicionais têm em cada projeto?

A ferramenta Scrape apresenta *feedback* estatístico, principalmente a quantidade de blocos programados no projeto.

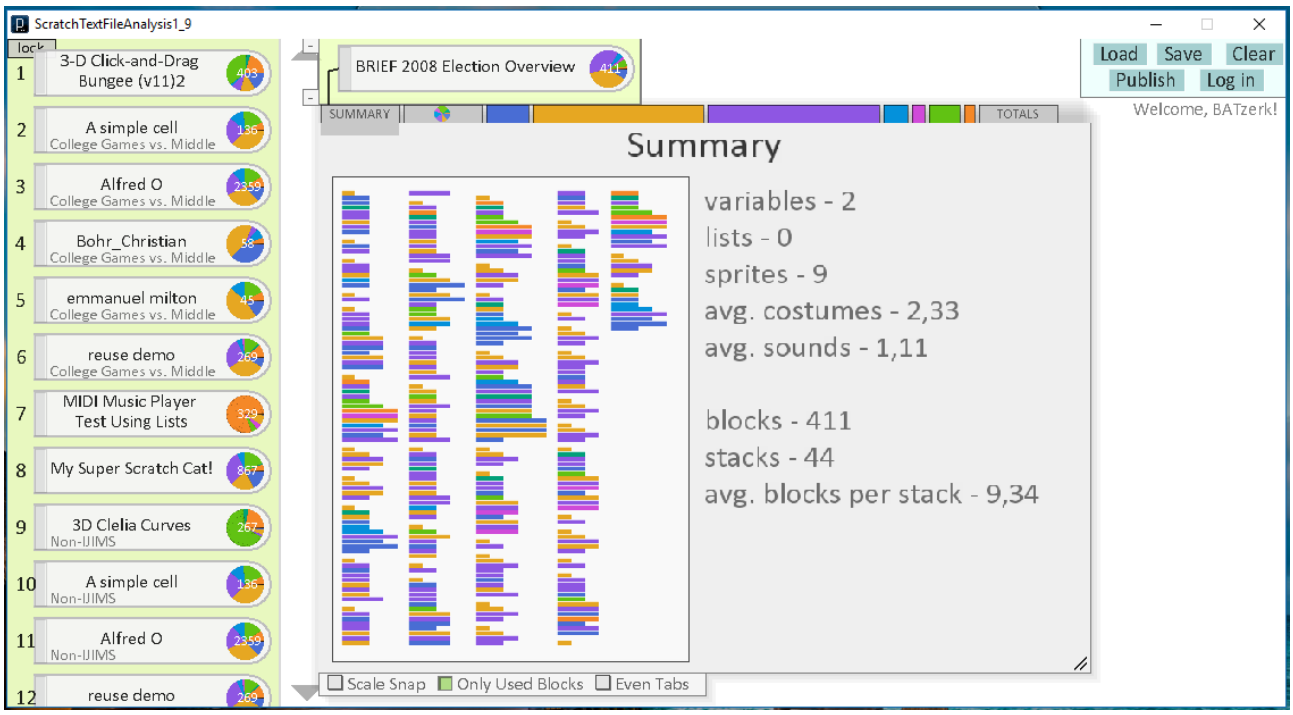


Figura 15 - Analisando um projeto no Scrape

Scrape é uma ferramenta *desktop* disponível em http://happyanalyzing.com/. O projeto está inativo e pode não ser compatível com a versão mais recente do Scratch (HALLBERG, 2017).

3.3.2 HAIRBALL

Hairball é uma ferramenta *open-source* inspirada em *lint* (ferramenta de análise estática que procura possíveis defeitos). É automatizada e pode ser utilizada para apontar práticas não seguras e para auxiliar na avaliação de projetos desenvolvidos na linguagem Scratch (BOE; HILL; LEN; DRESCHLER; CONRAD; FRANKLIN, 2013).

O conjunto inicial de *plugins* detecta sincronização entre blocos, uso de temporização e *loops* em animações complexas (que utilizam fantasias, movimento, temporização e controles de repetição como *loops)*. O Hairball apresenta “Correto”, “Semanticamente incorreto”, “Incorreto” ou “Incompleto” como *feedback* de avaliação, para cada um dos conceitos avaliados.

Embora Hairball tenha sido desenvolvida para auxiliar professores a avaliar projetos de seus alunos, o fato de que é implementado como *scripts Python* impede muitos professores de usar essa ferramenta devido à sua complexidade (MORENO-LEÓN; ROBLES, 2015).

3.3.3 DR.SCRATCH

Dr.Scratch (Figura 16) é uma ferramenta web que realiza análise e avaliação automatizada de projetos Scratch. A ferramenta é inspirada em Scrape e baseada em Hairball (MORENO-LEÓN; ROBLES, 2015).

Dr.Scratch é uma ferramenta que pode ser utilizada por alunos e professores para analisar projetos Scratch e receber uma avaliação da qualidade do projeto (MORENO-LEÓN; ROBLES, 2015).

Dr.Scratch avalia o código dos projetos para atribuir uma pontuação para cada um dos conceitos avaliadas: abstração, lógica, sincronização, paralelismo, controle de fluxo, interatividade do usuário e representação de dados (MORENO-LEÓN; ROBLES, 2015). Cada um desses conceitos é pontuado em uma escala de 0 e 3 pontos (conforme apresenta a Tabela 2), e uma pontuação total é atribuída pela soma das pontuações parciais (MORENO-LEÓN; ROBLES, 2015).

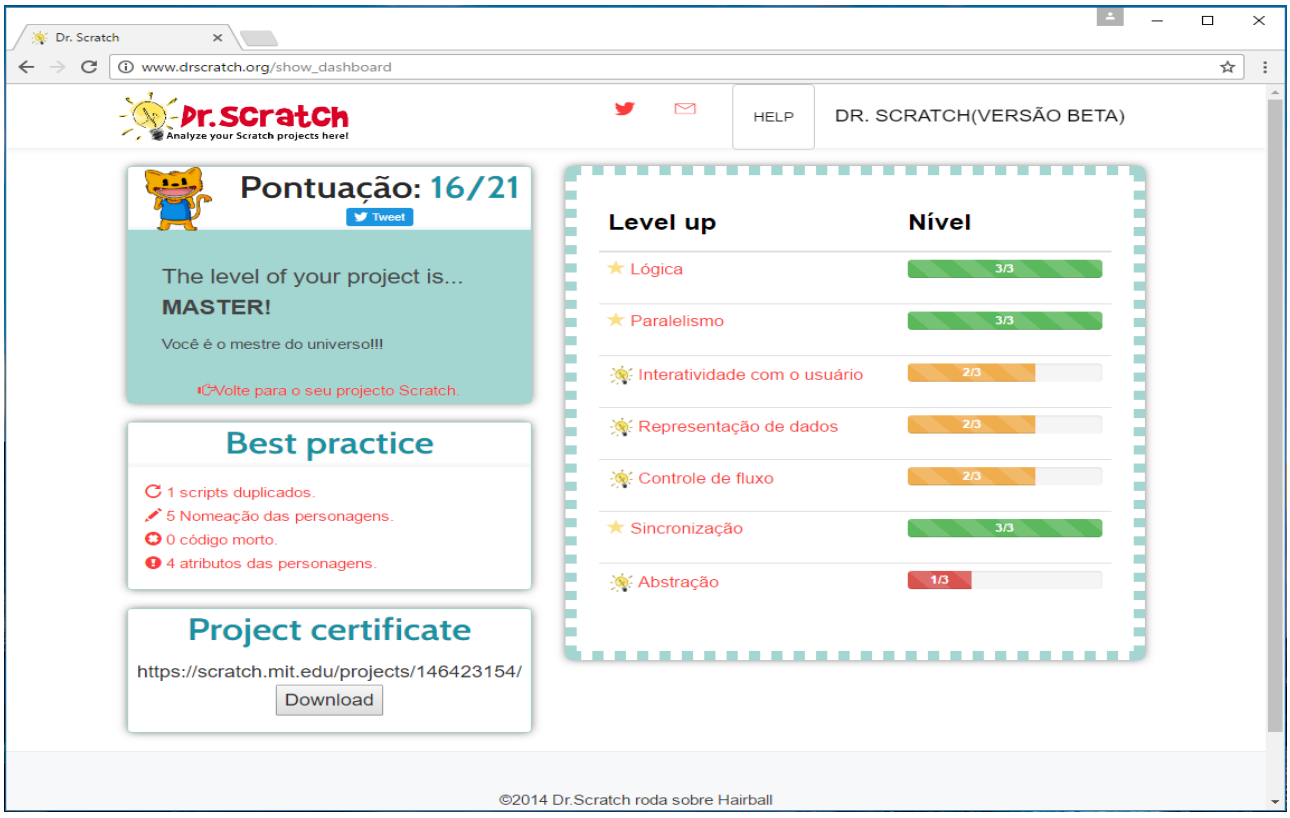


Figura 16 - Analisando um projeto com Dr.Scratch

O feedback do Dr.Scratch apresenta, além das pontuações, um Nível de projeto (p.ex. “O nível do seu projeto é… MASTER!”) e informa o usuário sobre boas práticas (p.ex. “1 *Script* duplicado.”).

Dr.Scratch é uma ferramenta *free/open-source* disponível em http://www.drscratch.org/

3.3.4 NINJA CODE VILLAGE

Ninja Code Village (Figura 17) é uma ferramenta web que realiza análise e avaliação automatizada de projetos Scratch (OTA; MORIMOTO; KATO, 2016).

O NCV não apresenta *feedback* instrucional. Estatísticas sobre o projeto são apresentadas, além da avaliação dos conceitos. Os conceitos avaliados são semelhantes à ferramenta Dr.Scratch, porém o critério de avaliação utilizado pelo NCV é diferente do utilizado pelo Dr.Scratch. Em vez de atribuir uma pontuação para cada um dos conceitos avaliados, o NCV apresenta, em uma tabela, quais conceitos estão presentes no código.

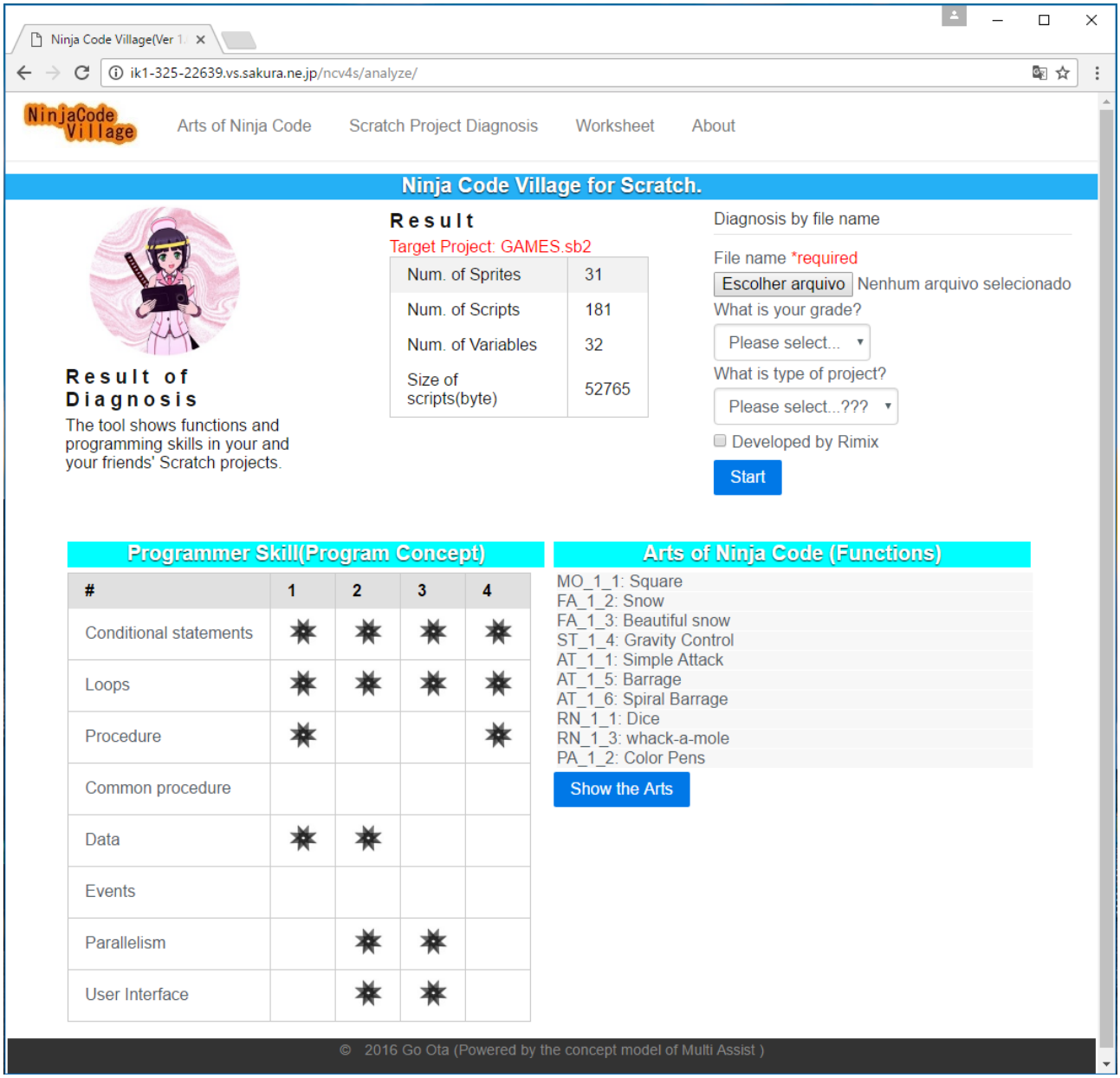


Figura 17 - Analisando um projeto no Ninja Code Village

3.3.5 λ – *AN AUTOGRADER FOR SNAP!*

A ferramenta *λ - An Autograder for Snap*! não satisfaz os critérios de inclusão deste estudo de mapeamento. Porém, sendo a única iniciativa de desenvolvimento de um analisador de código para Snap! identificada, esta foi destacada. *λ - An Autograder for Snap!* é uma ferramenta web que realiza análise dinâmica e avaliação automatizada de projetos Snap! (Figura 18). Usuários não cadastrados podem acessar a interface Snap! com *autograder* embutido para desenvolver soluções para exercícios propostos, submeter o trabalho para avaliação e receber o *feedback* (BALL, 2016).

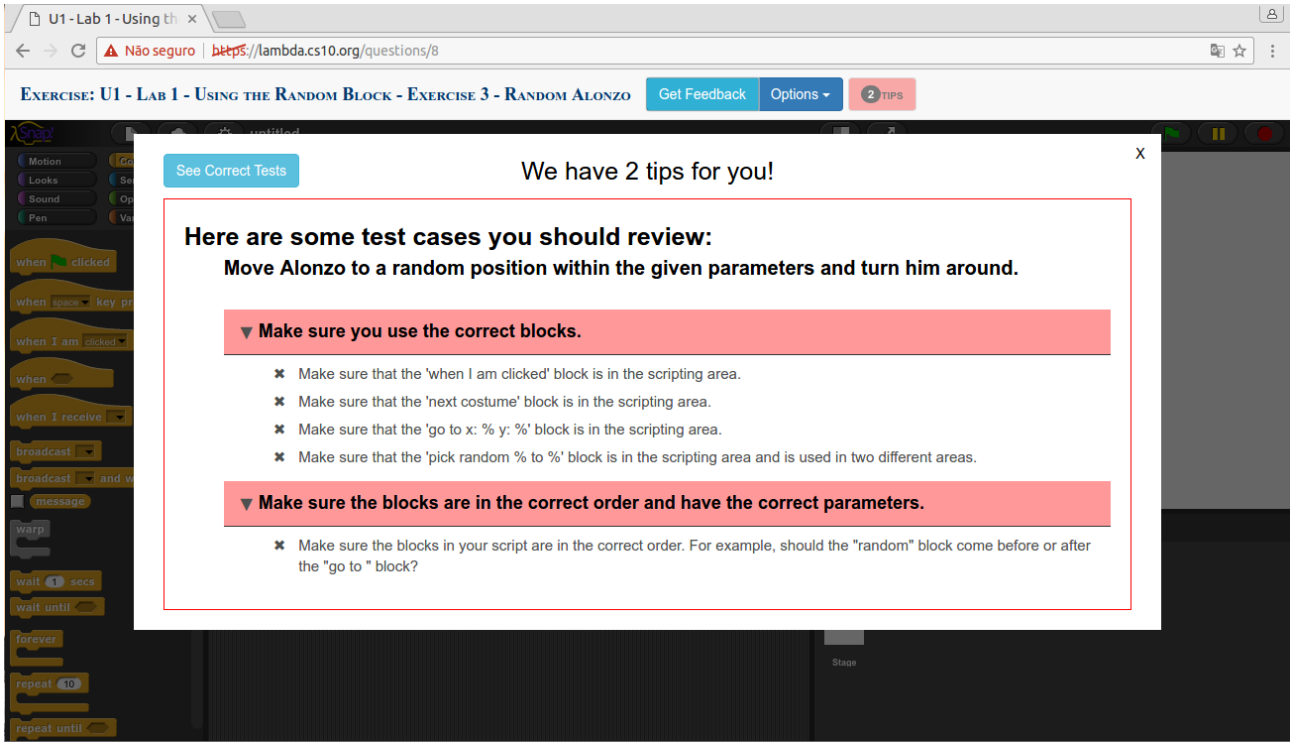


Figura 18 - Analisando um projeto na ferramenta λ

O *feedback* atual fornecido pela ferramenta λ é focado principalmente nos resultados de casos de teste individuais, comparando saídas esperadas com saídas obtidas (BALL, 2016). Os autores de casos de teste são capazes de escrever *feedback* individual para cada caso de teste (BALL, 2016).

λ é uma ferramenta integrada a um curso. Exercícios, casos de teste e a ferramenta de avaliação estão disponíveis gratuitamente na página https://lambda.cs10.org/.

Desta forma são extraídos e analisados os dados somente das ferramentas consideradas relevantes.

3.4 DISCUSSÃO

Durante a execução das buscas poucas ferramentas foram encontradas para realizar análise estática de código de ambientes de programação visual utilizados no contexto educacional. A maioria para analisar projetos desenvolvidos com Scratch. Foi encontrada somente uma ferramenta para Snap!, porém esta realiza análise de exercícios predefinidos, não permitindo a avaliação de projetos desenvolvidos livremente por alunos (BALL, 2016). As Tabelas 11 e 12 apresentam uma análise comparativa entre as ferramentas identificadas por meio do estudo de mapeamento.

Tabela 11 - Comparativo entre as ferramentas identificadas

Ferramenta	Linguagem analisada	Interface	Licença
Scrape	Scratch	Desktop	Free
Hairball	Scratch	Desktop	Open-source
Dr.Scratch	Scratch	Web	Open-source
Ninja Code Village	Scratch	Web	Free

Percebeu-se também que o acesso à ferramentas mais recentes é facilitado por meio da interface web.

Tabela 12 - Comparativo entre as ferramentas identificadas

Ferramenta	Conceitos avaliados	*Feedback*
Scrape	– Lógica – Paralelismo – Interatividade com o usuário – Representação de dados – Controle de fluxo – Sincronização – Abstração	– Quais blocos foram utilizados. – Quantas vezes cada bloco foi utilizado. – Quantos *sounds*, *sprites*, variáveis, listas, *costumes* e *sounds* foram utilizados.
Hairball	– Lógica – Paralelismo – Controle de fluxo – Sincronização	- “Correto”, “Semanticamente incorreto”, “Incorreto” ou “Incompleto” para cada um dos conceitos avaliados.
Dr.Scratch	– Lógica – Paralelismo – Interatividade com o usuário – Representação de dados – Controle de fluxo – Sincronização – Abstração	– Pontuação total entre 0 e 21. – Pontuação parcial entre 0 e 3 para cada um dos conceitos avaliados. – Nível do projeto (p.ex. “*The level of your project is… MASTER!*”). – Boas práticas (p.ex. “1 *Script* duplicado.”).
Ninja Code Village	– Lógica – Paralelismo – Interatividade com o usuário – Representação de dados – Controle de fluxo – Abstração	– Tabela com quais áreas avaliadas estão presentes no código. – Quais funções foram utilizadas. – Quantos *scripts*, variáveis e *sprites* foram utilizados.

A Tabela 12 mostra que tipicamente as ferramentas analisam os conceitos de computação programados em um projeto. Esse tipo de análise permite que qualquer projeto desenvolvido na linguagem seja avaliado. A ferramenta λ é diferente das demais. A ferramenta λ avalia se solução proposta pelo aluno para resolver um exercício está correta. Esse tipo de análise só permite avaliar projetos desenvolvidos para resolver um determinado exercício predefinido.

As ferramentas apresentam os resultados da análise de forma diferente. A ferramenta Scrape apresenta *feedback* estatístico, principalmente a quantidade de blocos programados no projeto. A ferramenta Dr.Scratch apresenta *feedback "*gamificado", uma pontuação parcial para cada conceito avaliado e uma pontuação total é atribuída pela soma das pontuações parciais. A ferramenta Dr.Scratch também apresenta más práticas encontradas no código, como código duplicado. Entre todas as cinco ferramentas, a ferramenta λ apresenta o feedback mais detalhado e instrucional. Isso ocorre, pois, os autores de testes escrevem o *feedback*, individualmente para cada exercício predefinido permitindo a definição de gabaritos.

Apresentar *feedback* instrucional detalhado, por exemplo, "Certifique-se de que o bloco X está sendo utilizado", é uma tarefa complexa para uma ferramenta que realiza análise de diversos projetos. Este tipo de *feedback* se torna possível quando a ferramenta está programada para executar casos de teste predefinidos.

Uma ferramenta de análise e avaliação automatizada, quando utilizada pelo professor, deve apresentar os resultados de forma resumida, para que ele possa ter uma visão geral de todos os projetos avaliados. Já quando é utilizada pelo aluno, deve apresentar os resultados de forma instrucional, para que ele fique ciente de suas dificuldades e como superá-las (p. ex. Dr.Scratch).

Durante a análise do estado da arte, encontrou-se apenas uma ferramenta que realize análise de projetos desenvolvidos com Snap!. Esta ferramenta não avalia se conceitos de computação foram programados, apenas aplica casos de teste para verificar a resolução de exercícios propostos. Assim, a falta atual de uma ferramenta que realize análise e avaliação automatizada de qualquer projeto desenvolvido com o ambiente Snap! demonstra a importância do presente trabalho.

3.4.1 AMEAÇAS À VALIDADE

Existem diversas ameaças à validade de um estudo de mapeamento. Uma possível ameaça é não encontrar trabalhos relevantes. Para mitigar esta ameaça algumas medidas foram tomadas. As buscas foram realizadas nas bases digitais IEEE, Springer Link, ACM e Science Direct por considerar que estas abrangem uma grande quantidade de trabalhos científicos relevantes na área de Engenharia de Software. O mecanismo de busca Google foi utilizado para buscar ferramentas relevantes, mas que não tem nenhum trabalho científico publicado nas bases digitais pesquisadas. Além dos termos de busca extraídos, são utilizados alguns sinônimos e traduções dos termos. A tradução é realizada para a língua inglesa por possuir grande quantidade de publicações nesta língua. Em cada base digital duas *strings* de busca foram utilizadas. A *string* “reduzida” é mais abrangente, visa encontrar um número maior de resultados. Já a *string* “completa” é mais restritiva, visa encontrar resultados de maior relevância.

Um outro risco é na seleção dos artigos. Para mitigar este risco os resultados foram revisados por outros pesquisadores do grupo de pesquisa GQS/INCoD/INE/UFSC até que um consenso foi obtido.

4. DESENVOLVIMENTO DA CODEMASTER – SNAP!

Por meio da análise do estado da arte é possível identificar a necessidade de ferramentas que realizem análise de código para linguagens de programação visual. A proposta apresentada pelo presente trabalho é desenvolver uma ferramenta *web* semelhante à ferramenta Dr.Scratch, que automatize a análise e avaliação de projetos desenvolvidos com Snap!, a ser utilizada em unidades instrucionais para ensinar a computação no Ensino Básico – A CodeMaster.

O desenvolvimento da arquitetura *web* é feito em parceria com o aluno de graduação em Ciências da Computação/INE/UFSC Matheus Faustino Demetrio (DEMETRIO, 2017), cujo trabalho de conclusão de curso é o desenvolvimento de um analisador de código para App Inventor para ensino da computação. Esta parceria também cria, ao final, um *front-end* unificado que permitirá avaliar projetos Snap! e App Inventor. Heliziane Barbosa e Luiz Felipe Azevedo, bolsistas da CnE, também participaram do design das telas, logotipos e imagens dos ninjas utilizados na gamificação dos projetos. A ferramenta proposta deve atender as necessidades de dois tipos de usuários, alunos e professores. Os alunos, sem necessidade de cadastro, devem poder realizar *upload* de apenas um projeto por vez, a ferramenta deve analisá-lo, avaliá-lo e apresentar o resultado de forma completa. Os professores, previamente cadastrados, devem poder realizar upload de múltiplos projetos de uma vez, a ferramenta deve analisá-los, avaliá-los e apresentar os resultados de forma resumida, apenas uma visão geral das notas.

4.1 MODELO CONCEITUAL

As principais funcionalidades da ferramenta são:

- Permitir o *upload* de projetos Snap!, em formato .XML, via interface web.
- Analisar projetos Snap! mapeando os *tokens* e suas respectivas frequências de uso.
- Avaliar projetos Snap! em relação aos conceitos: lógica, paralelismo, interatividade com o usuário, representação de dados, controle de fluxo, sincronização e abstração.
- Apresentar avaliação dos projetos Snap! para o professor. Incluindo em uma tabela as pontuações de cada conceito, pontuação total, nota e nível do projeto, de cada projeto, além das médias de cada conceito entre os projetos submetidos para avaliação.

- Apresentar avaliação de projeto Snap! para o aluno, de forma detalhada, incluindo as pontuações de cada conceito, pontuação total, nota e nível do projeto. A interface deve apresentar também sugestões de como melhorar a pontuação.

Quando um projeto desenvolvido com a linguagem de programação Snap! é exportado, nenhuma informação sobre o(s) autor(es) do projeto é registrada. Para que a ferramenta consiga identificar o(s) autor(es), entre múltiplos projetos, e atribuir as pontuações, notas e níveis corretamente para cada um deles, a ferramenta deve considerar o nome do arquivo do projeto como nome do(s) autor(es). Por exemplo, caso o aluno “João da Silva” seja o autor, o arquivo do projeto deve ser renomeado para “João da Silva.xml”.

4.2 PROCESSO DE ANÁLISE E AVALIAÇÃO

Neste subcapítulo o processo de análise e avaliação de código realizada na ferramenta CodeMaster é apresentado detalhadamente. A partir do *upload* de um projeto Snap! (arquivo XML) a ferramenta CodeMaster analisa, avalia e apresenta os resultados da avaliação do projeto (Figura 19).

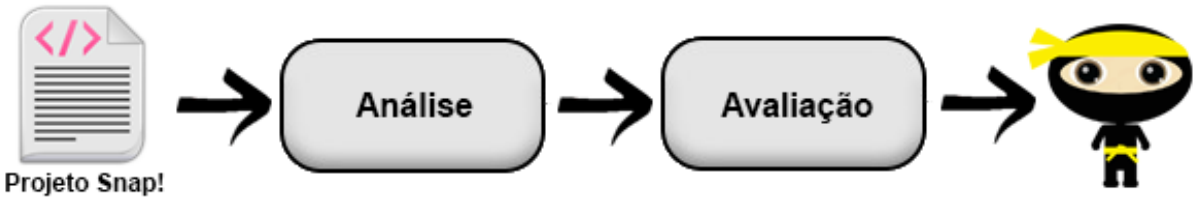


Figura 19 - Processo de análise e avaliação

Processo de análise de código

A ferramenta deve analisar projetos desenvolvidos com a linguagem de programação Snap! versão 4.0. Os projetos devem ser enviados em formato XML por meio de interface web. A análise deve ser realizada em três etapas:

1. O código do projeto, em XML, deve ser lido e convertido em uma *string*, para ser manipulado com maior facilidade.
2. A *string* resultante da etapa 1 deve ser dividida nas ocorrências dos caracteres “aspas”, “espaço em branco”, “nova linha”, “maior que”, “menor que”, resultando em uma lista de *tokens*.

3. A ferramenta deve percorrer a lista de *tokens*, resultado da etapa 2, realizando uma contagem da frequência uso de cada *token*, resultando em uma tabela de *tokens* e frequências de uso.

A Figura 20 exemplifica o processo de análise de um projeto Snap!.

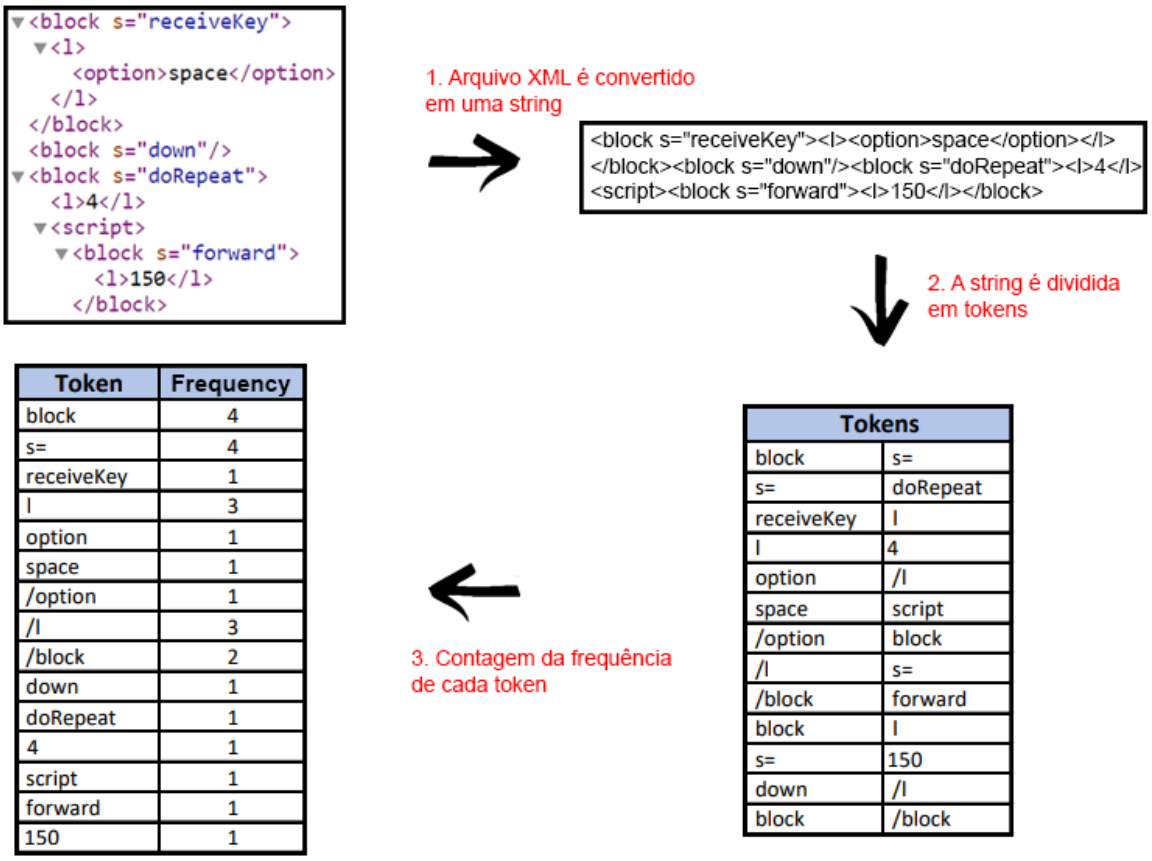


Figura 20 - Processo de análise de um projeto Snap!

Processo de avaliação

Com o objetivo de avaliar a aprendizagem do pensamento computacional, a ferramenta utiliza os dados extraídos do código para avaliar o projeto desenvolvido. Esta avaliação é feita em relação às dimensões-chave do Framework de pensamento computacional apresentado por Brennan & Resnick (2012): conceitos de pensamento computacional, práticas de pensamento computacional e perspectivas de pensamento computacional (Tabela 13), que também foi adotado como base por Dr.Scratch (MORENO-LEÓN; ROBLES, 2015).

Tabela 13 - Dimensões-chave de avaliação

***Framework* de pensamento computacional (BRENNAN; RESNICK, 2012)**	**CodeMaster (Snap!)**
Conceitos de pensamento computacional	
Sequencias	Controle de fluxo
Laços de repetição	Controle de fluxo
Eventos	Interatividade com o usuário
Paralelismo	Paralelismo
Condicionais	Lógica
Operadores	Operadores
Dados	Representação de dados
	Sincronização
Práticas de pensamento computacional	
Ser incremental e iterativo	
Testes e depuração	
Reutilização e remixagem	
Abstração e modularização	Abstração
Perspectivas de pensamento computacional	
Expressando	
Conectando	
Questionando	

A partir destas dimensões chave é definida uma rubrica como base para a ferramenta CodeMaster realizar avaliação de projetos Snap! (Tabela 14).

Tabela 14 - Rubrica de avaliação da ferramenta CodeMaster (Snap!)

Critérios	**Nível de performance**			
	0	**1**	**2**	**3**
Lógica	Não utilizou nenhum comando de lógica.	Utilização do comando “Se, então”.	Utilização do comando “Se, então; senão”.	Utilização de comandos de operações lógicas que combinam as condições.
Paralelismo	Não utilizou nenhum comando de paralelismo.	2 scripts iniciando com “bandeira verde”.	2 scripts com o comando “quando a tecla for pressionada” utilizando a mesma tecla ou 2 scripts com o comando “quando eu for clicado”.	2 scripts de recebimento de mensagens ou criação de clones ou 2 scripts de sensores.
Interatividade com o usuário	Não utilizou nenhum comando de interatividade com o usuário.	Utilização do comando da “bandeira verde”.	Utilização de comandos de teclas/atores pressionadas, perguntas ao usuário, e botão do mouse.	Utilização de comandos de reprodução de sons.

Representação de dados	Não utilizou nenhum comando de representação de dados.	Modificação de propriedades dos atores como coordenadas, tamanho e aparência.	Operações com variáveis.	Operações com listas.
Controle de fluxo	Não utilizou nenhum comando de controle de fluxo.	Programação de uma sequência de blocos.	Utilização dos comandos “repita x vezes” e “sempre”.	Utilização do comando “repita até que”.
Sincronização	Não utilizou nenhum comando de sincronização.	Utilização do comando “espere”.	Utilização de comandos de envio de mensagens com duração de tempo.	Utilização do comando “espere até”.
Abstração	Não utilizou nenhum comando de abstração.	Programação com mais de 1 *script*.	Utilização e programação de funções por meio de blocos personalizados.	Utilização de clones.
Operadores	Não utilizou nenhum tipo de operador.	Utilização de operador de um tipo.	Utilização de operadores de dois tipos diferentes.	Utilização de mais do que dois tipos de operadores diferentes.

Referente ao critério de operadores, a Tabela 15 detalha os tipos de operadores.

Tabela 15 - Tipos de operadores

Tipos	Operadores
Comparações	=, <, >.
Operações matemáticas básicas	+, -, *, /.
Operações matemáticas avançadas	*mod, round, sqrt* e *pick random.*
Operações com *strings*.	*join, split by, letter of, length of, is identical to, unicode of* e *unicode as letter.*

O resultado esperado de uma avaliação inclui uma pontuação para cada conceito avaliado, uma pontuação total, uma nota e um nível de expertise. Ao avaliar um projeto a ferramenta utiliza a rubrica apresentada na Tabela 14 e a tabela de *tokens* e frequências de uso resultante do processo de análise para atribuir **pontuações individuais** para cada conceito. A **pontuação total** de um projeto Snap! é calculada somando cada uma das pontuações individuais.

$$PontuaçãoTotal = \sum PontuaçãoIndividualDeCadaConceito$$

A maior pontuação possível, chamada de **pontuação máxima**, é calculada multiplicando a quantidade de conceitos avaliados (por padrão oito) pela pontuação máxima de cada conceito (três).

$$PontuaçãoMáxima = QuantidadeConceitosAvaliados * 3$$

No Brasil não há uma escala determinada para avaliação dos alunos no Ensino Básico, porém, observa-se que a escala entre 0 e 10 é amplamente utilizada. Visando facilitar o entendimento do professor e do aluno em relação à avaliação de um projeto, a pontuação total deve ser mapeada em uma nota, que pode variar entre 0 e 10. A nota de um projeto é calculada da seguinte forma:

$$Nota = \left(\frac{PontuaçãoTotal}{PontuaçãoMáxima}\right) * 10$$

Considerando a aplicação da ferramenta CodeMaster no contexto educacional com crianças e/ou jovens, o resultado é apresentado também de forma mais lúdica adotando uma gamificação. O termo “gamificação” refere-se ao uso de elementos baseados em jogos em contextos não relacionados a jogos, que visam envolver as pessoas, motivá-las, melhorar a aprendizagem e resolver problemas (DETERDING; DIXON; KHALED; NACKE, 2011). Visando esses benefícios, a ferramenta atribui e apresenta ao usuário um **nível de expertise** gamificado do projeto, além das pontuações e nota. Para isto são utilizadas imagens de Ninjas com faixas de diferentes cores (Figura 21).

Figura 21 - Ninjas do CodeMaster

A cor da faixa utilizada pelo ninja é atribuída com base na nota, como mostra a Tabela 16.

Tabela 16 - Mapeamento entre nota e nível de expertise

Nota	Nível de expertise
De 0,0 até 0,9	Faixa branca
De 1,0 até 1,9	Faixa amarela
De 2,0 até 2,9	Faixa laranja
De 3,0 até 3,9	Faixa vermelha
De 4,0 até 4,9	Faixa roxa
De 5,0 até 5,9	Faixa azul
De 6,0 até 6,9	Faixa turquesa
De 7,0 até 7,9	Faixa verde
De 8,0 até 8,9	Faixa marrom
De 9,0 até 10	Faixa preta

A Figura 22 exemplifica o processo de avaliação de um projeto Snap!.

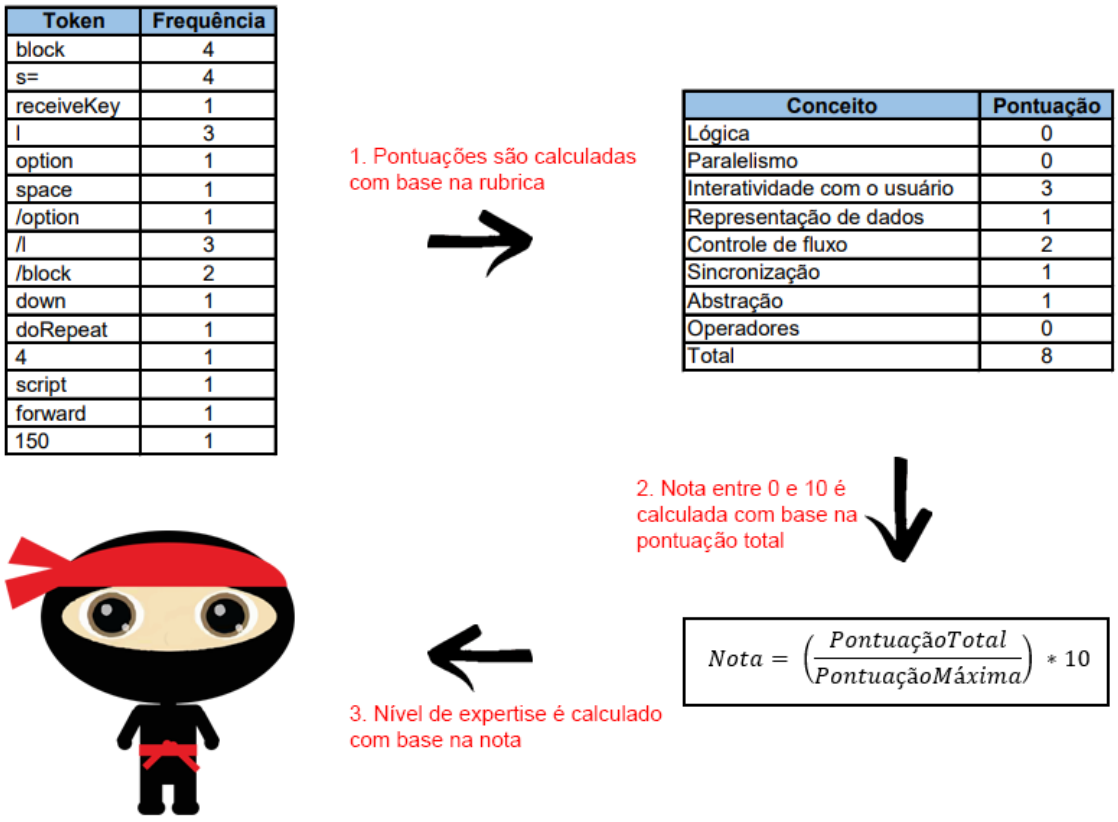


Figura 22 – Processo de avaliação de um projeto Snap!

4.3 ANÁLISE DOS REQUISITOS

Nesta seção são apresentados os requisitos funcionais e não funcionais da ferramenta proposta pelo presente trabalho.

Requisitos funcionais

Os requisitos funcionais da ferramenta, apresentados na Tabela 17, foram elaborados com base no modelo conceitual apresentado na seção 4.1.

Tabela 17 - Requisitos funcionais

ID	Requisito	Descrição	Artefatos	
			Entrada	**Saída**
RF001	Realizar *upload* de projeto Snap!	A ferramenta deve permitir que qualquer usuário realize *upload* de um projeto Snap! no formato XML via web.		Arquivo XML.
RF002	Realizar *upload* de conjunto de projetos Snap!	A ferramenta deve permitir que um usuário cadastrado realize *upload* de um conjunto de projetos Snap! no formato XML via web. O(s) autor(es) de cada projeto são identificados por meio do nome do arquivo.		Conjunto de arquivos XML.
RF003	Analisar projeto Snap!	A ferramenta deve ser capaz de analisar o código de um projeto Snap! identificando a frequência de uso de cada bloco programado no projeto.	*Arquivo* XML.	*Map<String, Integer>* com os *tokens* e suas respectivas frequências de uso.
RF004	Avaliar projeto Snap!	A ferramenta deve ser capaz de avaliar os conceitos de computação, a partir da frequência de uso de blocos (artefato de saída do requisito RF003), seguindo rubrica definida na Tabela 13.	*Map<String, Integer>* com os *tokens* e suas quantidades respectivamente.	Uma avaliação, formada por uma pontuação para cada um dos conceitos avaliados, nota total e nível do projeto.
RF005	Apresentar avaliação detalhada do projeto Snap!	A ferramenta deve ser capaz de apresentar a avaliação feita (artefato de saída do requisito RF004) em sua interface com o usuário de forma detalhada.	Uma avaliação, formada por uma pontuação para cada um dos conceitos avaliados, nota total e nível do projeto.	Interface com o usuário apresentando a avaliação (as pontuações de cada conceito, pontuação total e nível do projeto). A interface deve apresentar também sugestões de como melhorar a pontuação (citando os próximos níveis na rubrica).

RF006	Apresentar avaliação resumida dos projetos Snap!	A ferramenta deve ser capaz de apresentar múltiplas avaliações feitas (múltiplos artefatos de saída do requisito RF004) em sua interface com o usuário de forma resumida.	Um conjunto de avaliações, formadas por uma pontuação para cada um dos conceitos avaliados, nota total e nível do projeto.	Interface com o usuário apresentando em uma tabela as avaliações (pontuações de cada conceito, pontuação total e nível do projeto) além das médias de cada conceito entre as avaliações submetidas para análise.
RF007	Realizar cadastro de professor	A ferramenta deve permitir que o professor se cadastre no sistema por meio da interface *web*.	Nome completo, E-mail e senha.	Interface com o usuário apresentando mensagem de confirmação caso o cadastro seja bem-sucedido ou mensagem de erro caso contrário. Inserir o cadastro no banco de dados.
RF008	Recuperar senha de professor	A ferramenta deve ser capaz de enviar um e-mail para o professor que deseja recuperar sua senha.	E-mail	Interface com o usuário apresentando a mensagem de confirmação, caso o e-mail seja válido, ou mensagem de erro caso o e-mail seja inválido. Enviar e-mail informando a senha do usuário para o e-mail informado.
RF009	Realizar *Login*	A ferramenta deve permitir que o usuário autentique sua identidade por meio de *login*.	E-mail e senha.	Redirecionamento para a página principal dos professores caso *login* bem-sucedido ou mensagem de erro caso contrário.
RF010	Realizar *Logout*	A ferramenta deve permitir que o usuário encerre a sessão atual de sua identidade por meio de *logout*.	*Login* do professor e sessão atual.	Redirecionamento para a página principal do sistema e encerramento da sessão atual.
RF011	Manter registro de avaliações do professor	A ferramenta deve ser capaz de manter registro com todas as avaliações de projetos Snap! realizadas por um determinado professor.	*Login* do professor e as avaliações realizadas durante a sessão atual.	As avaliações são adicionadas ao banco de dados.
RF012	Exibir todas as avaliações do professor	A ferramenta deve ser capaz de buscar o registro com todas as avaliações de projetos Snap! realizadas por um determinado professor e exibir na interface com o usuário.	*Login* do professor e acesso ao banco de dados que possui o registro das avaliações.	Interface com o usuário apresentando em uma tabela as avaliações (pontuações de cada conceito, pontuação total e nível do projeto) além das médias de cada conceito entre as avaliações, de todas as avaliações que um determinado professor já realizou no sistema.

RF013	Exibir estatísticas de uso	A ferramenta deve ser capaz de buscar o registro de todas as avaliações de projetos Snap! já realizadas por meio do sistema e exibir estatísticas de uso na interface com o usuário. Devem ser apresentadas as médias de cada conceito de todas as avaliações, quantidade de professores cadastrados e quantidade de projetos avaliados.	Acesso ao banco de dados que possui o registro das avaliações	Interface com o usuário apresentando em uma tabela as médias de cada conceito entre as avaliações, de todas as avaliações já realizadas no sistema. Apresentando também quantos professores estão cadastrados no sistema e quantos projetos já foram avaliados.
RF014	Exibir rubrica de avaliação	A ferramenta deve ser capaz de exibir a rubrica utilizada pela ferramenta para realizar avaliações de projetos Snap!.		Interface com o usuário apresentando em uma tabela a rubrica utilizada pela ferramenta para realizar avaliações de projetos Snap!.

Requisitos não funcionais

Os requisitos não funcionais da ferramenta são apresentados na Tabela 18.

Tabela 18 - Requisitos não funcionais

ID	Nome	Descrição
RNF001	Sistema web	A ferramenta deve ser acessada via navegador web com conexão à internet. Navegadores compatíveis: Google Chrome versão 56.0.2924.87; Mozilla Firefox versão: 51.0.1; e Microsoft Edge versão 38.14393.0.0.
RNF002	Linguagem de programação	A linguagem de programação utilizada para implementação da ferramenta e arquitetura do sistema deve ser a linguagem Java. A facilidade de encontrar profissionais familiarizados com a linguagem, facilidade de manutenção e produtividade são os principais motivos para a escolha.
RNF003	Políticas de privacidade	A ferramenta deve apresentar no rodapé da interface com o usuário as informações sobre as políticas de privacidade adotadas ou *link* para a página com essas informações.
RNF004	Termos de uso	A ferramenta deve apresentar no rodapé da interface com o usuário os termos de uso adotados ou *link* para a página com essas informações.
RNF005	Usabilidade	- Eficácia: 90% dos usuários devem conseguir completar a tarefa de analisar um ou múltiplos projetos sem assistência. - Satisfação: Pontuação total SUS: 80 pontos.
RNF006	Design de interface	O estilo de design de interface utilizado no sistema deve estar alinhado ao estilo do design de interface do site CnE (http://www.computacaonaescola.ufsc.br)
RNF007	Extensibilidade	A arquitetura desenvolvida deve permitir que outros analisadores de código, futuramente desenvolvidos, sejam acrescentados.
RNF008	Internacionalização	O sistema deve contar a opção de textos em português em inglês.

4.4 CASOS DE USO

Atores do sistema são usuários e elementos computacionais que interagem com o sistema. Os atores da ferramenta proposta pelo presente trabalho são apresentados na Tabela 19.

Tabela 19 – Atores do sistema

Ator	Descrição
A01 – Aluno	Envia projeto Snap! para avaliação e vê o resultado da avaliação.
A02 – Professor	Envia um conjunto de projetos para avaliação e vê o resultado das avaliações. Também acessa avaliações de projetos que enviou no previamente.
A03 – Pesquisador	Acessa as estatísticas de todas as avaliações presentes no banco de dados.

Analisando o modelo conceitual da seção 4.1, os requisitos da seção 4.2 e os atores do sistema apresentados na Tabela 19, foram elaborados os casos de uso do sistema. A Figura 23 apresenta o diagrama de Casos de Uso da ferramenta CodeMaster (Snap!).

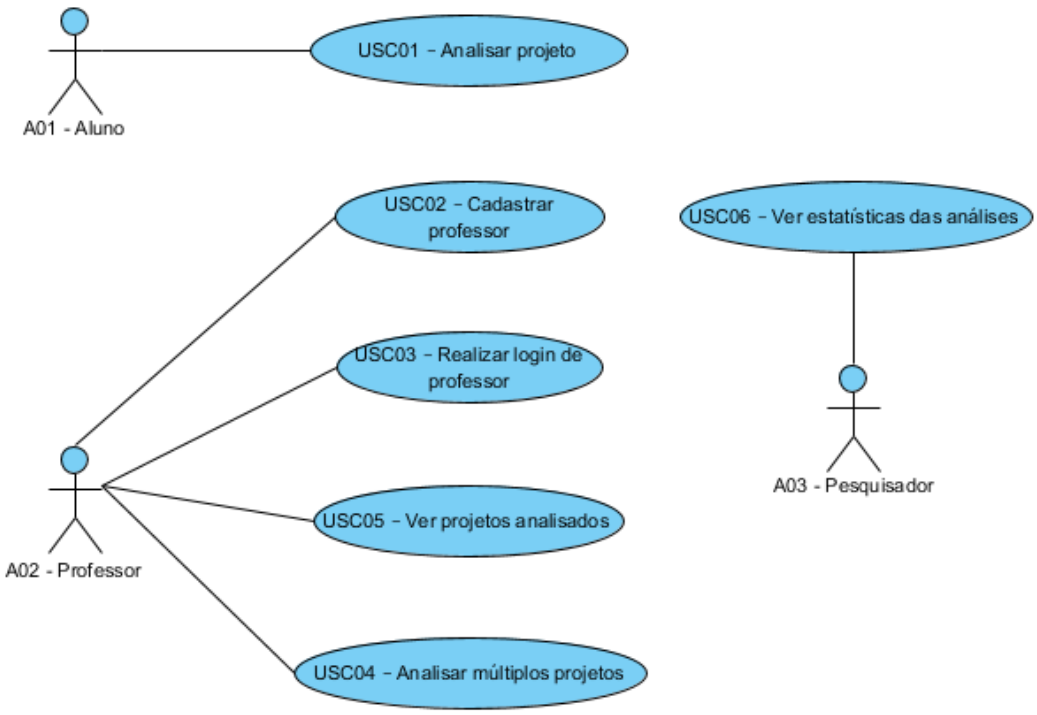


Figura 23 - Diagrama de Casos de Uso

Caso de Uso 1 – USC01 Analisar projeto

Atores:

- A01 – Aluno

Fluxo principal:

1. Aluno acessa o site do CodeMaster.
2. Aluno clica no botão “escolher arquivo”.
3. Aluno escolhe arquivo de projeto Snap! em seu computador.
4. Aluno clica no botão “avaliar”.
5. Sistema realiza a análise/avaliação do código do projeto.
6. Sistema retorna o resultado da análise do projeto.

Fluxo de exceção:

3a. Aluno clica no botão “avaliar” sem escolher projetos. Sistema informa uma mensagem de erro: “Nenhum projeto foi selecionado para análise!”.

Caso de Uso 2 – USC02 Cadastrar professor

Atores:

- A02 – Professor

Fluxo Principal:

1. Professor acessa site do CodeMaster.
2. Professor clica no menu superior no botão “professores”.
3. Professor clica na opção “Ainda não sou cadastrado” abaixo dos campos de login e senha.
4. Professor informa, nome completo, e-mail, instituição de ensino, cidade, estado, país, senha, confirmação de senha e marca o *checkbox* “Li e aceito os termos de contrato”.
5. Professor clica no botão “cadastrar”.
6. Professor é redirecionado para a página principal dos professores.

Fluxo de exceção:

4a. Professor não preenche um dos campos do formulário de cadastro e o sistema apresenta a mensagem “Todos os campos devem ser preenchidos”.

4b. Professor não marca o *checkbox* “Li e aceito os termos de contrato” e o sistema apresenta a mensagem “Você deve ler e aceitar os termos de contrato”.

5b. E-mail informado já existe cadastrado, sistema apresenta a mensagem “Já existe uma conta com este e-mail, verifique o e-mail informado!”.

Caso de Uso 3 – USC03 Login de professor

Atores:

- A02 – Professor

Pré-condição:

- USC02

Fluxo Principal:

1. Professor acessa o site do CodeMaster.
2. Professor clica no botão “professores” no menu superior da tela.
3. Professor informa seu e-mail e senha nos campos indicados.
4. Professor clica no botão “login” e seu e-mail e senha estão corretos.

5. O sistema autentica o professor no sistema.
6. Professor é redirecionado para a página principal dos professores.

Fluxo de exceção:

4a. O e-mail e senha do professor estão incorretos. O sistema informa uma mensagem de erro: “Usuário e senha estão incorretos”.

Caso de Uso 4 – USC04 Analisar múltiplos projetos

Atores:

- A02 – Professor

Pré-condição:

- USC03

Fluxo Principal:

1. Professor seleciona o tipo de projeto (App inventor ou Snap!) e nome da turma.
2. Professor seleciona os conceitos que serão avaliados.
3. Professor escolhe múltiplos arquivos de seu computador que correspondem a uma turma.
4. Professor clica no botão “avaliar”.
5. Professor recebe o resultado da análise de todos os projetos em uma tabela.

Fluxo de exceção:

3a. Professor clica em avaliar sem escolher projetos. Sistema informa uma mensagem de erro: “Nenhum projeto foi selecionado para análise!”.

Caso de Uso 5 – USC05 Ver projetos analisados

Atores:

- A02 - Professor

Pré-condição:

- USC03

Fluxo Principal:

1. Professor clica em ver projetos avaliados no menu lateral da área de professores.
2. Professor seleciona a turma que deseja ver as análises.
3. Sistema consulta e Banco de Dados e retorna o resultado das análises.
4. Professor visualiza as análises de todos os projetos daquela turma em forma de tabela.

Fluxo de exceção:

FE1. Nenhum projeto foi enviado para análise anteriormente pelo professor. Sistema apresenta mensagem: “Nenhum resultado foi encontrado”.

Caso de Uso 6 – USC06 Ver estatísticas das análises

Atores:

- A03 - Pesquisador

Fluxo Principal:

1. Pesquisador acessa o site do CodeMaster.
2. Pesquisador clica no menu superior em estatísticas.
3. Sistema faz consulta e Banco de Dados e retorna os dados estatísticos.

4. Sistema apresenta em uma tabela as médias de cada conceito de todas as avaliações já realizadas no sistema, quantos professores estão cadastrados no sistema e quantos projetos já foram avaliados no sistema.

Fluxo de exceção:

FE1. Nenhum projeto foi analisado anteriormente. Sistema apresenta mensagem: “Nenhuma análise encontrada em nossa base de dados”.

O último caso de uso (USC06 Ver estatísticas das análises) não está no escopo do presente trabalho, sendo uma sugestão para trabalhos futuros.

4.5 PROTOTIPAÇÃO DE TELAS

Nesta seção são apresentados os protótipos de telas da ferramenta proposta pelo presente trabalho. Heliziane Barbosa e Luiz Felipe Azevedo, bolsistas da CnE, também participaram do design das telas, logotipos e imagens dos ninjas utilizados na gamificação dos projetos.

A tela inicial (Figura 24) dá as boas-vindas ao usuário e disponibiliza *links* para páginas específicas para alunos, professores e pesquisadores.

Figura 24 – Protótipo da página inicial

A tela “aluno” (Figura 25) oferece ao usuário não cadastrado a possibilidade de enviar um único projeto para análise.

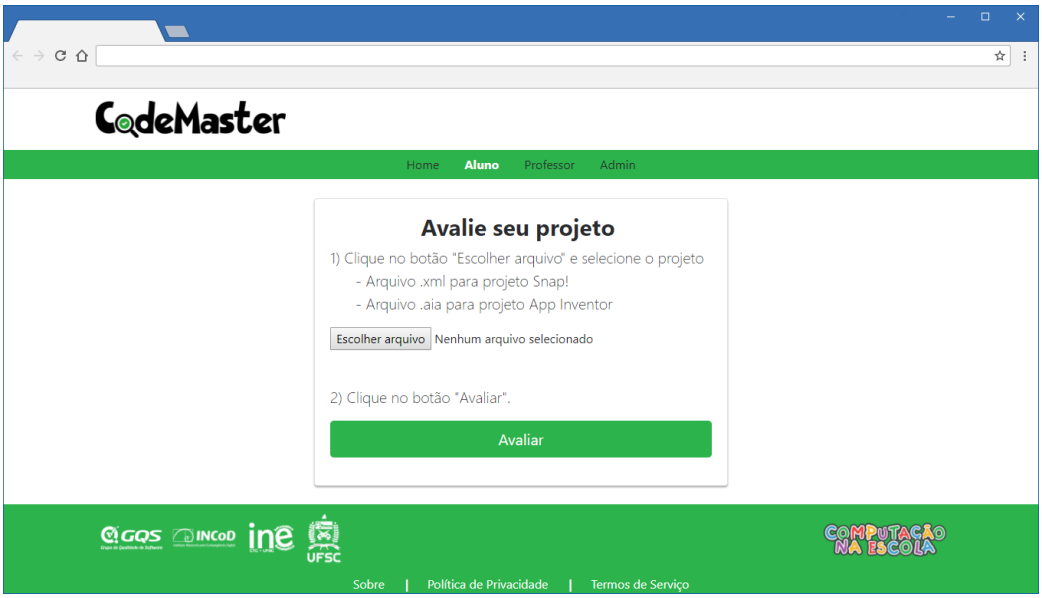


Figura 25 - Protótipo da página “aluno”

A tela que apresenta o resultado da avaliação para o aluno (Figura 26) quando um único projeto é enviado para avaliação, destaca a nota, pontuação total, as pontuações individuais de cada conceito e o nível gamificado do projeto avaliado. A tela também disponibiliza um *link* para um guia que apresenta para o usuário o que é necessário para melhorar sua avaliação (rubrica).

Figura 26 – Protótipo da tela "resultado para aluno"

A tela de *login* para professores (Figura 27) destaca os campos necessários para a autenticação. A tela também disponibiliza *links* para realizar cadastro de professor e recuperar senha de professor.

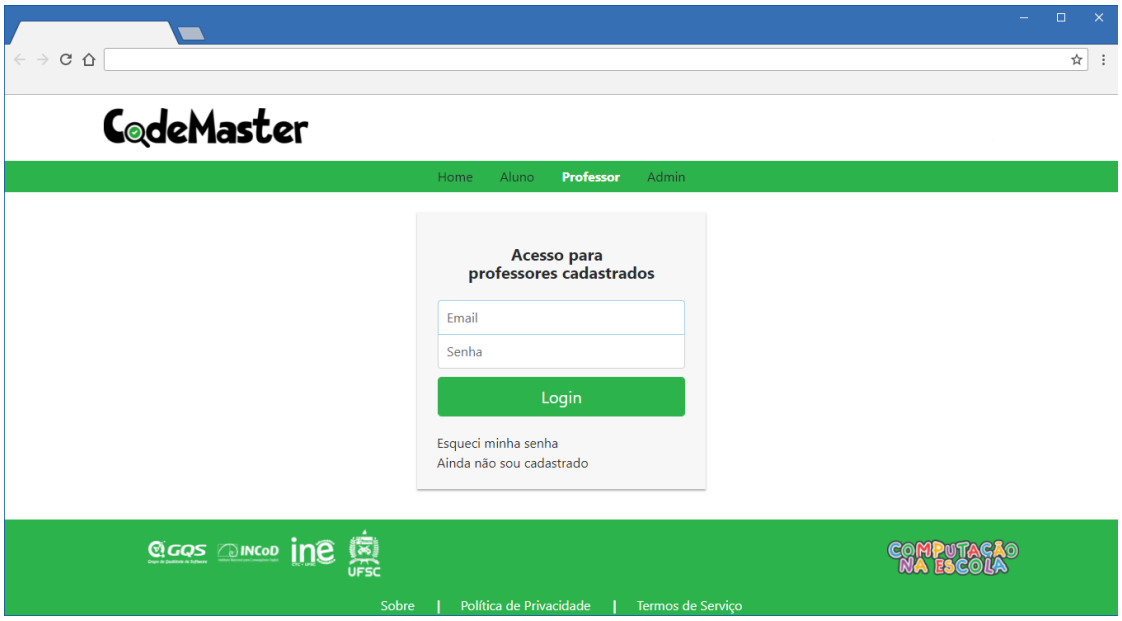


Figura 27 - Protótipo da tela "login para professores"

A página principal do professor (Figura 28) destaca a funcionalidade realizar upload de conjunto de projetos Snap!, para realizar a avaliação de múltiplos projetos. A tela também disponibiliza as funcionalidades realizar *logout* e exibir todas as avaliações do professor (*dropdown* no menu "Professor").

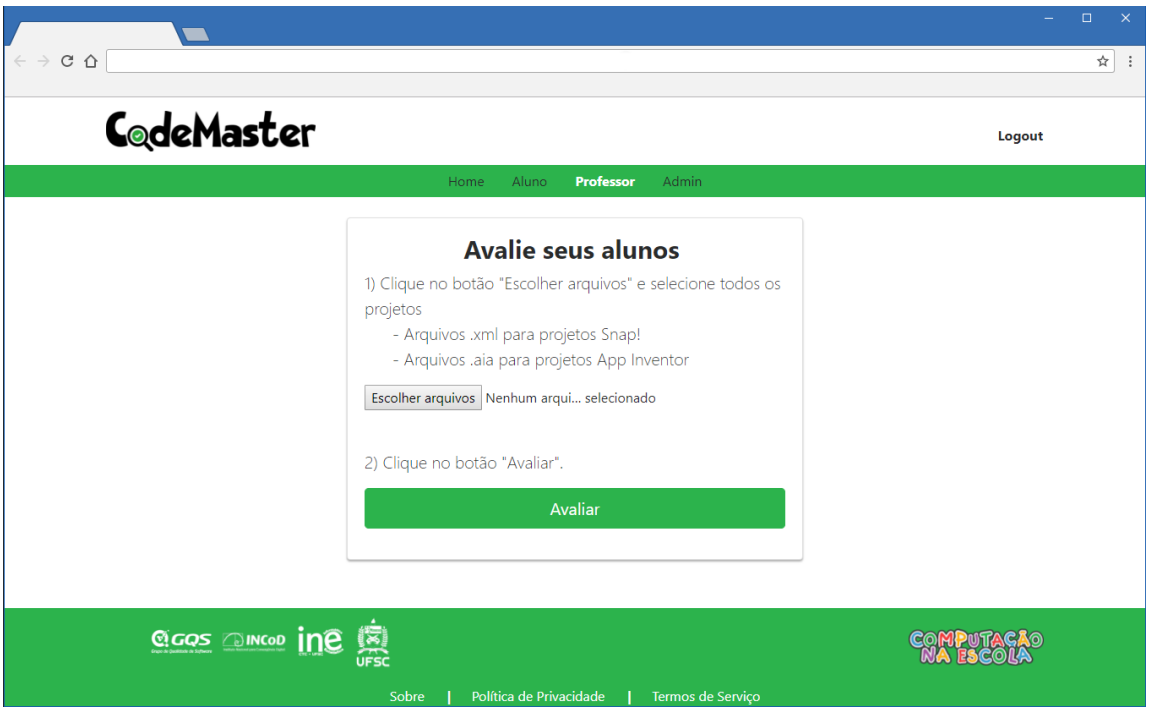


Figura 28 - Protótipo da tela "página principal para professores"

Para realizar *login* e acessar a página principal para professores o professor deve ser cadastrado. A Figura 29 apresenta o protótipo da tela para cadastrar professores.

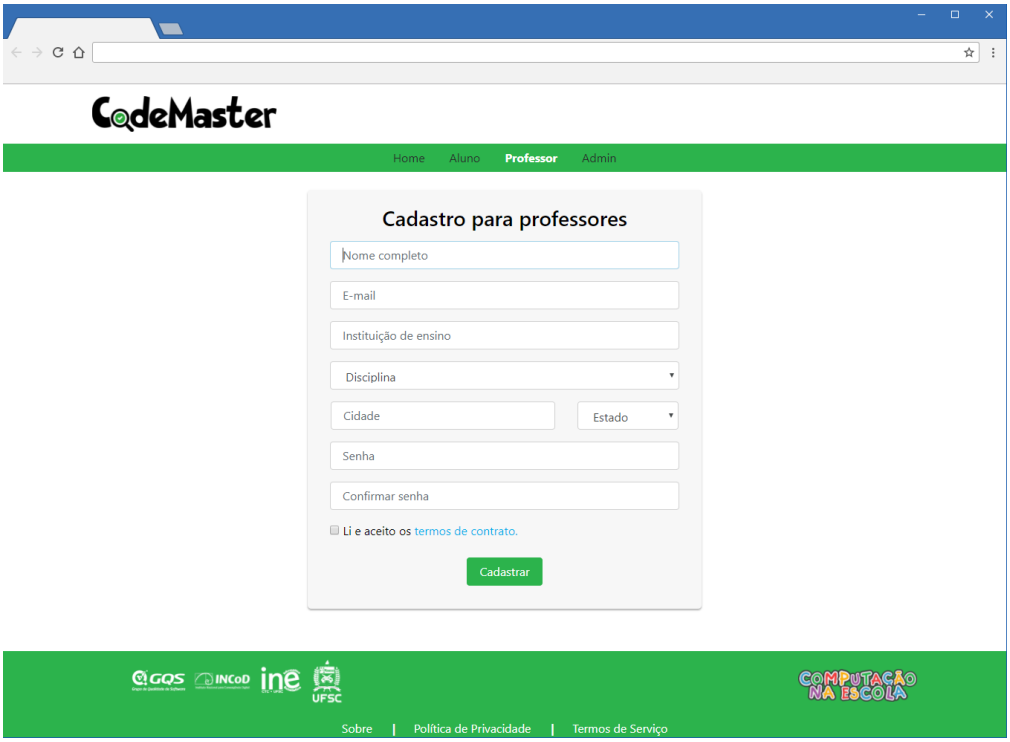


Figura 29 – Protótipo da tela "Cadastrar professor"

A tela que apresenta o resultado das avaliações de múltiplos projetos para o professor (Figura 30) apresenta em uma tabela a nota, pontuação total, pontuação individual de cada conceito e nível gamificado de todos os projetos enviados para avaliação. As médias das notas e pontuações são apresentadas na última linha da tabela.

Projeto	Lógica	Paralelismo	Interatividade com o usuário	Representação de dados	Controle de fluxo	Sincronização	Abstração	Pontuação Total	Nota	Nível
Ana.xml	1	1	1	1	1	1	1	7	3,5	faixa branca
José.xml	2	2	2	2	2	2	2	14	7	faixa verde
Luiz.xml	3	3	3	3	3	3	3	21	10	faixa preta
Média	**2,0**	**2,0**	**2,0**	**2,0**	**2,0**	**2,0**	**2,0**	**14,0**	**6,83**	

Figura 30 - Protótipo da tela "resultado para professor"

4.6 MODELAGEM E IMPLEMENTAÇÃO

A ferramenta CodeMaster foi desenvolvida usando as linguagens de programação Java com JSP (JavaServer Pages) e JavaScript. Essas tecnologias foram escolhidas pelo fato do aluno e equipe de desenvolvimento ter conhecimento prévio delas e por serem as tecnologias disponíveis na infraestrutura de servidor utilizado. Outra vantagem das linguagens de programação escolhidas é a sua popularidade, que facilita encontrar profissionais capacitados para realizar a manutenção da ferramenta durante seu ciclo de vida.

O modelo de arquitetura do sistema foi definido objetivando a separação das camadas de apresentação e analise em diferentes módulos, tornando a aplicação escalável a longo prazo e permitindo a conexão direta de outras aplicações ao serviço de análise de código no futuro. A Figura 31 apresenta um diagrama de componentes do CodeMaster. O sistema é dividido em 3 blocos, um container web com módulo de apresentação e acesso a banco de dados, um container web exclusivo para o módulo de análise e avaliação de código, e o próprio navegador de internet do usuário onde a interface gráfica do sistema é exibida.

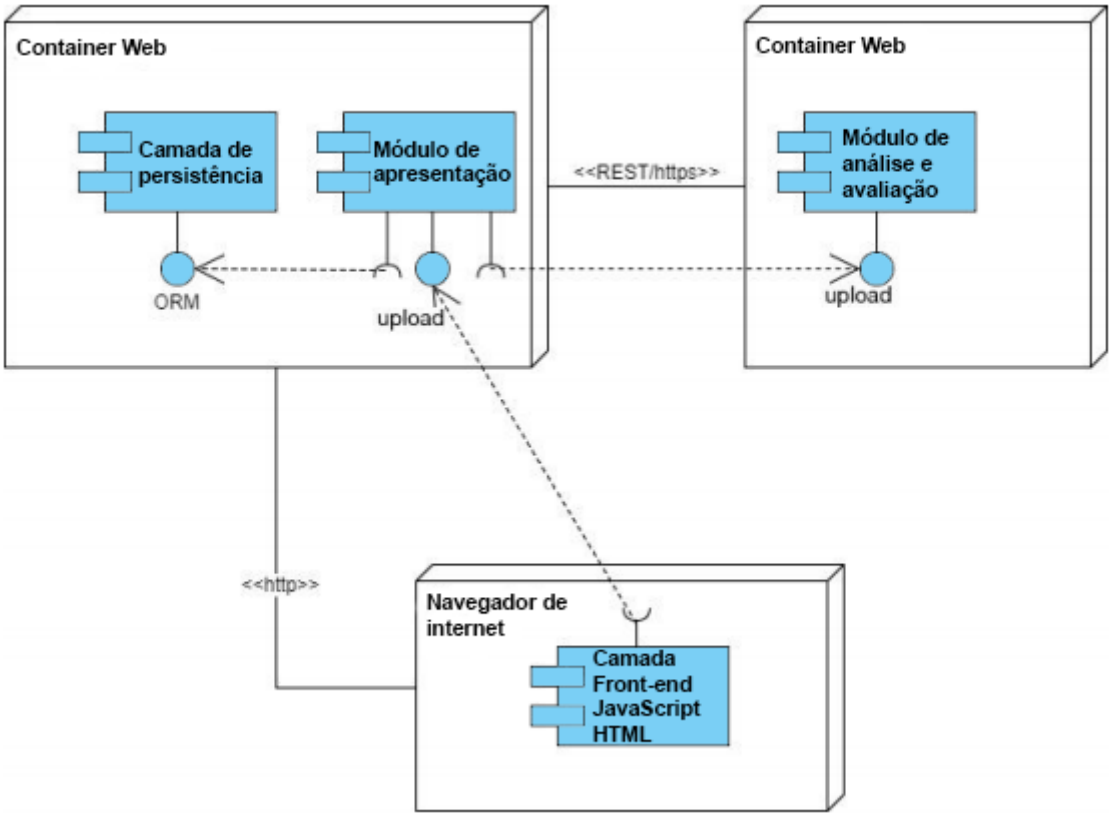


Figura 31 - Diagrama de componentes

O módulo de avaliação é um serviço web *Representational State Transfer* (REST) responsável no sistema por receber o projeto e as configurações de avaliação e retornar o resultado da avaliação, de acordo com o modelo de avaliação definido na seção 4.2. REST é uma arquitetura que permite a separação de sistemas web em módulos (serviços). Para implementar esse serviço o CodeMaster utiliza o *framework* Jersey (JERSEY, 2017) que usa a API JAX-RS (JAX-RS API, 2017) abstraindo os detalhes de baixo nível da implementação de comunicação entre os servidores e simplifica a implementação de um serviço REST. A escolha desse *framework* foi feita pela simplicidade de uso, por ser adequado ao que é proposto no CodeMaster e pela grande quantidade de material disponível, auxiliando na implementação do sistema.

O módulo de apresentação é o responsável pelo controle da interface de usuário, o registro de professores e turmas, a submissão de projetos e a apresentação de resultados tanto para professores quando alunos. O componente de *front-end* utiliza as tecnologias JSP, JavaScript, HTML5 e CSS3 comuns no desenvolvimento de páginas web.

4.6.1 ARQUITETURA DO MÓDULO DE AVALIAÇÃO

O módulo de avaliação foi projetado para que seja expansível no sentido de poder receber mais analisadores de código de outros sistemas (p. ex. Scratch) no futuro e também no sentido de expandir as análises dos analisadores presentes, permitindo que novos conceitos do pensamento computacional sejam adicionados ou, os já existentes, alterados de forma simples e baixo esforço.

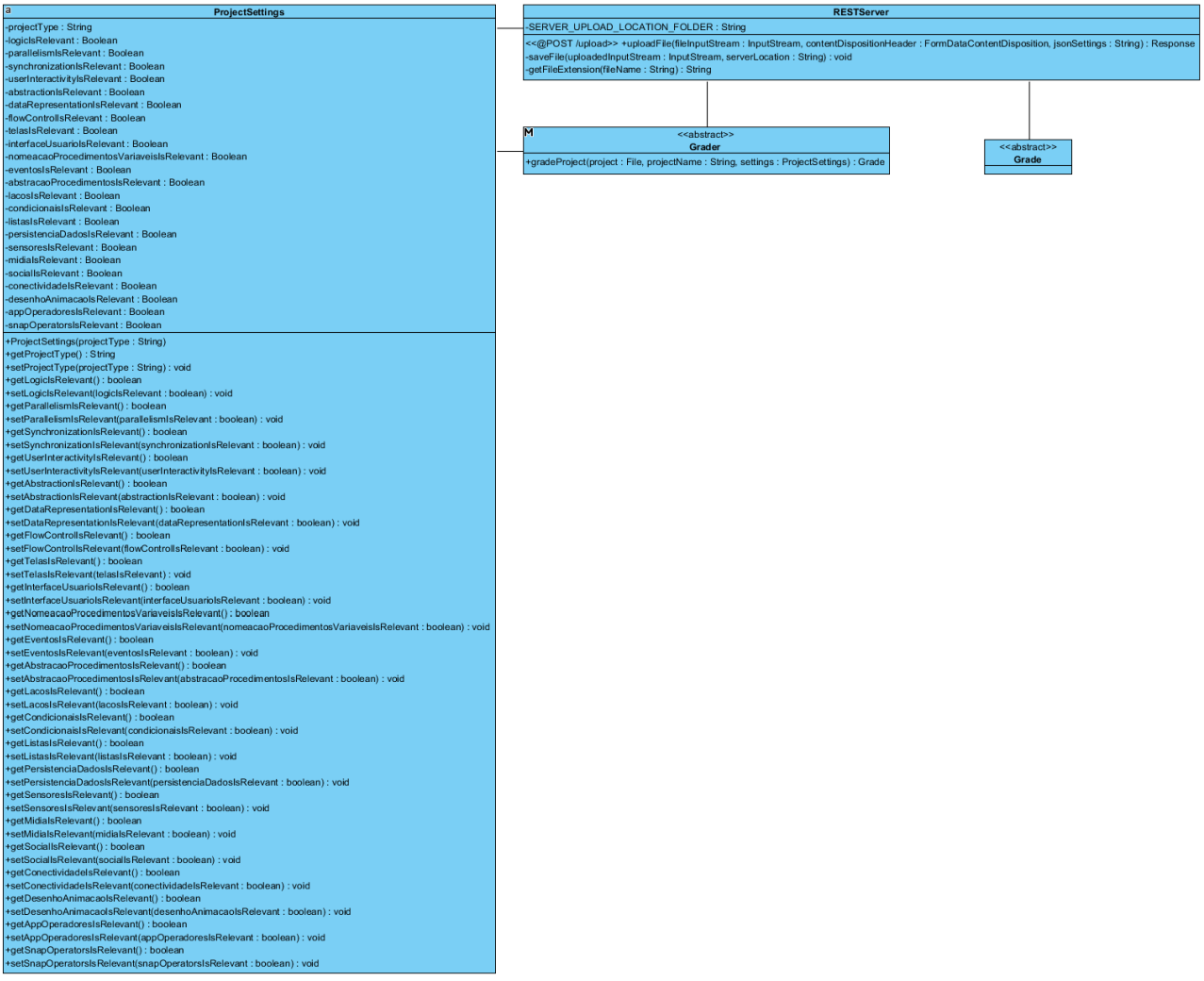


Figura 32 - Diagrama de Classes do serviço REST

A Figura 32 apresenta as classes principais e de controle do módulo de avaliação do CodeMaster. O módulo recebe o projeto e o arquivo de configuração e deve coordenar a execução da análise. Foi definida uma classe abstrata *Grader*, e qualquer instância de analisador que estiver disponível no sistema deve expandir essa classe e implementar o método *gradeProject* que recebe o projeto e a configuração. Da mesma forma, deve retornar ao coordenador do serviço uma instância da classe *Grade* com a avaliação completa do projeto recebido.

A classe *ProjectSettings* é a classe de configuração do analisador, ela determina quais conceitos do pensamento computacional devem ser analisados, evitando o processamento desnecessário de informação por parte do analisador e permitindo ao usuário da ferramenta determinar quais conceitos são relevantes para a avaliação de uma determinada turma.

A Figura 33 apresenta o diagrama de classes do analisador de código proposto pelo presente trabalho, implementado no sistema CodeMaster. A classe *SnapMaster* estende a classe abstrata *Grader* e é a classe principal da implementação.

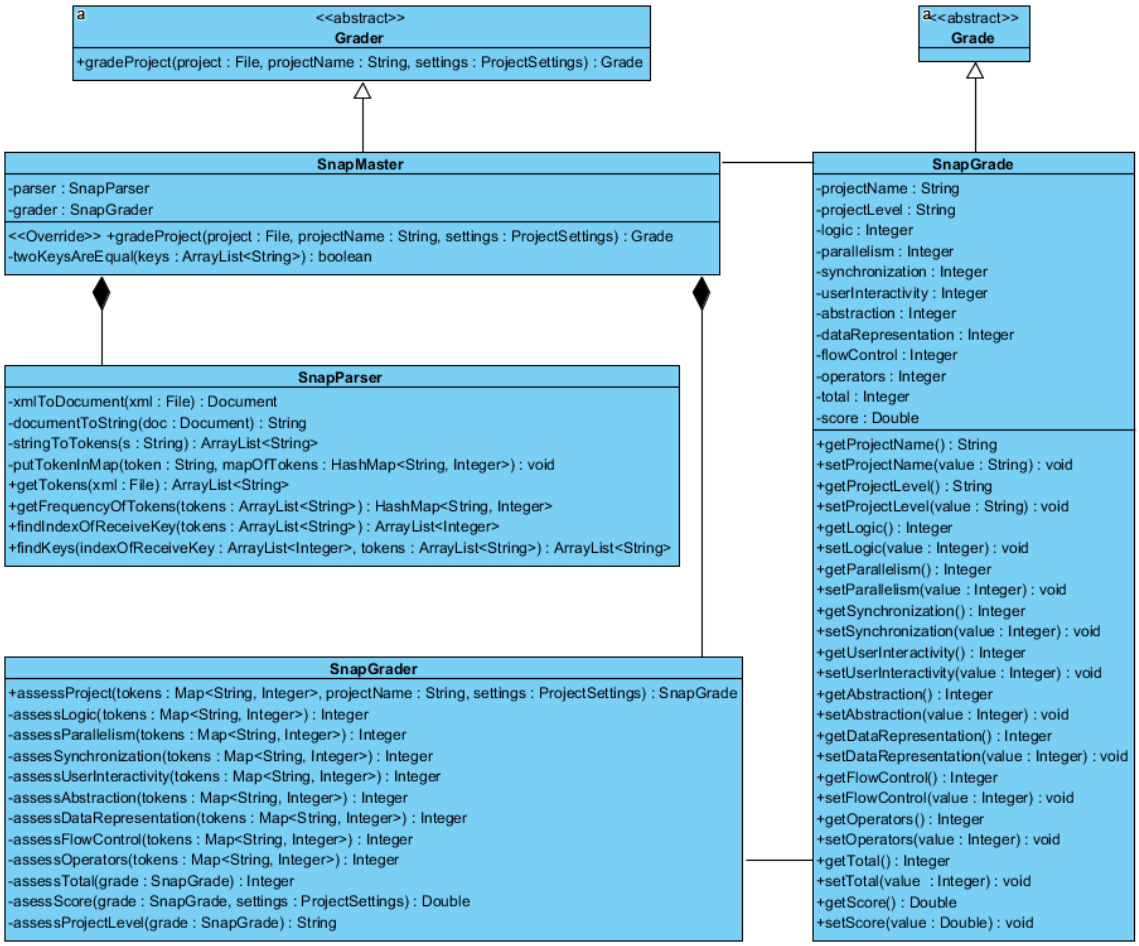


Figura 33 - Diagrama de Classes do serviço REST para Snap!

A classe *SnapMaster* possui dois atributos, *parser* (classe *SnapParser*) e *grader* (classe *SnapGrader*). Primeiramente a classe *SnapMaster* recebe o arquivo XML do projeto Snap! e um *ProjectSettings*. O arquivo é repassado para o *parser*, responsável por todo o processo de análise, gerando uma tabela de *tokens* e suas respectivas frequências de uso. Após o processo de análise, a *SnapMaster* passa a tabela de *tokens* e frequências junto com o *ProjectSettings* para o *grader*, responsável por avaliar o projeto e retornar uma avaliação (objeto da classe *SnapGrade)*.

A Figura 34 apresenta o diagrama de sequência detalhando o método gradeProject implementado pela classe SnapMaster.

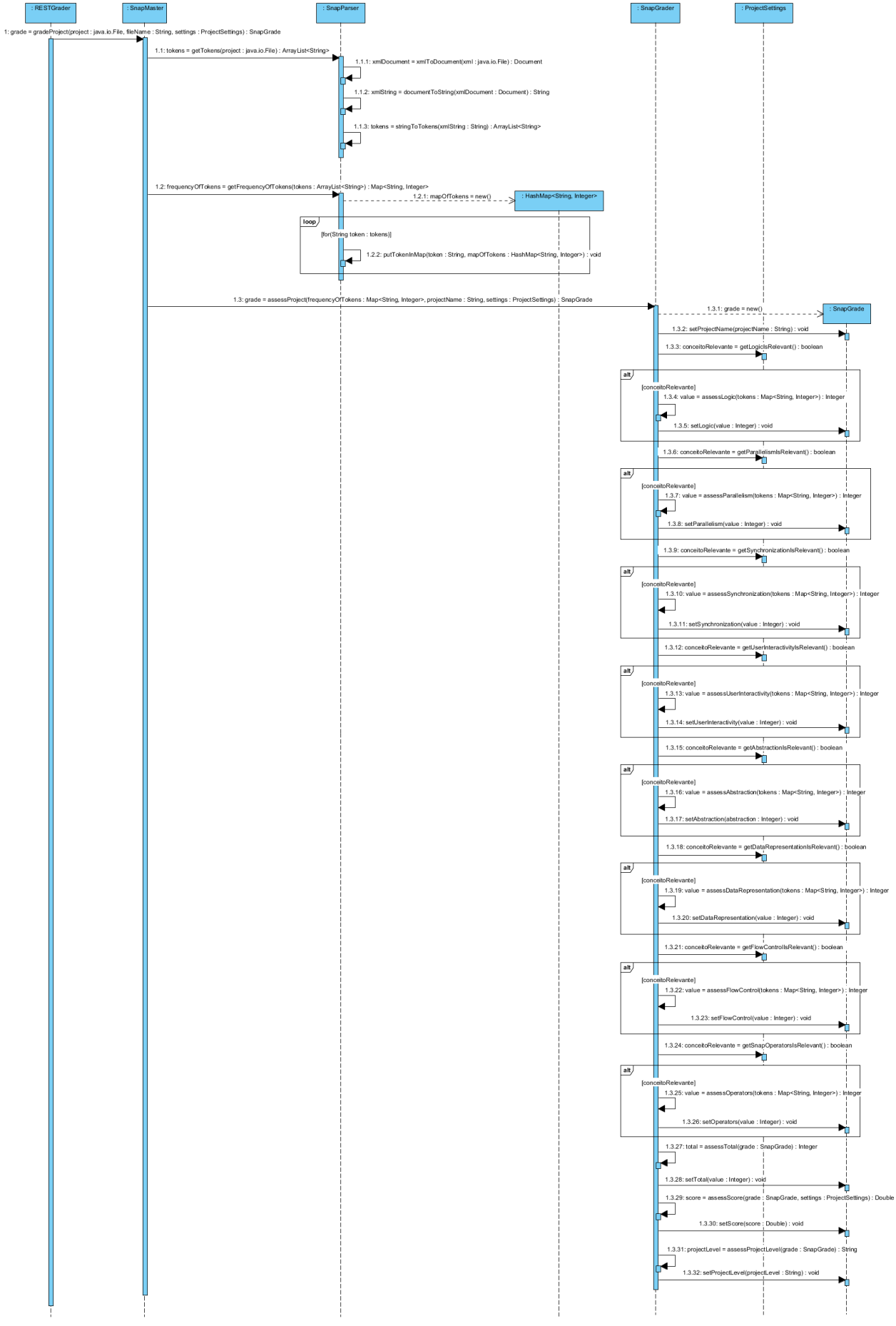


Figura 34 - Diagrama de Sequência do método gradeProject

4.6.2 ARQUITETURA DO MÓDULO DE APRESENTAÇÃO

O módulo de apresentação é responsável por receber o arquivo do projeto do usuário por meio da interface web, enviar o projeto ao módulo de avaliação, receber do módulo de avaliação o resultado da avaliação e apresentar o resultado da avaliação ao usuário. Além disso, este módulo é responsável pelo cadastro de professores, a autenticação do professor perante o sistema, envio de e-mail para confirmação de cadastro, gerenciar o idioma e gerenciar todo o armazenamento de projetos, turmas, professores e administradores no banco de dados.

Para possibilitar a implementação deste módulo em Java foi utilizada a tecnologia JSP para a criação das páginas web acessadas pelo usuário. As páginas JSP se conectam a servlets Java que recebem as requisições vindas do usuário. Por meio das requisições aos servlets os fluxos de execuções acontecem. Cada caso de uso gera fluxos de execuções diferentes.

Todos os fluxos que dependem da análise dos projetos invocam a classe HTTPClient que se comunica com o módulo de avaliação, fazendo o envio do arquivo de projeto e recebendo a resposta do módulo de avaliação. Os fluxos que dependem do banco de dados, tanto para salvar informações ou para consultas, chamam métodos da classe Control, que acessam o banco de dados através dos respectivos objetos de acesso aos dados (DAOs). Os fluxos que dependem de envio de e-mail chamam o método da classe Mail que faz o envio de e-mail simples.

No diagrama de classes apresentado na Figura 35 é possível observar as classes Upload e UploadAluno, que tratam as requisições de avaliação de projetos da área de professor e da área de aluno respectivamente. Neste diagrama é apresentado apenas as classes principais que interagem com as classes de *upload*, como a classe HTTPClient responsável pela comunicação com o módulo de avaliação e a classe Control responsável por salvar os projetos avaliados no banco de dados.

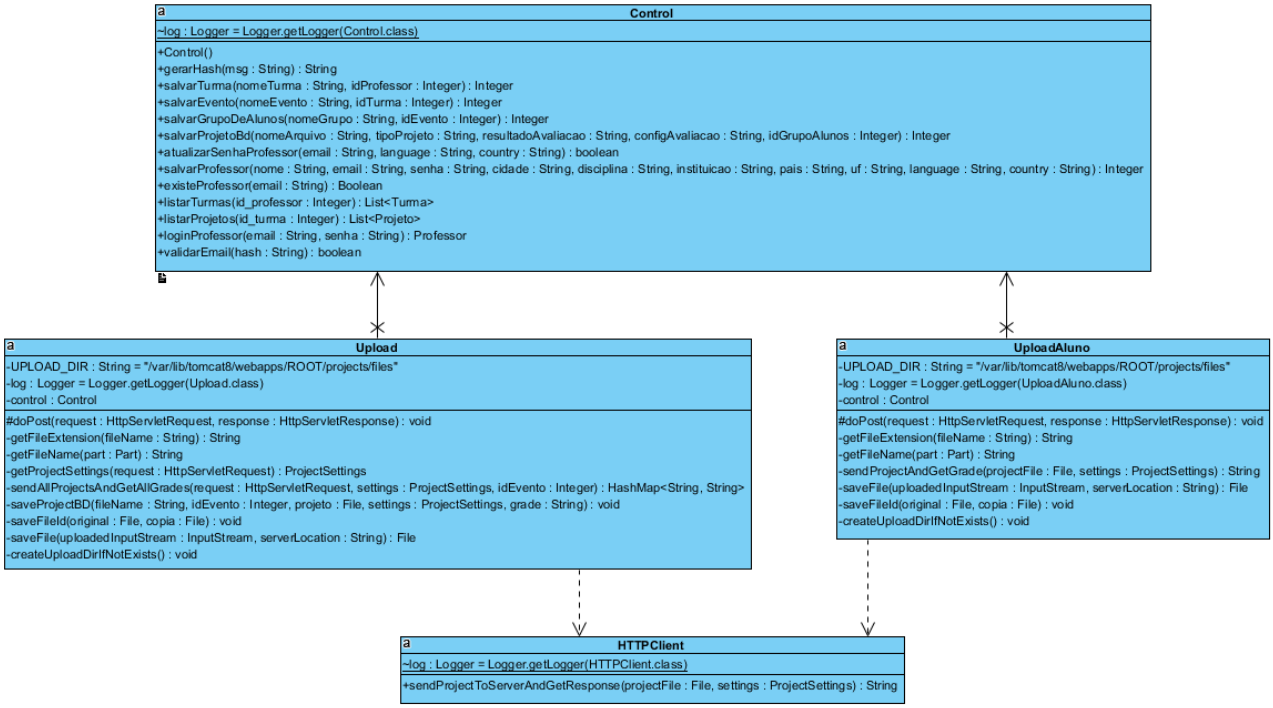


Figura 35 - Diagrama de Classes simplificado do módulo de apresentação

O diagrama apresentado na Figura 36 ilustra as classes responsáveis por armazenar as informações no banco de dados durante a execução do caso de uso “USC02 - Cadastrar professor”. A classe CadastroProfessor recebe a requisição de um novo cadastro, para isso ela chama métodos responsáveis da classe Control que armazenam os dados por meio do objeto de acesso aos dados do professor (ProfessorDAO), após o cadastro concluído um e-mail com o link de confirmação é enviado ao endereço de e-mail do professor cadastrado.

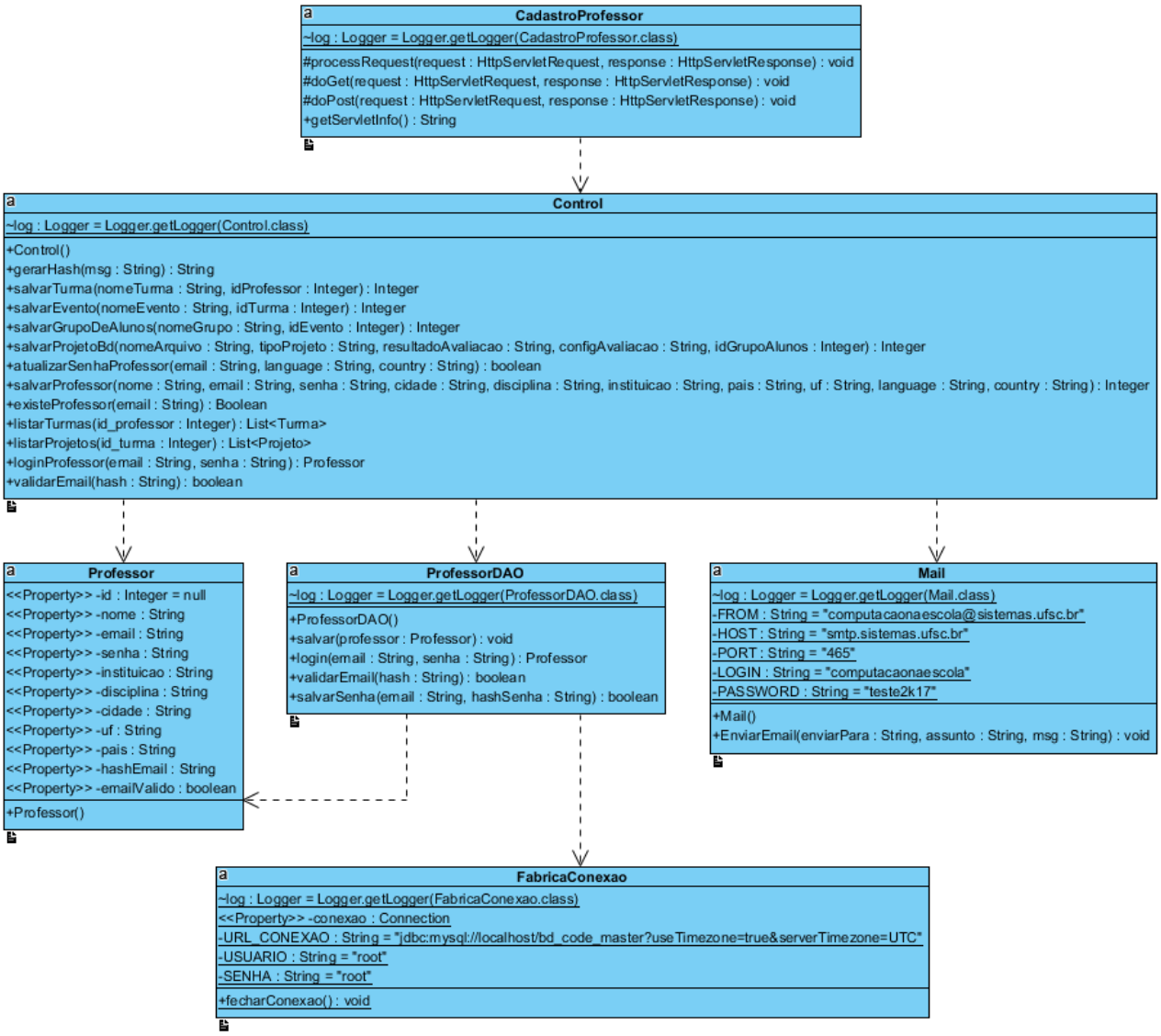


Figura 36 - Diagrama de Classes do caso de uso "USC02 - Cadastrar professor"

A Figura 37 apresenta o diagrama de sequência que modela as mensagens trocadas pelo módulo de apresentação para a avaliação de múltiplos projetos, enviados por um professor. O fluxo de execução para análise de um único projeto enviado por um aluno é análogo, alterando apenas a presença de um laço.

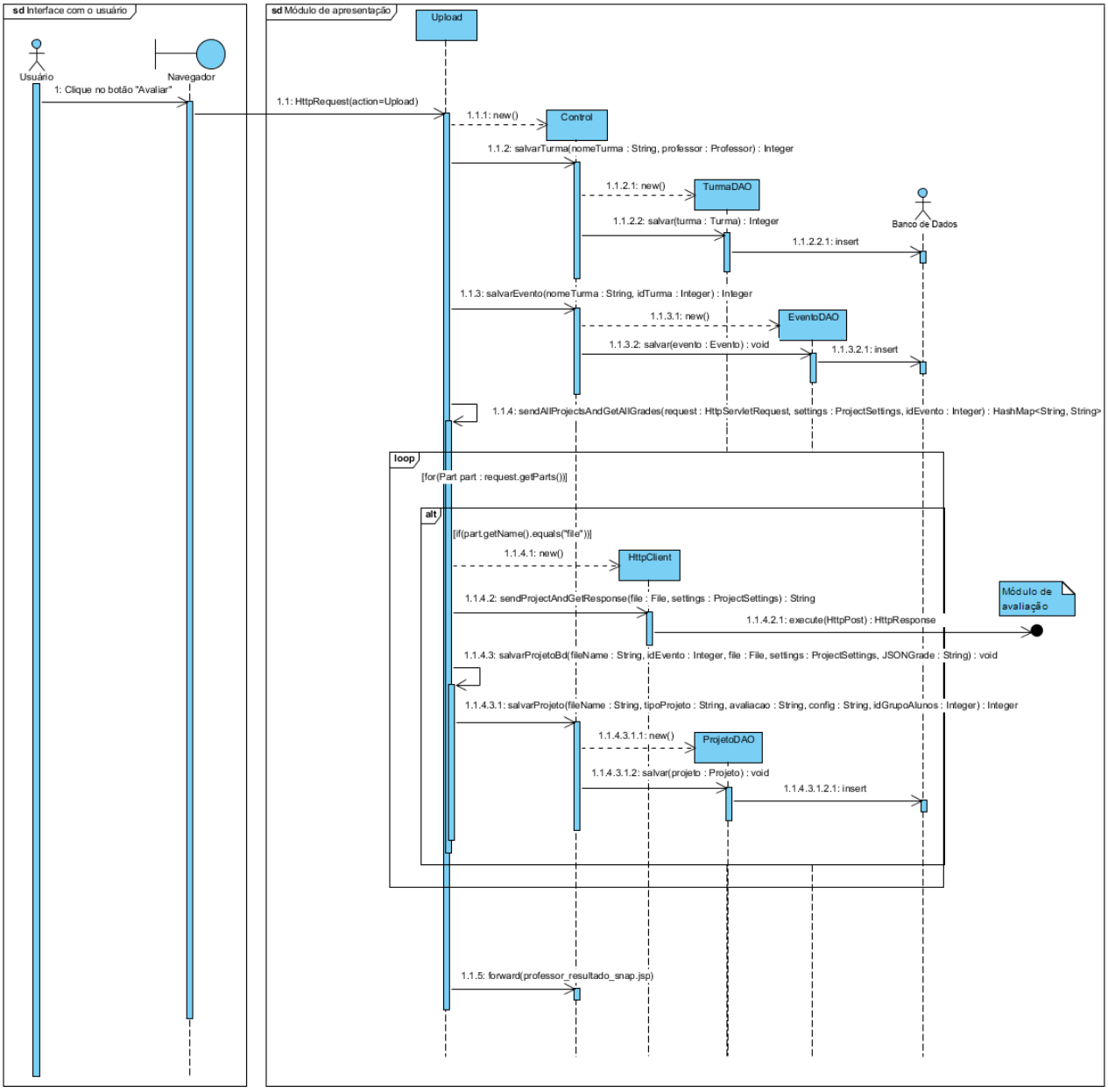


Figura 37 - Diagrama de Sequência modelo do upload de múltiplos projetos

A camada de persistência é utilizada apenas pelo módulo de apresentação. Esta camada utiliza um banco de dados MySQL (MYSQL, 2017). O MySQL permite a criação de bancos de dados relacionais. O banco de dados do CodeMaster foi projetado de forma que sua estrutura ficasse simples e objetiva, porém, que permita futuras implementações no

sistema sem que os dados já armazenados se tornem inconsistentes. A Figura 38 apresenta as tabelas presentes no banco de dados do CodeMaster bem como as suas relações.

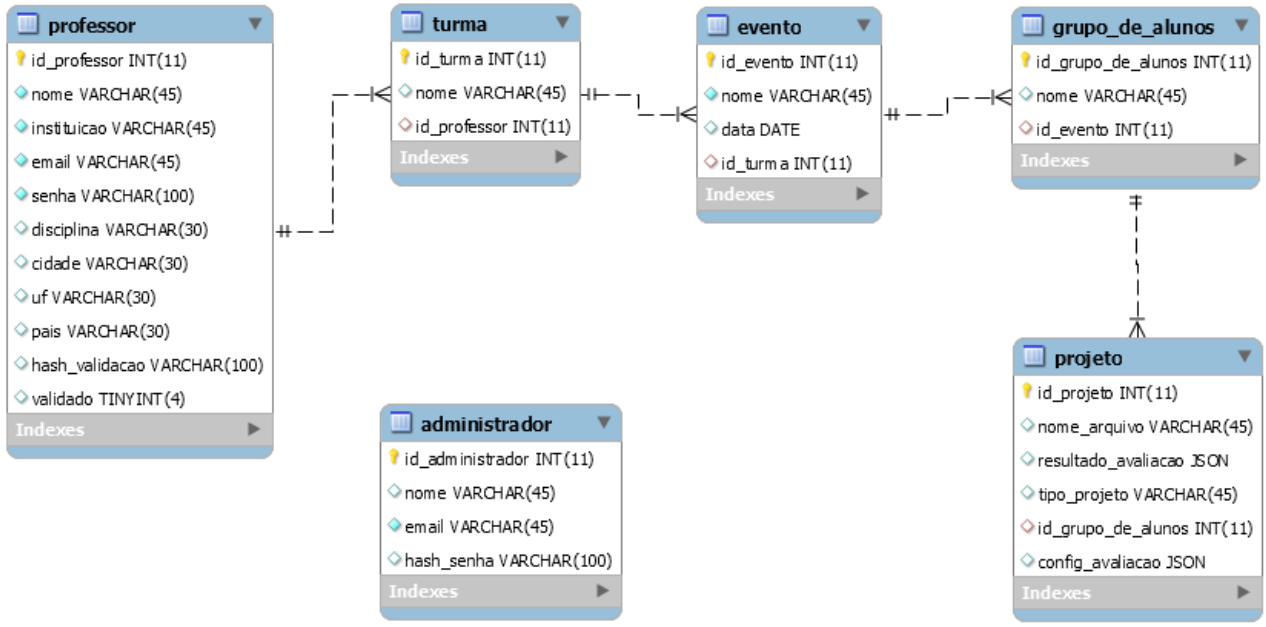


Figura 38 - Diagrama Entidade Relacionamento do Banco de Dados

Os dados de cadastro de professores são armazenados na tabela professor, bem como *login* e senha para acesso à área do professor do CodeMaster. Para uma maior segurança, a senha do professor cadastrado não é salva em formato original, mas em sim um *hash* da senha, obtido por meio do algoritmo SHA-256, desta forma é possível a partir da senha obter o *hash*, mas a partir do *hash* é impossível obter a senha. Isso dificulta a descoberta da senha dos usuários por pessoas mal-intencionadas que possam vir a acessar o banco de dados. Além dos dados comuns cadastrados como nome, cidade, estado, disciplina, dois campos são utilizados para a validação do e-mail do professor. O hash_validacao armazena o *hash* do e-mail cadastrado, esse *hash* é enviado por e-mail para a validação da conta do professor e o campo validado que armazena uma variável do tipo booleana informa se o professor validou o seu cadastro por meio do link de confirmação recebido por e-mail.

A tabela turma armazena um ID da turma criada, o nome que o professor deu para a turma e uma chave estrangeira que relaciona a tabela professor e informa qual é o professor de determinada turma.

É possível que futuramente se deseje acrescentar a informação de eventos que determinada turma participa ao CodeMaster, cada evento será armazenado na tabela evento que contém um nome e uma data do respectivo evento.

A tabela grupo_de_alunos armazena a informação do nome do grupo de alunos ou o nome de um aluno em especifico que criou determinado projeto. O nome do grupo de alunos ou aluno é determinado por meio do nome do arquivo de projeto enviado ao sistema. Portanto, no momento do envio do projeto para avaliação ao CodeMaster é importante que o nome do projeto contenha esta informação (p. ex.: MariaSilva.xml). Essa informação facilita a diferenciação dos alunos na exibição do resultado da análise e avaliação de projetos de uma turma.

A tabela projeto armazena as informações de projetos avaliados. Ela contém um ID de projeto que é utilizado como novo nome de arquivo para armazena-lo no servidor, o nome original do projeto avaliado, o resultado da avaliação em JSON salvo a partir da classe Grade, a configuração da análise em JSON salva a partir da classe ProjectSettings, o tipo de projeto (App Inventor ou Snap!) e uma chave estrangeira relacionando o projeto a um grupo de alunos. Se o projeto for submetido à avaliação do CodeMaster por um aluno, de forma anônima, o campo que relaciona o projeto a um grupo de alunos ou aluno é nulo.

A tabela administrador ainda não é utilizada pelo CodeMaster, ela existe para possibilitar o cadastro de possíveis administradores de sistema e pesquisadores que poderão ter acesso a páginas de estatísticas e informações especificas do sistema que o usuário comum não deverá possuir.

4.7 EXPANDINDO O CODEMASTER

O CodeMaster foi projetado para que as expansões sejam fáceis de serem implementadas. Nesta seção é apresentado como adicionar um novo critério de avaliação ao analisador de código para Snap! e como adicionar um novo analisador de código ao CodeMaster.

4.7.1 ADICIONANDO NOVO CRITÉRIO DE AVALIAÇÃO

Adicionar um novo critério de avaliação diz respeito a expandir as avaliações do analisador de código para Snap! atualmente disponível. Futuramente, talvez se queira adicionar mais um critério de avaliação à rubrica, como por exemplo, avaliar “jogabilidade”.

Para isso não é necessário criar um novo analisador de código, fazendo algumas alterações consegue-se expandir o atual analisador de código para avaliar o novo critério.

No módulo de apresentação deve ser adicionado mais um *checkbox* a tela de professor que contém os critérios que se deseja analisar. Nas telas de resultado de aluno e professor o novo critério deve ser adicionado à tabela. Outra alteração neste módulo é na classe ProjectSettings que deve receber uma nova variável booleana, informando se o novo critério deve ser avaliado. A classe SnapGrade deve receber uma nova variável que receberá a pontuação do novo critério de avaliação.

No módulo de avaliação, as classes ProjectSettings e SnapGrade devem receber as mesmas alterações que as respectivas classes do módulo de apresentação. Nenhuma alteração é necessária na classe SnapParser, caso o novo critério seja avaliado verificando a frequência em que algum *token* aparece no código. Na classe SnapGrader, que contém um método para a avaliação de cada conceito, um novo método deve ser implementado. Esse método será responsável por avaliar os *tokens* do código e retornar uma nota, de acordo com a rubrica definida. O novo método deve ser chamado, de forma condicional, de acordo com a nova variável em ProjectSettings, dentro do método “assessProject”. Por fim, o método “assessTotal” deve usar a pontuação parcial do novo critério na soma da pontuação total e o método “assessScore” deve adicionar o novo critério ao cálculo da nota.

4.7.2 ADICIONANDO NOVO ANALISADOR DE CÓDIGO

Adicionar um novo analisador de código diz respeito a incluir no CodeMaster a análise e avaliação de projetos de uma nova linguagem de programação. O núcleo do novo analisador deve ser implementado, isso é composto por reconhecimento de projeto, extração de código, análise e estatística dos componentes e avaliação de critérios. Partindo do pressuposto que o núcleo já está implementado, inclui-lo ao CodeMaster é apresentado abaixo.

Primeiramente para adicionar um novo analisador ao sistema, deve-se criar no módulo de apresentação: telas de resultado para o novo analisador, classe do novo analisador, que estende a classe Grade, e que receberá o resultado da avaliação, além de adicionar a nova opção de linguagem na tela de avaliação do professor e novos critérios se os disponíveis não forem suficientes na classe ProjectSettings. Na classe Upload deve ser liberado o upload de arquivos na extensão de projeto do novo analisador.

No módulo de avaliação toda a implementação do novo analisador deve ser inserida, basta que a classe principal do novo analisador estenda a classe Grader e que o método gradeProject seja implementado. Esse método recebe o ProjectSettings com os critérios que devem ser analisados, o nome do projeto e o arquivo de projeto em si. Este método retorna uma instância de Grade que contém as pontuações de cada critério analisado no novo analisador, a pontuação total, a nota de 0 a 10 e o nível ninja. A classe RESTGrader é a responsável por direcionar o fluxo de execução para o analisador correspondente do projeto recebido, portanto, nela deve-se permitir o novo tipo de projeto que virá especificado no ProjectSettings adicionando uma nova condicional de tipo de projeto e instanciando o novo analisador.

4.8 TESTES

Os primeiros testes realizados foram nas funções que analisam e avaliam os 8 critérios definidos na rubrica. Para cada critério há uma função correspondente no código. Cada função foi testada inicialmente por projetos Snap! criados de forma que contivessem apenas blocos específicos do critério analisado. Após resultados satisfatórios na avaliação de cada critério separado, foram feitos testes com aplicativos funcionais desenvolvidos pela iniciativa Computação na Escola, como por exemplo o projeto “Coliseu”, já integrando todos os critérios e gerando uma nota final. Após os testes no avaliador de código para Snap!, foram feitos testes na integração dos avaliadores App Inventor e Snap! e serviço REST, o módulo de avaliação propriamente dito. Para isso foi criada uma tela de upload de arquivo temporária neste módulo. Com isso foi possível testar o envio de arquivos de projetos das diferentes linguagens, testar o algoritmo de seleção do avaliador que deve ser alocado pelo sistema e testar o recebimento do resultado das avaliações.

Acompanhando o desenvolvimento incremental do módulo de apresentação, os testes foram sendo feitos. Primeiramente foi desenvolvida a função de envio de projetos únicos, portanto, a função de envio para alunos. Deste modo foi possível testar: o upload de arquivos de projetos no módulo de apresentação, o envio do projeto ao módulo de avaliação, o recebimento do resultado da avaliação e a apresentação do resultado ao usuário. Após os testes com apenas um projeto terem retornado resultados satisfatórios, testes nas funções de upload de múltiplos projetos foram realizados.

Por fim os testes a camada de persistência foram feitos. Foram feitos testes nas funções de cadastro de professores, de *login* de professores e salvamento de turmas, alunos e projetos. Também foram feitos testes na função de envio de e-mail e validação de conta de professores.

4.8.1 CASOS DE TESTE

Com o objetivo de testar todas as funções e exceções previstas no sistema, foram definidos casos de teste. A Tabela 20 apresenta os casos de teste, definidos a partir dos casos de uso da seção 4.4. Também são apresentados os resultados da execução dos casos de teste.

Tabela 20 - Casos de teste

Caso de uso	Procedimento de teste	Resultado esperado	Status
USC01 Analisar projeto (sucesso)	Acessar o CodeMaster na página para aluno, selecionar um arquivo de projeto Snap! e clicar no botão “Avaliar”.	Avaliação do projeto é exibida na tela de resultado para aluno.	OK.
USC01 Analisar projeto (arquivo inválido)	Acessar o CodeMaster na página para aluno, selecionar um arquivo com formato diferente de .xml ou .aia e clicar no botão “Avaliar”.	Redirecionamento para página de erro.	OK.
USC02 Cadastrar professor (sucesso)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, clicar em “Ainda não sou cadastrado”, inserir os dados cadastrais e clicar em “Cadastrar”.	A ferramenta envia um e-mail de validação e o cadastro é efetuado com sucesso.	OK.
USC02 Cadastrar professor (E-mail em uso)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, clicar em “Ainda não sou cadastrado”, inserir os dados cadastrais (e-mail já cadastrado no sistema) e clicar em “Cadastrar”.	A ferramenta mostra a notificação de que o e-mail informado já está cadastrado.	OK.

USC02 Cadastrar professor (senhas diferentes)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, clicar em “Ainda não sou cadastrado”, inserir os dados cadastrais (senha e confirmação de senha são diferentes) e clicar em “Cadastrar”.	A ferramenta mostra a notificação de que as senhas são diferentes.	OK.
USC02 Cadastrar professor (sem marcar o checkbox “Li e aceito os termos de serviço”)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, clicar em “Ainda não sou cadastrado”, inserir os dados cadastrais (sem marcar o checkbox “Li e aceito os termos de serviço”) e clicar em “Cadastrar”.	A ferramenta mostra a notificação de que é necessário marcar o checkbox para continuar o cadastro.	OK.
USC03 Login de professor (sucesso)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, informar usuário e senha válidos e clicar em “Login”	Redirecionamento para a página do professor.	OK.
USC03 Login de professor (usuário ou senha inválidos)	Acessar o CodeMaster na página para professor, ser redirecionado para a tela de login, informar usuário ou senha inválidos e clicar em “Login”	A ferramenta mostra a notificação de que o usuário ou senha são inválidos	OK.
USC04 Analisar múltiplos projetos (sucesso)	Após login de professor com sucesso, na tela do professor, digitar o nome da turma, selecionar tipo de projeto Snap!, clicar no botão “Proximo”, marcar todos os critérios de avaliação, clicar no botão “Proximo”, selecionar múltiplos arquivos de projeto Snap! e clicar no botão “Avaliar”	Avaliações dos projetos são exibidas na tela de resultado para professor.	OK.

USC04 Analisar múltiplos projetos (nome da turma vazio)	Após login de professor com sucesso, na tela do professor, não digitar o nome da turma e clicar no botão “Proximo”	A ferramenta mostra a notificação de que o usuário deve informar um nome para a turma.	OK.
USC04 Analisar múltiplos projetos (arquivos inválidos)	Após login de professor com sucesso, na tela do professor, digitar o nome da turma, selecionar tipo de projeto Snap!, clicar no botão “Proximo”, marcar todos os critérios de avaliação, clicar no botão “Proximo”, selecionar múltiplos arquivos com formato diferente de .xml e clicar no botão “Avaliar”	Redirecionamento para página de erro.	O sistema apresentou avaliações com todas as pontuações com valor 0 na página de resultado para professor
USC05 Ver projetos analisados	Após login de professor com sucesso, clicar na opção de menu “Turmas”, selecionar uma turma e clicar em “Ver turma”	Avaliações dos projetos são exibidas na tela de resultado para professor.	OK.

Entre os 12 testes executados, apenas um apresentou resultado diferente do esperado. O *upload* de múltiplos arquivos inválidos (formato diferente de .xml ou .aia), na tela do professor deveria resultar em erro mas resultou em uma falsa avaliação, que foi apresentada para o usuário e guardada no banco de dados.

Os casos de teste foram executados em novembro de 2017, quando o caso de uso “USC06 – Ver estatísticas das análises” ainda não estava implementado e, portanto, nenhum teste relacionado a este caso de uso foi executado.

4.8.2 TESTES DE DESEMPENHO

O desempenho (tempo de resposta) é um fator importante na utilização de quase todo sistema de software. No caso da ferramenta CodeMaster o desempenho não é fundamental, porém está ligado diretamente a satisfação do usuário. Sabe-se também que o fator determinante para o desempenho, no caso do CodeMaster, é a velocidade e qualidade de conexão de internet do usuário. Como cada projeto tem que ser enviado ao CodeMaster para análise, projetos maiores podem ter um tempo de resposta maior também.

Visando avaliar o desempenho da ferramenta CodeMaster sem influência da conexão, que varia de usuário para usuário, o desempenho foi medido com os três componentes do sistema (navegador, modulo de apresentação e módulo de avaliação) sendo executados na mesma máquina (*localhost)*. A máquina utilizada nos testes foi preparada para não ter nenhuma outra aplicação de usuário sendo executada durante os testes. As especificações da máquina utilizada nos testes de desempenho são apresentadas na Tabela 21.

Tabela 21 - Especificações da máquina

Componente	Nome
Processador	Intel® Core™ i7-4510U
Memória (RAM)	8GB DDR3-1600MHz
Dispositivo de armazenamento	120GB SSD
Sistema Operacional	Ubuntu 16.04
Servidor	Apache Tomcat 8.0.27

Para realizar os testes, primeiramente, dez projetos Snap! foram selecionados. Os dez projetos são selecionados aleatoriamente entre projetos encontrados na única galeria de projetos Snap! encontrada (NATHALIERUN, 2017). Os projetos têm tamanhos diferentes, sendo o menor deles 16 *kilobytes* e o maior deles 1036 *kilobytes*.

O primeiro teste de desempenho realizado foi enviar cada um dos dez projetos individualmente com o perfil de aluno e coletar o tempo de resposta de cada envio. O segundo teste realizado foi enviar os dez projetos de uma vez no perfil de professor e coletar o tempo de resposta de cada envio. A Tabela 22 detalha os resultados obtidos nos testes.

Tabela 22 - Resultados obtidos nos testes de desempenho

Teste	Projeto	Resultado		
		Upload 1	Upload 2	Upload 3
Upload de um projeto Snap! (aluno)	Pgcd	213ms	235ms	226ms
	GeometrieAnalytiqueNathanModifie	152ms	129ms	135ms
	CarresRemplis	136ms	131ms	134ms
	JeuDeBellF	133ms	126ms	135ms
	Dinosaure.A.VitesseConstante	181ms	161ms	184ms
	RVB.Couleur	129ms	124ms	125ms
	CalculsSurLesCoordonnees	172ms	143ms	133ms
	ChasseAuTresor.Simulation.1200x900	140ms	146ms	134ms

	ConvertisseurShadokEntier	133ms	121ms	138ms
	Comprendre.Clones	128ms	158ms	131ms
	Média dos dez projetos:	**152ms**	**147ms**	**147ms**
Upload de múltiplos projetos Snap! (professor)	Pacote com os dez projetos	842ms	756ms	657ms
	Média dos três *uploads*:	**752ms**		

Pode-se observar na Tabela 22 que o tamanho de um projeto Snap! não teve influência considerável no tempo de resposta do sistema, quando os três componentes do sistema estavam sendo executados na mesma máquina. Também pode-se observar na Tabela 22 que o tempo de resposta do sistema para avaliar um único projeto foi, em média, de aproximadamente 150 milissegundos e o tempo de resposta do sistema para avaliar um pacote de dez projetos foi, em média, de aproximadamente 750 milissegundos. Para determinar se esse tempo de resposta é considerado satisfatório, avaliações com usuários podem ser realizadas.

5. AVALIAÇÃO

Para avaliar a qualidade do protótipo do CodeMaster é realizada uma avaliação preliminar. O objetivo é analisar a qualidade da ferramenta CodeMaster. A qualidade é avaliada em termos da corretude da avaliação realizada pela ferramenta e também em termos de utilidade, adequação funcional, eficiência de desempenho e usabilidade do ponto de vista dos professores e estudantes de computação no ensino básico.

5.1 DEFINIÇÃO DA AVALIAÇÃO

Baseados em ISO/IEC-25010 (ISO/IEC-25010, 2011), ISO/IEC-9241 (ISO/IEC-9241, 2010), TAM (DAVIS, 1989), e SUS (BROOKE et al., 1996) os fatores de qualidade a serem avaliados são decompostos (Tabela 23).

Tabela 23 - Fatores de qualidade a serem avaliados

Característica	**Subcaracterística**	**Avaliação do usuário**			
		Item do questionário		**Observação**	**Teste**
		Questionário do professor	**Questionário do aluno**		
Utilidade		Você acha a ferramenta CodeMaster útil no ensino de computação no ensino básico?	Você acha a ferramenta CodeMaster útil na aprendizagem de programação?		

		Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) a ferramenta CodeMaster é prática no seu dia-a-dia?			
Adequação funcional	Completude funcional	Você acha que existem aspectos relevantes no processo de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta CodeMaster?			
		Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?			
		Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?		
	Corretude funcional	Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?	Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?		Testes de corretude comparando os resultados do CodeMaster com os resultados de avaliações manuais.
			Você achou a nota justa?		
Eficiência de desempenho	Comportamento temporal	A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória?	A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória?		
Usabilidade	Eficácia			Usuário completou a tarefa.	
	Satisfação	Eu penso que usarei a ferramenta CodeMaster com frequência.	Eu penso que usarei a ferramenta CodeMaster com frequência.		
		Acho a ferramenta CodeMaster desnecessariamente complexa.	Acho a ferramenta CodeMaster desnecessariamente complexa.		
		Penso que a ferramenta CodeMaster é fácil de usar.	Penso que a ferramenta CodeMaster é fácil de usar.		
		Acho que vou precisar da ajuda de um técnico para usar a ferramenta CodeMaster.	Acho que vou precisar da ajuda de um técnico para usar a ferramenta CodeMaster.		
		Acho as funções da ferramenta CodeMaster bem integradas.	Acho as funções da ferramenta CodeMaster bem integradas.		
		Encontro muitas inconsistências na ferramenta CodeMaster.	Encontro muitas inconsistências na ferramenta CodeMaster.		
		Imagino que as pessoas aprenderão rapidamente a usar a ferramenta CodeMaster.	Imagino que as pessoas aprenderão rapidamente a usar a ferramenta CodeMaster.		
		Não acho a ferramenta CodeMaster prática de usar.	Não acho a ferramenta CodeMaster prática de usar.		
		Senti-me confiante ao usar a ferramenta CodeMaster.	Senti-me confiante ao usar a ferramenta CodeMaster.		

		Precisei aprender muitas coisas antes de ser capaz de operar a ferramenta CodeMaster.	Precisei aprender muitas coisas antes de ser capaz de operar a ferramenta CodeMaster.		
	Operabilidade	Você achou fácil usar a ferramenta CodeMaster?	Você achou fácil usar a ferramenta CodeMaster?		
		Você acha que a ferramenta CodeMaster possui elementos ambíguos ou difíceis de entender?	Você acha que a ferramenta CodeMaster possui elementos ambíguos ou difíceis de entender?		

Os dados são coletados por meio de:

- Avaliações com usuários, para obter dados sobre eficácia das tarefas bem como sobre a qualidade percebida do ponto de vista de professores e alunos.
- Testes de corretude comparando o resultado de avaliações feitas com a ferramenta CodeMaster com o resultado de avaliações manuais.

5.1.1 DEFINIÇÃO DA AVALIAÇÃO COM USUÁRIOS

A avaliação com usuários é, basicamente, um teste de usabilidade como uma forma sistemática de observar usuários reais testando um produto e coleta de informações sobre os modos específicas pelos quais o produto é fácil ou difícil para eles. Primeiramente os usuários recebem uma visão geral básica sobre o objetivo da ferramenta CodeMaster e então eles executam uma tarefa predefinida (avaliar um projeto ou conjunto de projetos de programação com a ferramenta CodeMaster). A descrição da tarefa é apresentada no Anexo B.

Ao final do teste os participantes respondem um questionário (Anexo B) que foi derivado a partir dos fatores de qualidade definidos na Tabela 24. Foi utilizada, basicamente, uma escala nominal para os itens do questionário (sim/não), com exceção dos itens relativos à satisfação. Essa subcaracterística de qualidade foi medida adotando o questionário SUS. Os usuários também foram solicitados a identificar os pontos fortes e pontos fracos da ferramenta.

5.1.2 DEFINIÇÃO DA AVALIAÇÃO DA CORRETUDE

Para realizar os testes de corretude, primeiramente, dez projetos Snap! são selecionados para avaliação. Como o Snap! não possui uma galeria de projetos em sua página oficial, os dez projetos são selecionados aleatoriamente entre projetos encontrados na única galeria de projetos Snap! encontrada (NATHALIERUN, 2017). Os projetos desta galeria são licenciados com Creative Commons Attribution-NonCommercial-ShareAlike 4.0

Internacional (CC BY-NC-AS 4.0). Os projetos, então, são avaliados manualmente pelo presente autor com base na rubrica definida para a ferramenta CodeMaster (Tabela 14). Os mesmos projetos são avaliados automaticamente pela ferramenta CodeMaster e os resultados são comparados.

5.2 EXECUÇÃO DA AVALIAÇÃO

As avaliações foram realizadas durante o segundo semestre de 2017 e ocorreram conforme definidas.

5.2.1 EXECUÇÃO DA AVALIAÇÃO COM USUÁRIOS

Os usuários convidados para participar da avaliação da ferramenta CodeMaster (Figura 39) foram selecionados por já possuir algum nível de experiência com linguagens de programação visual, como Scratch e Snap! e por critério de proximidade geográfica do presente autor e proximidade com a iniciativa Computação na Escola. Foram convidados dois tipos de usuários, professores e alunos do Ensino Básico, representando dois atores do sistema. Três professores e três alunos foram convidados, todos participaram da avaliação.

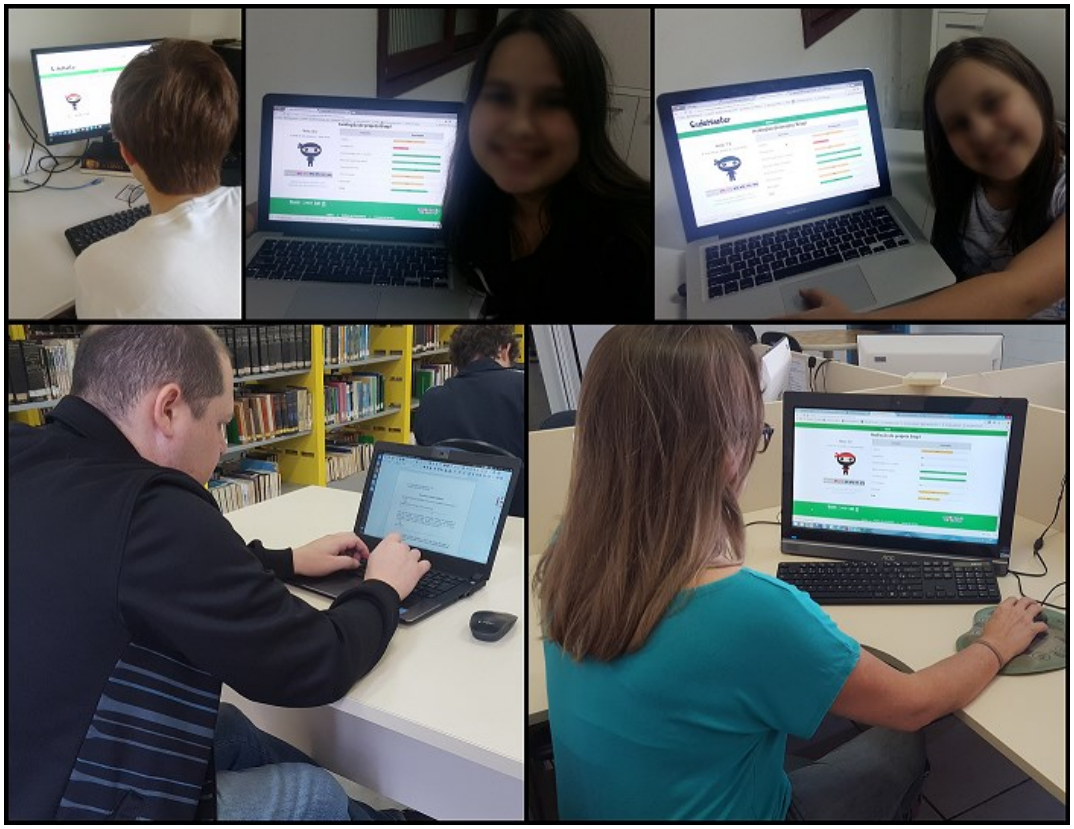
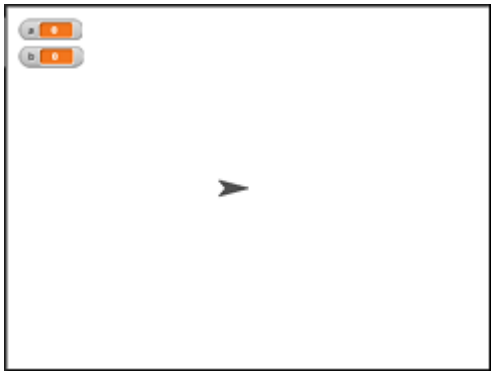
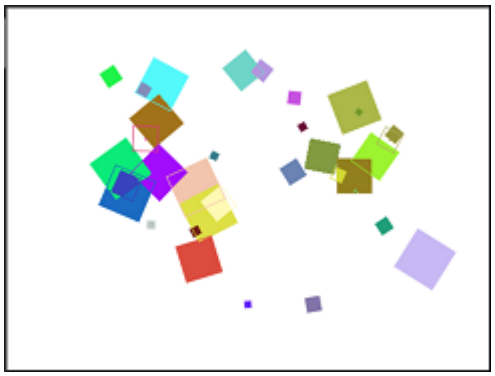


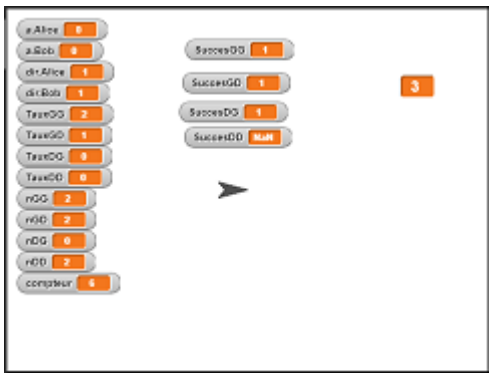
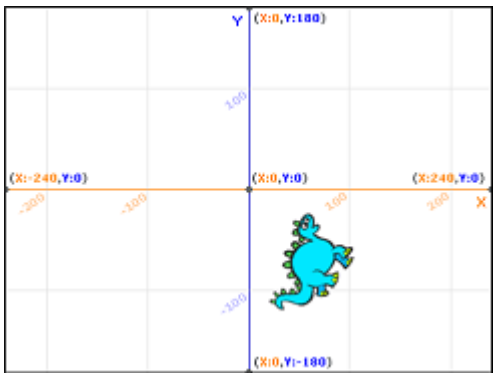
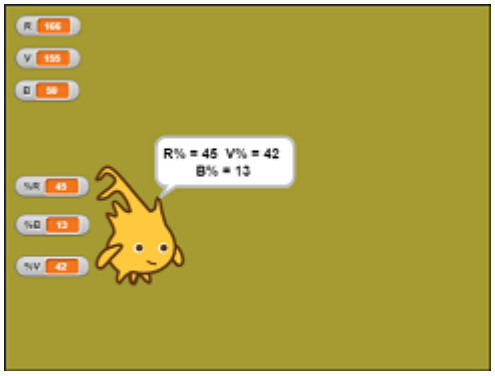
Figura 39 - Usuários que participaram da avaliação

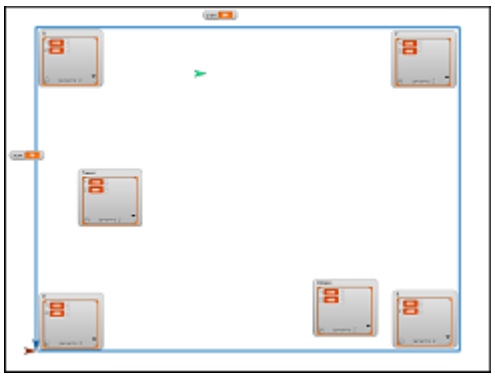
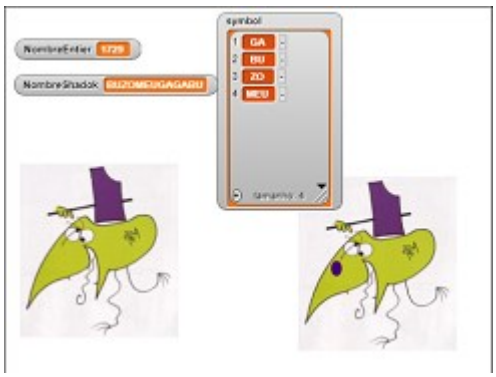
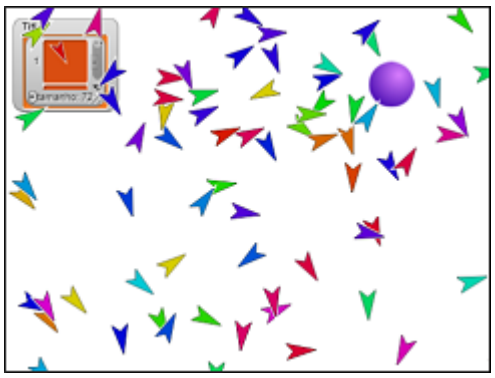
5.2.2 EXECUÇÃO DA AVALIAÇÃO DA CORRETUDE

Para executar a avaliação da corretude, dez projetos foram selecionados aleatoriamente entre projetos encontrados na galeria de projetos Snap! Nathalierun (2017). Os projetos selecionados são apresentados na Tabela 24.

Tabela 24 - Projetos selecionados para avaliação

Nome	Disponível em	Captura de tela
Pgcd	https://nathalierun.net/snap/Snap.Galerie/Algorithme.Euclide/pgcd.xml	
GeometrieAnalytiqueNathanModifie	https://nathalierun.net/snap/Snap.Galerie/Geometrie.Reperee/GeometrieAnalytiqueNathanModifie.xml	
CarresRemplis	https://nathalierun.net/snap/Snap.Galerie/CarresEnFolie/CarresRemplis.xml	

JeuDeBellF	https://nathalierun.net/snap/Snap.Galerie/LeJeuDeBell/JeuDeBellF.xml	
Dinosaure.A.VitesseConstante	https://nathalierun.net/snap/Snap.Galerie/Scratch.Projects/Dinosaure.Direction.VitesseConstante/Dinosaure.A.VitesseConstante.xml	
RVB.Couleur	https://nathalierun.net/snap/Snap.Galerie/Couleurs.RVB/RVB.Couleur.xml	
CalculsSurLesCoordonnees	https://nathalierun.net/snap/Snap.Galerie/Scratch.Projects/CalculsSurLesCoordonnees/CalculsSurLesCoordonnees.xml	

ChasseAuTresor.Simulation.1200x900	https://nathalierun.net/snap/Snap.Galerie/ChasseAuTresor/ChasseAuTresor.Simulation.1200x900.xml	
ConvertisseurShadokEntier	https://nathalierun.net/snap/Snap.Galerie/Convertisseur.Shadok/ConvertisseurShadokEntier.xml	
Comprendre.Clones	https://nathalierun.net/snap/Snap.Galerie/Clones/Comprendre.Clones.xml	

O código de cada um dos dez projetos foi aberto para realizar a avaliação manual. Uma pontuação foi dada para cada conceito de acordo com a rubrica da ferramenta CodeMaster. Após a avaliação manual, os dez projetos foram submetidos à avaliação da ferramenta CodeMaster no modo professor.

5.3 ANÁLISE DOS DADOS

Nesta seção são analisados os dados das avaliações com usuários e os dados da avaliação da corretude.

5.3.1 ANÁLISE DOS DADOS DAS AVALIAÇÕES COM USUÁRIOS

Os dados das avaliações com usuários são analisados em relação aos objetivos de avaliação.

O CodeMaster é útil?

Todos os usuários consideram o CodeMaster útil para a aprendizagem e ensino de computação no Ensino Básico. A forma de fazer o upload de projetos foi considerada positiva, porém algumas sugestões foram dadas, como por exemplo, a criação de uma galeria de projetos no site do CodeMaster possibilitando que o próprio aluno faça o envio do projeto para o professor via o sistema. Os dados coletados sobre a utilidade são apresentados na Tabela 25.

Tabela 25 - Avaliação da utilidade da ferramenta CodeMaster

Questão de análise	**Total de respostas**	
	Sim	**Não**
Você acha a ferramenta CodeMaster útil na aprendizagem de programação?	3 alunos	0 alunos
Você acha a ferramenta CodeMaster útil no ensino de computação no ensino Básico?	3 professores	0 professores
Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) a ferramenta CodeMaster é prática no seu dia-a-dia?	2 professores	1 professor
Comentários		
“Pode ser atrativo para os estudantes”.		
“Criar uma galeria na própria plataforma”.		
“Acredito que no ensino fundamental, uma avaliação qualitativa do projeto pode ser mais relevante. Na minha visão educacional do uso de informática na educação, não pretendo tornar os alunos programadores, mas que possam conseguir produzir mídias utilizando mais este recurso – expressão mais popular seria “saber se expressar usando tecnologias”… Mas a relevância do projeto está em avaliar também a parte técnica, a construção dos produtos, se a turma consegue avançar na aprendizagem dos conceitos ligados também a programação, mesmo que não seja o foco principal na minha aula...”		
“Porque ele desenvolve capacidades e habilidades como lógica, concentração, organização, criatividade, pesquisa utilizando as tecnologias. Espera-se também que o interesse pela utilização de tecnologias pode motivar a aprendizagem dos conteúdos disciplinares.”		

O CodeMaster é adequado funcionalmente?

Em relação à adequação funcional, a ferramenta recebeu uma avaliação positiva dos usuários. O *feedback* apresentado como resultado da avaliação foi considerado completo, nenhum erro (*bug*) foi identificado e a nota foi considerada justa pelos usuários. Algumas observações foram feitas sobre os critérios de avaliação, por exemplo, que eles devem estar bem claros no sistema. Também foram sugeridos novos critérios como “jogabilidade”. Os dados coletados sobre a adequação funcional são apresentados na Tabela 26.

Tabela 26 - Avaliação da adequação funcional da ferramenta CodeMaster

Questão de análise	Total de respostas	
	Sim	Não
Você acha que existem aspectos relevantes no processo de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta CodeMaster?	1 professor	2 professores
Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?	1 professores	2 professores
Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	3 professores 3 alunos	0 professores 0 alunos
Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?	0 professores 0 alunos	3 professores 3 alunos
Você achou a nota justa?	3 alunos	0 alunos
Comentários		
“Na minha atividade o interesse maior recai sobre a “jogabilidade” dos produtos desenvolvidos pelos alunos.”		
“É preciso utilizar a ferramenta na prática para perceber se existem outros aspectos/critérios para serem inseridos.”		

O CodeMaster tem desempenho satisfatório?

Na análise do desempenho a ferramenta recebeu um *feedback* positivo, porém, sabe-se que o fator determinante para o desempenho é a velocidade e qualidade de conexão de internet do usuário. Como cada projeto tem que ser enviado ao CodeMaster para análise, projetos maiores podem ter um tempo de resposta maior também. Os dados coletados sobre o desempenho são apresentados na Tabela 27.

Tabela 27 - Avaliação do desempenho da ferramenta CodeMaster

Questão de análise	Total de respostas	
	Sim	Não
A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória?	3 professores 3 alunos	0 professores 0 alunos

O CodeMaster tem boa usabilidade?

Durante a avaliação os usuários também responderam o questionário SUS. O resultado é apresentado na Tabela 28

.

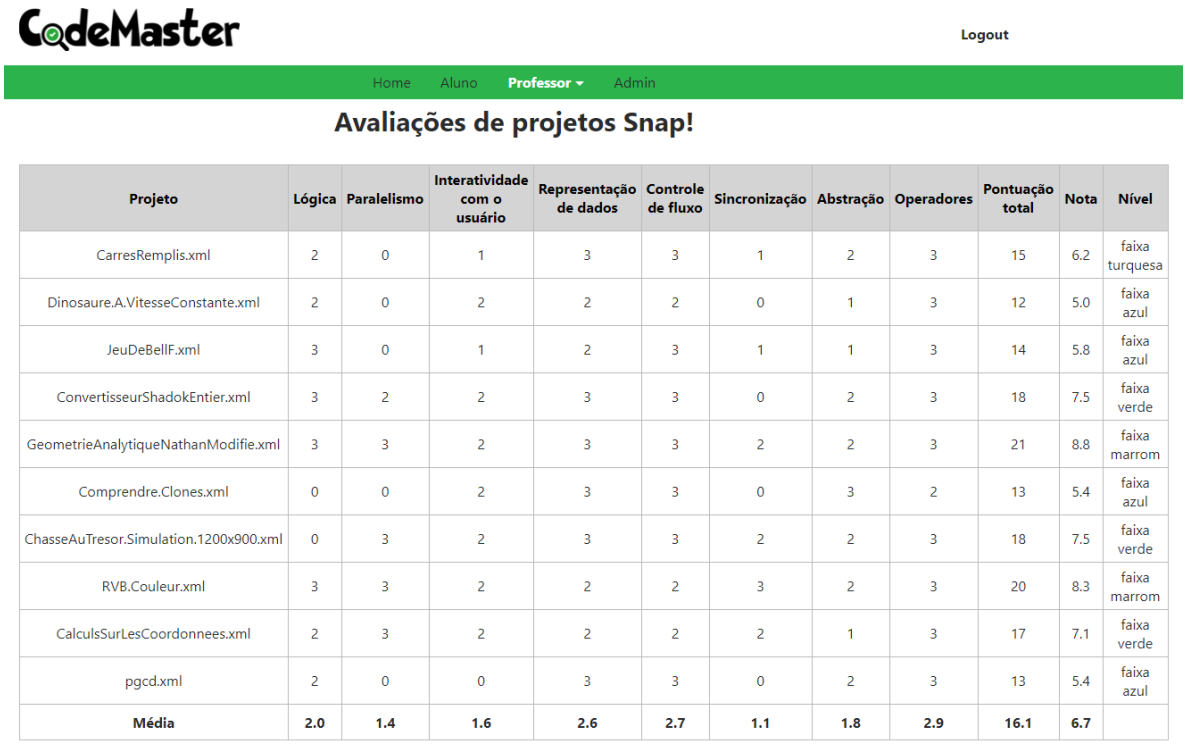
Tabela 28 - Resultado da aplicação do questionário SUS

Usuário	Satisfação (SUS)
Professor 1	75
Professor 2	92,5

Professor 3	100
Aluno 1	82,5
Aluno 2	95
Aluno 3	95
Média	**90**

5.3.2 ANÁLISE DOS DADOS DA AVALIAÇÃO DA CORRETUDE

A avaliação realizada pela ferramenta CodeMaster é apresentada na Figura 40. Os resultados das avaliações de dez projetos Snap! realizados com a ferramenta CodeMaster e avaliações manuais são comparados na Tabela 29.



Projeto	Lógica	Paralelismo	Interatividade com o usuário	Representação de dados	Controle de fluxo	Sincronização	Abstração	Operadores	Pontuação total	Nota	Nível
CarresRemplis.xml	2	0	1	3	3	1	2	3	15	6.2	faixa turquesa
Dinosaure.A.VitesseConstante.xml	2	0	2	2	2	0	1	3	12	5.0	faixa azul
JeuDeBellF.xml	3	0	1	2	3	1	1	3	14	5.8	faixa azul
ConvertisseurShadokEntier.xml	3	2	2	3	3	0	2	3	18	7.5	faixa verde
GeometrieAnalytiqueNathanModifie.xml	3	3	2	3	3	2	2	3	21	8.8	faixa marrom
Comprendre.Clones.xml	0	0	2	3	3	0	3	2	13	5.4	faixa azul
ChasseAuTresor.Simulation.1200x900.xml	0	3	2	3	3	2	2	3	18	7.5	faixa verde
RVB.Couleur.xml	3	3	2	2	2	3	2	3	20	8.3	faixa marrom
CalculsSurLesCoordonnees.xml	2	3	2	2	2	2	1	3	17	7.1	faixa verde
pgcd.xml	2	0	0	3	3	0	2	3	13	5.4	faixa azul
Média	**2.0**	**1.4**	**1.6**	**2.6**	**2.7**	**1.1**	**1.8**	**2.9**	**16.1**	**6.7**	

Figura 40 - Avaliação realizada pela ferramenta CodeMaster

Analisando a Tabela 29, que resume os testes de corretude, é possível identificar uma diferença entre a avaliação do projeto “ChasseAuTresor.Simulation.1200x900” realizada manualmente e a avaliação realizada pela ferramenta CodeMaster. Essa diferença entre as avaliações permitiu detectar e corrigir uma falha na ferramenta CodeMaster, que não detectava o bloco “envie a todos e espere”, representado no código pelo *token “doBroadcastAndWait”*. Todos os outros testes de corretude não apontaram nenhuma imprecisão da ferramenta.

Tabela 29 - Comparação entre os resultados das avaliações CodeMaster e avaliações manuais

Projeto	Método de avaliação	Lógica	Paralelismo	Interatividade com o usuário	Representação de dados	Controle de fluxo	Sincronização	Abstração	Operadores	Total	Nota	Nível
pgcd	CodeMaster	2	0	0	3	3	0	2	3	13	5,4	Faixa azul
	Manual	2	0	0	3	3	0	2	3	13	5,4	Faixa azul
GeometrieAnalytiqueNathanModifie	CodeMaster	3	3	2	3	3	2	2	3	21	8,8	Faixa marrom
	Manual	3	3	2	3	3	2	2	3	21	8,8	Faixa marrom
CarresRemplis	CodeMaster	2	0	1	3	3	1	2	3	15	6,2	Faixa turquesa
	Manual	2	0	1	3	3	1	2	3	15	6,2	Faixa turquesa
JeuDeBellF	CodeMaster	3	0	1	2	3	1	1	3	14	5,8	Faixa azul
	Manual	3	0	1	2	3	1	1	3	14	5,8	Faixa azul
Dinosaure.A.VitesseConstante	CodeMaster	2	0	2	2	2	0	1	3	12	5,0	Faixa azul
	Manual	2	0	2	2	2	0	1	3	12	5,0	Faixa azul
RVB.Couleur	CodeMaster	3	3	2	2	2	3	2	3	20	8,3	Faixa marrom
	Manual	3	3	2	2	2	3	2	3	20	8,3	Faixa marrom
CalculsSurLesCoordonnees	CodeMaster	2	3	2	2	2	2	1	3	17	7,1	Faixa verde
	Manual	2	3	2	2	2	2	1	3	17	7,1	Faixa verde
ChasseAuTresor.Simulation.1200x900	CodeMaster	0	3	2	3	3	0	2	3	16	6,7	Faixa turquesa
	Manual	0	3	2	3	3	2	2	3	18	7,5	Faixa verde
ConvertisseurShadokEntier	CodeMaster	3	2	2	3	3	0	2	3	18	7,5	Faixa verde
	Manual	3	2	2	3	3	0	2	3	18	7,5	Faixa verde
Comprendre.Clones	CodeMaster	0	0	2	3	3	0	3	2	13	5,4	Faixa azul
	Manual	0	0	2	3	3	0	3	2	13	5,4	Faixa azul

Com esses resultados, é possível verificar a corretude do CodeMaster. Também, é possível verificar a corretude das fórmulas de cálculo da nota final e nível ninja. Além da avaliação formal com os 10 projetos citados, ao longo do desenvolvimento do CodeMaster muitos testes foram executados com projetos Snap! diferentes e os resultados também se mostraram consistentes com o definido na rubrica apresentada.

5.3.3 PONTOS FORTES

A ferramenta CodeMaster foi considerada útil no ensino de computação no ensino básico por todos os usuários. Os professores que participaram da avaliação comentaram que atualmente utilizam linguagens de programação visual para ensinar conceitos de outras disciplinas, como história, mas que mesmo nesse caso a ferramenta CodeMaster é útil para acompanhar o progresso dos alunos. O *feedback* apresentado como resultado das avaliações e performance da ferramenta também foram avaliados de forma positiva por todos os usuários. A facilidade de entender os elementos e utilização da ferramenta receberam avaliações positivas da maioria dos usuários.

5.3.4 SUGESTÕES DE MELHORIA

Poucas sugestões de melhoria foram feitas pelos usuários. Uma sugestão foi a adicionar novos conceitos, para projetos mais específicos, como conceitos relacionados à criação de jogos. Outra sugestão foi a criação de uma galeria de projetos, em que professores e alunos compartilhassem projetos, facilitando o recebimento dos projetos e a avaliação por parte do professor.

5.3.5 AMEAÇAS À VALIDADE

Nesta avaliação inicial da ferramenta pode-se observar diversos fatores que, de alguma forma, podem influenciar ou ameaçar o resultado da avaliação. Devido ao curto prazo para realização da avaliação e pouco contato com usuários de Snap!, a avaliação foi feito como uma amostra pequena, porém razoável (três professores e três alunos), com baixa representatividade – visto que todos têm proximidade com a mesma instituição (Computação na Escola). Como algumas funcionalidades, como exibir a rubrica da ferramenta e seleção dos conceitos que são avaliados, ainda estavam sendo melhoradas e/ou implementadas durante o processo de avaliação, alguns problemas estavam sendo observados, o que pode ter prejudicado a avaliação. Os usuários utilizaram diferentes computadores e diferentes conexões com a internet para realizar a avaliação, o que pode alterar a experiência do usuário.

A quantidade de projetos e a escolha dos projetos utilizados nos testes de corretude da ferramenta também podem influenciar o resultado da avaliação. Para mitigar esses riscos dez projetos foram selecionados, por ser uma quantidade razoável, e os dez projetos foram selecionados aleatoriamente entre projetos encontrados na única galeria de projetos Snap! encontrada (NATHALIERUN, 2017).

6. CONCLUSÃO

O objetivo geral do presente trabalho foi desenvolver uma ferramenta web para analisar e avaliar automaticamente código desenvolvido com a linguagem de programação para iniciantes Snap!, a ser utilizada em unidades instrucionais no ensino de computação no Ensino Básico. Neste contexto foi feita a análise da fundamentação teórica sobre aprendizagem, ensino de computação no Ensino Básico, sobre a ferramenta de desenvolvimento Snap! e sobre análise de código **(O1)**. Em seguida foi realizado um estudo de mapeamento da literatura e identificado que existem poucos analisadores de código de linguagens de programação visual baseadas em blocos **(O2)**.

Desta forma, com base na fundamentação teórica e na análise do estado da arte foi desenvolvida uma ferramenta web para análise e avaliação automatizada de projetos Snap!, nomeada de CodeMaster – Snap! **(O3)**. A ferramenta proposta atende as necessidades de dois tipos de usuários, alunos e professores. Os alunos, sem necessidade de cadastro, podem realizar *upload* de apenas um projeto por vez, a ferramenta realiza a análise, avaliação e apresenta o resultado de forma completa. Os professores, previamente cadastrados, podem realizar upload de múltiplos projetos de uma vez, a ferramenta realiza a análise, avaliação e apresenta os resultados de forma resumida, apenas uma visão geral das notas.

Posteriormente, a ferramenta CodeMaster foi avaliada **(O4)**. Avaliações com usuários, para obter dados sobre eficácia das tarefas bem como sobre a qualidade percebida do ponto de vista de professores e alunos e testes de corretude comparando o resultado de avaliações feitas com a ferramenta CodeMaster com o resultado de avaliações manuais. As avaliações sobre sua utilidade no ensino de computação no ensino básico, seu *feedback* apresentado como resultado das avaliações, a clareza de seus elementos, sua performance e sua facilidade de utilização foram muito positivas. Nos testes de corretude a ferramenta também foi bem avaliada, apenas uma falha foi detectada e corrigida.

Espera-se que a automatização do processo de análise e avaliação de projetos facilite a avaliação da aprendizagem dos alunos em relação a projetos de programação desenvolvidos com Snap!, reduzindo o tempo despendido e esforço dos professores, aumentando as chances de o ensino de computação ser adotado mais amplamente nas escolas.

Como trabalho futuro é sugerida a adição de um novo critério de avaliação à ferramenta, para avaliação de jogos desenvolvidos com Snap!, além da criação de uma galeria de projetos, para que professores e alunos possam compartilhar projetos, facilitando

a avaliação e o recebimento dos projetos por parte do professor. Também é sugerida a alteração do sistema para bloquear o *upload* de arquivos com formatos inválidos (diferentes de .xml ou .aia). A expansão do sistema para permitir que pesquisadores realizem avaliação de projetos e recebam um feedback mais detalhado (p. ex. quais blocos foram programados, más práticas de programação, etc.) também é sugerida.

REFERÊNCIAS

AULD, R., BELFIORE, P., & SCHEELER, M. (2010). Increasing pre-service teachers' use of differential reinforcement: Effects of performance feedback on. Journal of Behavioral Education, 19(1), pp. 169–183.

BALL, M. λ - An Autograder for Snap!, 2016. Disponível em:<https://www.gitbook.com/book/cycomachead/thesis/details>.
Acesso em: Março 2017.

BASQUE, J. Ingénierie Pédagogique Et Technologies Éducatives. Québec: Télé-université, 2010.

BILAL, M., CHAN, P., MEDDINGS, F., & KONSTADOPOULOU, A. (2012). SCORE: An advanced assessment and feedback framework with a universal marking scheme in higher education. In Proc. of the Int. Conf. on Education and e-Learning Innovations, (pp. 1-6). Sousse/Tunísia

BINKLEY, D. 2007. Source Code Analysis: A Road Map. In *2007 Future of Software Engineering* (FOSE '07). IEEE Computer Society, Washington, DC, USA, 104-119. DOI=http://dx.doi.org/10.1109/FOSE.2007.27

BLACK, P.; WILIAM, D. (1996). Meanings and Consequences: A Basis for Distinguishing Formative and Summative Functions of Assessment? British Educational Research Journal, 22(5), 537-548.

BLACK, P.; WILIAN, D. (1998). Assessment and classroom learning. Assessment in Education: Principles, Policy & Practice, 5(1), pp. 7–74.

BOE, B.; HILL, C.; LEN, M.; DRESCHLER, G.; CONRAD, P.; FRANKLIN, D. 2013. Hairball: lint-inspired static analysis of scratch projects. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (SIGCSE '13). Denver, USA.

BRANCH, R. M. Instructional Design: The ADDIE Approach. New York: Springer, 2009.

BRASSCOM. Índice BRASCOM de Convergência Digital, 2012. Disponível em: <www.brasscom.org.br/brasscom/Portugues/download.php?cod=437>. Acesso em: Setembro 2016.

BRENNAN, K.; RESNICK, M. (2012). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada (pp. 1-25).

BROOKE, J. et al. Sus-a quick and dirty usability scale. Usability evaluation in industry, London, v. 189, n. 194, p. 4–7, 1996.

CISCO. Networking Skills in Latin America, 2016. Disponível em: <http://www.cisco.com/assets/csr/pdf/IDC_Skills_Gap_-_LatAm.pdf>. Acesso em: Setembro 2016.

CSTA. ACM. CSTA K –12 Computer Science Standards, 2011. Disponível em: <https://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf>. Acesso em:

Setembro 2016.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, p. 319–340, 1989.

DEMETRIO, M. F. (2017) Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação. 2017. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Department of Defense Handbook. 1988. *Instructional System Development/ System Approach to Training and Education (part 2 of 5)*. MIL-HDBK-29613-2A. 31 Oct 1988.

DETERDING, S.; DIXON, D.; KHALED, R.; NACKE, L. 2011. From game design elements to gamefulness: defining "gamification". In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland.

EMANUELSSON, P.; NILSSON, U. 2008. A comparative study of industrial static analysis tools. Electronic notes in theoretical computer science, v. 217, p. 5-21.

FILATRO, A. Design instrucional contextualizado: educação e tecnologia. São Paulo: SENAC, 2004.

GOHN, M. 2006. Educação não-formal, participação da sociedade civil e estruturas colegiadas nas escolas. Rio de Janeiro, Brasil. 14(50), p. 27-38.

GOMES, I.; MORGADO, P.; GOMES, T.; MOREIRA, R. 2009. An overview on the Static Code Analysis approach in Software Development. Faculdade de Engenharia da Universidade do Porto, Portugal. Disponível em: < https://web.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf>. Acesso em: Junho 2017.

HADDAWAY, N. R. et al. The role of Google Scholar in evidence reviews and its applicability to grey literature searching. PloS one, v. 10, n. 9, 2015.

HALLBERG, C. 2017. Comunicação pessoal.

HUANG, W. H. Y.; SOMAN, D. 2013. Gamification of education. Research Report Series: Behavioural Economics in Action, Rotman School of Management, University of Toronto. Disponível em: <https://pdfs.semanticscholar.org/c1df/e1970305f257b08a9f2b9844b346452eb869.pdf>. Acesso em: Junho 2017.

INEP. Sinopses Estatísticas da Educação Superior – Graduação. Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Brasília. 2010 a 2014.

ISO/IEC-25010 (2011). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

ISSO/IEC-9241-210:2010 - Ergonomicsof human-system interaction – Part 210: Human-centred design for interactive systems. 1. ed. [S.l.], 2010.

JAX-RS API, Java™ API for RESTful Web Services (JAX-RS) delivers API for RESTful Web Services development in Java SE and Java EE. 2017. Disponível em: < https://github.com/jax-rs>. Acesso em: julho de 2017.

JERSEY, RESTful Web Services in Java. 2017. Disponível em: < https://jersey.github.io/ >. Acesso em: julho de 2017.

JONES, T. Static and dynamic testing in the software development life cycle, 2013. Disponível em: <https://www.ibm.com/developerworks/library/se-static/se-static-pdf.pdf>. Acesso em: Novembro 2016.

JONSSON, A. SVINGBY, G. 2007. The use of scoring rubrics: Reliability, validity and educational consequences. Educational Research Review, 2(2), pp.130-144.

JOHNSON, D., & JOHNSON, R. (1993). Cooperative learning and feedback in technology-based instruction. In: J. DEMPSEY, Interactive instruction and feedback (pp. 133–157). Englewood Cliffs: Educational Technology Publications.

KITCHENHAM, B. 2004. Procedures for Performing Systematic Reviews. Joint Technical Report, TR/SE-0401 and NICTA 0400011T.1: Keele University, Keele, Reino Unido.

LIBÂNEO, J. C. 2009. Didática. 29 ed. Cortez. 264 p.

LIU, C. (2010). The comparison of learning effectiveness between traditional faceto-face learning and e-learning among goal-oriented users. In Proc. of the 6th Int. Conf. on Digital Content, Multimedia Technology and its Applications, Seoul/South Korea.

LYE, S. Y., KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 2014, 41, 51-61.

MERRILL, M. D. et al. Reclaiming instructional design. Educational Technology, 36(5), p. 5-7, 1996.

MICHAELIS. Dicionário Brasileiro da Língua Portuguesa. Disponível em: <http://michaelis.uol.com.br/>. Acesso em: Setembro 2016.

MORENO-LEÓN, J.; ROBLES, G. 2015. Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. In: Proceedings of the 10th Workshop in Primary and Secondary Computing Education, London, UK.

MYSQL, 2017. Disponível em: < https://www.mysql.com/>. Acesso em: agosto de 2017.

NATHALIERUN, Galerie de mes projets Snap!. Disponível em: <https://nathalierun.net/snap/Snap.Galerie/>. Acesso em: Outubro 2017.

NIELSON, F.; NIELSON, H. R.; HANKIN, C. Principles of Program Analysis. 2 ed. Berlin: Springer, 2015. 452 p.

OTA, G.; MORIMOTO, Y.; KATO, H. 2016. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Cambridge, UK.

PCN, Parâmetros Curriculares Nacionais, Terceiro e Quarto ciclos do Ensino Fundamental. Disponível em: <http://portal.mec.gov.br/seb/arquivos/pdf/introducao.pdf>. Acesso em: Outubro de 2016.

PETERSEN, K. et al. Systematic Mapping Studies in Software Engineering. In: EASE. 2008, p. 68-77.

PRESSMAN, R. Engenharia de software: Uma abordagem profissional. 8 ed. Porto Alegre: Bookman, 2016. 968 p.

PROGRAMAMOS; DR.SCRATCH. Dr.Scratch – Analise seus projetos Scratch aqui. Disponivel em: <http://www.drscratch.org/>. Acesso em: Dezembro 2016.

RICHARDS, M., & SCHIFFEL, J. (2005). A distance learning framework for automatic instructor replies: articulable tacit knowledge used for feedback upon request. In Proceedings of the IEEE SoutheastCon, pp. 611-620, Lauderdale/USA.

SCRATCH; MIT. Scratch, 2013. Disponivel em: <http://scratch.mit.edu>. Acesso em: Novembro 2016.

SNAP!. Snap! Reference Manual Version 4.0, 2013. Disponível em: <http://snap.berkeley.edu/SnapManual.pdf>. Acesso em: Novembro 2016.

SNAP!; BERKELEY. Snap!, 2013. Disponível em: <http://snap.berkeley.edu>. Acesso em: Setembro 2016.

SOFTEX. Relatório Anual, 2014. Disponível em: <http://www.softex.br/wp-content/uploads/2015/04/Relatorio_Anual_2014.pdf>. Acesso em: Setembro 2016.

STRIEWE, M.; GOEDICKE, M. 2014. A Review of Static Analysis Approaches for Programming Exercises. In: International Computer Assisted Assessment Conference. Zeist, Holanda. Springer International Publishing. p. 100-113.

WAZLAWICK, R. Engenharia de software: conceitos e práticas. Elsevier Brasil, 2013.

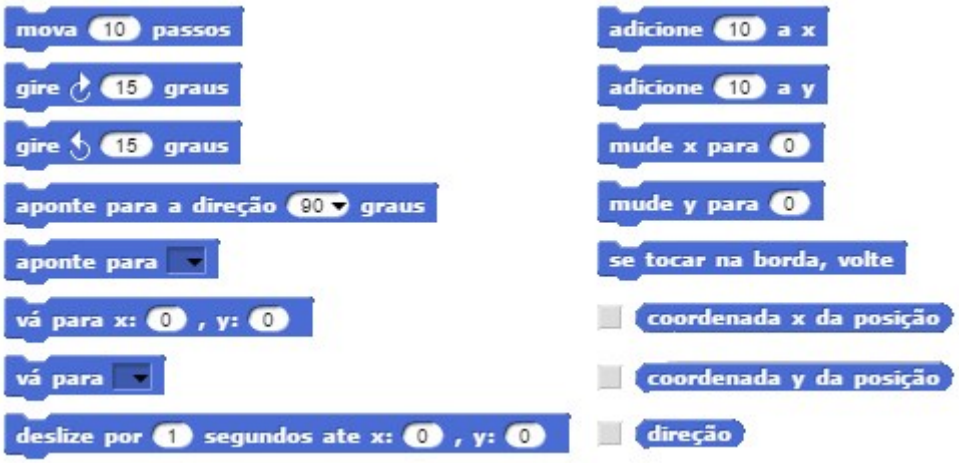
WING, J. M. Computational thinking. Communications of the ACM, v. 49, n. 3, 2006.

WIRTH, N. Compiler Construction. 1 ed. Redwood City, USA: Addison-Wesley Pub (Sd), 1996. 131 p.

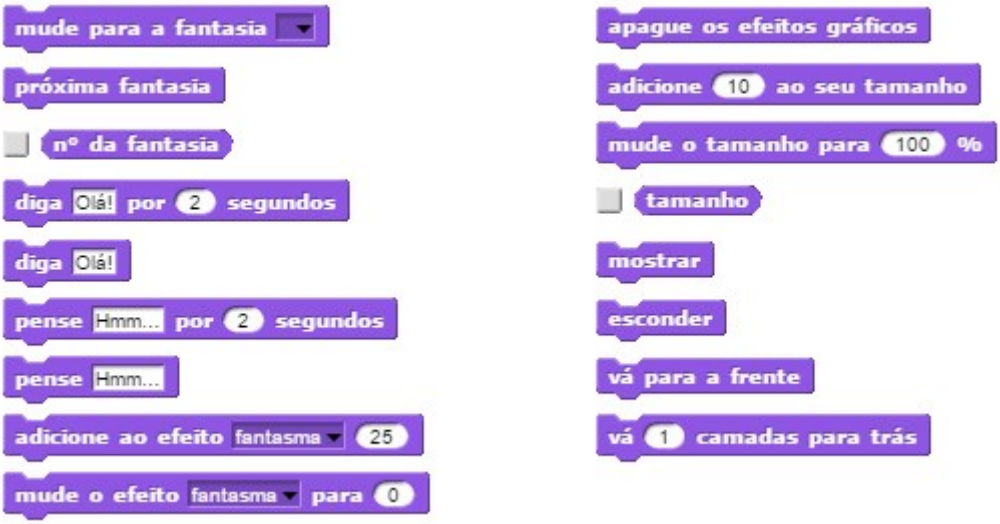
WOLZ, U.; HALLBERG, C.; TAYLOR, B. 2011. Scrape: A tool for visualizing the code of Scratch programs. In: Proceedings of 42nd ACM Technical Symposium on Computer Science Education, Dallas, USA. 2011.

Anexo A – Os blocos disponíveis no Snap! (versão 4.0)

- **Blocos da categoria Movimento** (alteram a posição e ângulo dos atores):

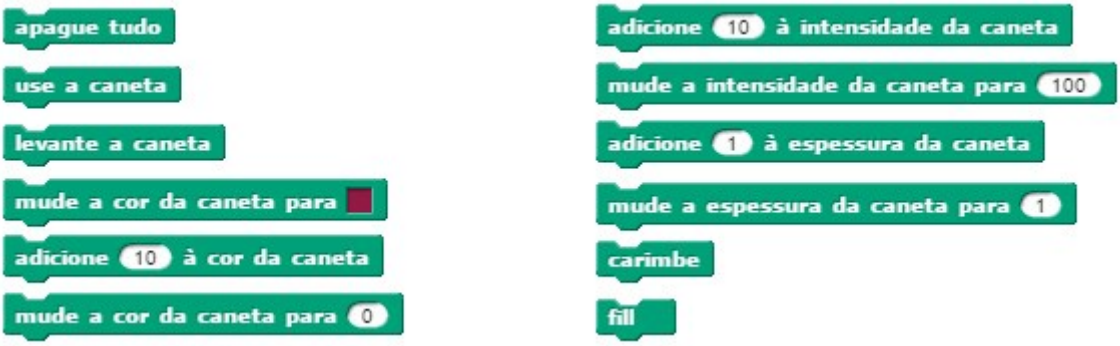


- **Blocos da categoria Aparência** (alteram a visibilidade, fantasia, efeitos, tamanho, falas do ator, etc.):



- **Blocos da categoria Som** (controlam quais sons devem ser reproduzidos, em que velocidade e volume, etc.):

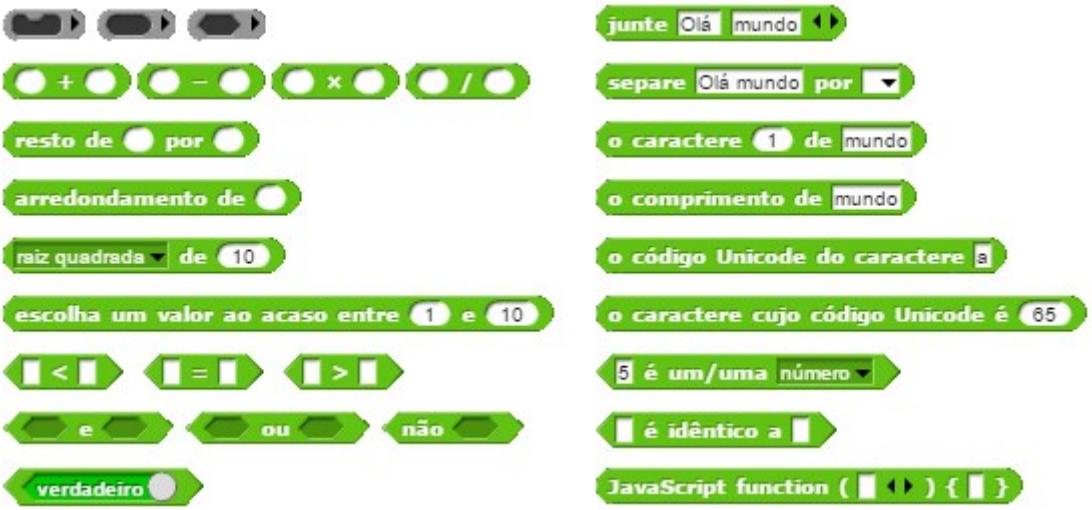
- **Blocos da categoria Caneta** (Desenham o que um ator faz na tela, alteram tamanho e cor da caneta, etc.):



- **Blocos da categoria Controle** (Alteram o fluxo de execução como laços, condicionais, etc.):

- **Blocos da categoria Sensores** (Detecção de entradas do usuário, sensores de cor, posição do mouse e dos atores, etc.):

- **Blocos da categoria Operadores** (Operadores matemáticos e booleanos):



- **Blocos da categoria Variáveis** (Criar variáveis ou listas, incluindo listas de listas e operá-las.):

Anexo B – Avaliação com os usuários

Convite para professores

Olá,
Gostaríamos de convidar você para participar da avaliação do CodeMaster, uma ferramenta online que analisa e avalia projetos programados no App Inventor e Snap! no contexto do ensino de programação em escolas.
A ferramenta foi desenvolvida como Trabalho de Conclusão de Curso de Rafael Pelle sob a orientação da Profa. Christiane Gresse von Wangenheim realizado na iniciativa Computação na Escola no INCoD/INE/UFSC.
Na avaliação solicitamos que você simule a avaliação de projetos programados no Snap!. Snap! (http://snap.berkeley.edu/snapsource/snap.html) é um ambiente de programação visual (similar ao Scratch) que permite criar jogos e animações, e visualizar sua execução. Gostaríamos de solicitar que você faça um programa com Snap! para conhecer o ambiente.
Disponibilizaremos para você um conjunto de projetos Snap! prontos para baixar e simular os resultados de uma turma e solicitamos que envie os projetos ao sistema CodeMaster simulando a avaliação de vários projetos de diferentes alunos, e ao final, o preenchimento de um questionário sobre a utilidade e usabilidade da ferramenta. No total tudo isso não deve levar mais do que 45 min.
Para facilitar a avaliação gostaríamos de realizar a avaliação junto com vocês para que possamos esclarecer quaisquer dúvidas que possam surgir no decorrer da tarefa. Caso você esteja de acordo, favor enviar e-mail para rafapelle@gmail.com para agendar um horário para a avaliação a ser realizado na escola onde você atua.
A sua participação é voluntaria sem remuneração. Todos os seus dados serão tratados de forma sigilosa usados somente para fins de pesquisa.
Desde já, muito obrigado pela ajuda! O seu *feedback* é muito importante para nossa pesquisa.
Ao final estaremos disponibilizando a ferramenta gratuitamente no site da iniciativa Computação na Escola (http://www.computacaonaescola.ufsc.br) para qualquer interessado.

Convite para alunos

Olá,
Gostaríamos de convidar você para participar da avaliação do CodeMaster, uma ferramenta online que analisa e avalia projetos programados no Snap!.
A ferramenta foi desenvolvida como Trabalho de Conclusão de Curso de Rafael Pelle sob a orientação da Profa. Christiane Gresse von Wangenheim realizado na iniciativa de Computação na Escola no INCoD/INE/UFSC.
Na avaliação solicitamos que você faça um programa com Snap!. Snap! (http://snap.berkeley.edu/snapsource/snap.html) é um ambiente de programação visual (similar ao Scratch) que permite criar jogos e animações, e visualizar sua execução. Gostaríamos de solicitar que você faça um programa com Snap! para conhecer o ambiente.
Gostaríamos então que você envie o seu projeto para receber uma avaliação do CodeMaster. No final, você deve responder um questionário sobre a utilidade e usabilidade da ferramenta. No total tudo isso não deve levar mais do que 45min.
Em anexo segue um arquivo com todas as instruções (links de acesso e download do projeto a ser avaliado) bem como um passo a passo explicando como proceder a análise no CodeMaster. Sugerimos que você peça os seus pais para ajudar em caso de qualquer dúvida.
O teste pode ser feito online na sua casa mesmo. Ao final favor envie o questionário respondido via e-mail para: rafapelle@gmail.com. Agradecemos se podem também mandar uma foto da criança analisando a ferramenta - mostrando o aluno de trás e a tela com o CodeMaster.
A sua participação é voluntaria sem remuneração. Todos os seus dados serão tratados de forma sigilosa usados somente para fins de pesquisa.
Desde já, muito obrigado pela ajuda! O seu *feedback* é muito importante para nossa pesquisa.
Ao final estaremos disponibilizando a ferramenta gratuitamente no site da iniciativa Computação na Escola (http://www.computacaonaescola.ufsc.br) para qualquer interessado.

Passo a passo para professores

- Cenário

Imagine que você aplicou uma atividade ensinando a programação de software em uma turma na escola em que os alunos deveriam desenvolver um programa utilizando o Snap!. A atividade poderia ser feita individualmente, em duplas ou trios. Assumimos, que você recebeu ao todo 10 projetos Snap! dos seus alunos para avaliação. Para possibilitar a identificação dos projetos, cada projeto já está nomeado corretamente, pelos próprios alunos, indicando os nomes dos alunos no nome do arquivo xml (NomeAluno1-NomeAluno2-

NomeAluno3.xml). Agora você deseja avaliar estes 10 projetos de forma automatizada no CodeMaster para poder acompanhar o progresso de aprendizagem dos seus alunos e/ou dar uma nota.

- Passo a Passo

1. Faça um programa com Snap! para conhecer esse ambiente de programação. (Guia: Como criar e salvar um projeto Snap!)
2. Para ter uma ideia do nível de expertise do seu programa você pode analisar seu programa Snap! individualmente no CodeMaster, simulando o contexto de um aluno (Guia ilustrado: como avaliar projetos Snap! no CodeMaster).
3. Agora para conhecer a ferramenta CodeMaster no contexto de um professor, simule a avaliação de um conjunto de programas desenvolvidos pelos seus alunos.
   a. Baixar o pacote de projetos Snap! (Download de um pacote de 10 projetos Snap! prontos) e descompactar em uma pasta de sua preferência. (Alternativamente se já usou Snap! em uma das suas disciplinas você também pode usar os arquivos xml criados por seus alunos).
   b. Abrir o CodeMaster no navegador de internet de sua preferência (Chrome, Firefox, Internet Explorer): http://apps.computacaonaescola.ufsc.br:8080/
   c. Simule a avaliação de uma turma de alunos (Guia ilustrado: como avaliar projetos Snap! no CodeMaster).
4. Preencher o questionário disponível em: Questionário para professores
5. Enviar o questionário preenchido para: rafapelle@gmail.com

Passo a passo para alunos

- Cenário

Imagine que você está fazendo um programa com Snap!. Agora você quer saber até que ponto você esta dominando a programação – se ainda é um iniciante ou já é um mestre em programação. Assim, você abre o CodeMaster e analisa o seu projeto.

- Passo a passo

1. Faça um programa com Snap! para conhecer esse ambiente de programação. (Guia ilustrado: Como criar e salvar um projeto Snap!)
2. Abrir o CodeMaster no navegador de internet de sua preferência (Chrome, Firefox, Internet Explorer?): http://apps.computacaonaescola.ufsc.br:8080/
3. Avaliar seu projeto Snap! no CodeMaster (Guia ilustrado: como avaliar projetos Snap! no CodeMaster).
4. Preencher o questionário disponível em: Questionário para alunos
5. Enviar o questionário preenchido para: rafapelle@gmail.com

Questionário para professores

Nome:

Área de Atuação/Nível de ensino (pode ser mais do que uma resposta)

- Ensino Fundamental 1
- Ensino Fundamental 2
- Ensino Médio
- Outro:

Área de Atuação/Disciplinas (pode ser mais do que uma resposta)

- Sala de Informática
- Língua Portuguesa
- Matemática
- Ciências (Biologia, Física, Química)
- Estudos sociais (História, Geografia)
- Inglês

- Educação Física
- Artes
- Outra:

Você já ensinou computação/programação nas suas disciplinas? *
- Nunca
- 1 – 2 vezes
- 3-5 vezes
- Mais de 5 vezes

Qual ambiente de programação você usou no ensino de computação?
- App Inventor
- Scratch
- Snap!
- Outro:

Qual Sistema operacional você utilizou agora no teste do CodeMaster? *
- Linux
- Windows
- Mac
- Outro

Qual Navegador web você utilizou agora no teste do CodeMaster? *
- Mozilla Firefox
- Google Chrome
- Safari
- Microsoft Edge
- Outro

Quais projetos xml você usou no teste?
- Os 10 projetos disponibilizados por nós
- Projetos criados pelos meus alunos

Você acha a ferramenta CodeMaster útil no ensino de computação no ensino básico? *
- Sim
- Não

Explique, porque:

Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta? (Atualmente a ferramenta analisa: lógica, paralelismo, interatividade com o usuário, representação de dados, controle de fluxo, sincronização, abstração e operadores) *
- Sim
- Não

Se sim, quais?

Você acha que existem aspectos relevantes no processo de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta CodeMaster? *
- Sim
- Não

Se sim, quais?

Você acha que as informações disponibilizadas como resultado da avaliação são suficientes? *
- Sim
- Não

Se não, quais deveriam ser adicionadas?

Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) a ferramenta CodeMaster é prática no seu dia-a-dia?
- Sim
- Não

Se não, o que deverá ser diferente?

Você acha que a ferramenta CodeMaster possui elementos ambíguos ou difíceis de entender?*

- Sim
- Não

Se sim, quais?

A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória? *

- Sim
- Não

Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?

- Sim
- Não

Se sim, qual(is)?

Você achou fácil usar a ferramenta CodeMaster? *

- Sim
- Não

Se não, o que achou difícil?

Você observou algum erro (de ortografia/gramática) na interface da ferramenta CodeMaster?*

- Sim
- Não

Se sim, qual?

Escala de resposta para cada um dos 10 itens: 1 (discordo totalmente), 2 , 3, 4, 5 (concordo completamente)

Item	Afirmação	discordo totalmente 1	2	3	4	concordo completamente 5
1	Eu penso que usarei a ferramenta CodeMaster com frequência.					
2	Acho a ferramenta CodeMaster desnecessariamente complexa.					
3	Penso que a ferramenta CodeMaster é fácil de usar.					
4	Acho que vou precisar da ajuda de um técnico para usar a ferramenta CodeMaster.					
5	Acho as funções da ferramenta CodeMaster bem integradas.					
6	Encontro muitas inconsistências na ferramenta CodeMaster.					
7	Imagino que as pessoas aprenderão rapidamente a usar a ferramenta CodeMaster.					

8	Não acho a ferramenta CodeMaster prática de usar.					
9	Senti-me confiante ao usar a ferramenta CodeMaster.					
10	Precisei aprender muitas coisas antes de ser capaz de operar a ferramenta CodeMaster.					

O que mais você gostou da ferramenta CodeMaster? *

Alguma sugestão de melhoria referente a ferramenta CodeMaster?*

Mais algum comentário?

Questionário para alunos

Nome:

Você está em que ano do ensino fundamental *

- 1° ano
- 2° ano
- 3° ano
- 4° ano
- 5° ano
- 6° ano
- 7° ano
- 8° ano
- 9° ano
- Outro:

Você já programou um programa de software? *

- Nunca
- 1 – 2 vezes
- 3-5 vezes
- Mais de 5 vezes

Qual ambiente de programação você usou para programar? (Pode ser mais de uma resposta)
- App Inventor
- Scratch
- Snap!
- Outro:

Qual Sistema operacional você utilizou agora no teste do CodeMaster? *

- Linux
- Windows
- Mac
- Outro

Qual Navegador web você utilizou agora no teste do CodeMaster? *

- Mozilla Firefox
- Google Chrome
- Safari
- Microsoft Edge
- Outro

Você acha a ferramenta CodeMaster útil na aprendizagem de programação? *
- Sim
- Não

Explique, porque:

Você achou a nota justa? *
- Sim
- Não

Se não, porquê?

Você acha que as informações disponibilizadas como resultado da avaliação são suficientes? *
- Sim
- Não

Se não, quais mais deveriam ser adicionadas?

Você acha que a ferramenta CodeMaster possui elementos ambíguos ou difíceis de entender? *
- Sim
- Não

Se sim, quais?

A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória? *
- Sim
- Não

Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?
- Sim
- Não *

Se sim, qual(is)?

Você achou fácil usar a ferramenta CodeMaster? *
- Sim
- Não

Se não, o que achou difícil?

Você observou algum erro (de ortografia/gramática) na interface da ferramenta CodeMaster? *
- Sim
- Não

Se sim, qual?

Escala de resposta para cada um dos 10 itens: 1 (discordo totalmente), 2 , 3, 4, 5 (concordo completamente)

Item	**Afirmação**	discordo totalmente **1**	**2**	**3**	**4**	concordo completamente **5**
1	Eu penso que usarei a ferramenta CodeMaster com frequência.					

2	Acho a ferramenta CodeMaster desnecessariamente complexa.					
3	Penso que a ferramenta CodeMaster é fácil de usar.					
4	Acho que vou precisar da ajuda de um técnico para usar a ferramenta CodeMaster.					
5	Acho as funções da ferramenta CodeMaster bem integradas.					
6	Encontro muitas inconsistências na ferramenta CodeMaster.					
7	Imagino que as pessoas aprenderão rapidamente a usar a ferramenta CodeMaster.					
8	Não acho a ferramenta CodeMaster prática de usar.					
9	Senti-me confiante ao usar a ferramenta CodeMaster.					
10	Precisei aprender muitas coisas antes de ser capaz de operar a ferramenta CodeMaster.					

O que mais você gostou da ferramenta CodeMaster?

Alguma sugestão de melhoria referente a ferramenta CodeMaster?

Mais algum comentário?

Resultado da avaliação com usuários

Pergunta	Resposta		
	Sim	Não	Comentários
Você acha a ferramenta CodeMaster útil no ensino de computação no ensino básico?	6/6 (100%)	0/6 (0%)	- “Pode ser atrativo para os estudantes.” - “A ferramenta em questão é muito útil. Nos permite avaliar o nível de conhecimentos que a turma está, e propor atividades de programação nesta direção.” - “Espera-se também que o interesse pela utilização de tecnologias pode motivar a aprendizagem dos conteúdos disciplinares.”
Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?	1/3 (33,3%)	2/3 (66,7%)	- “Na minha atividade o interesse maior recai sobre a “jogabilidade” dos produtos desenvolvidos pelos alunos.” - “É preciso utilizar a ferramenta na prática para perceber se existem outros aspectos/critérios para serem inseridos.”
Você acha que existem aspectos relevantes no processo de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta CodeMaster?	1/3 (33,3%)	2/3 (66,7%)	
Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	6/6 (100%)	0/6 (0%)	
Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) a ferramenta CodeMaster é prática no seu dia-a-dia?	2/3 (66,7%)	1/3 (33,3%)	- “Criar uma galeria da própria plataforma.” (CodeMaster).
Você acha que a ferramenta CodeMaster possui elementos ambíguos ou difíceis de entender?	1/6 (16,7%)	5/6 (83,3%)	- “Se os critérios desta linguagem que professores que não sejam de computação não conhecem estiverem explicados na plataforma acredito que os elementos sejam compreensíveis.” - “Para mim não, porem para uma criança que está aprendendo há alguns comandos espalhados que seriam de difícil entendimento.”

A performance (tempo de resposta) da ferramenta CodeMaster é satisfatória?	6/6 (100%)	0/6 (0%)	
Você observou algum erro (bug) em relação a funcionalidade da ferramenta CodeMaster?	0/6 (0%)	6/6 (100%)	
Você achou fácil usar a ferramenta CodeMaster?	5/6 (83,3%)	1/6 (16,7%)	- “A maioria é fácil, porem acho ruim a ferramenta de criar as formas. “
Você achou a nota justa?	3/3 (100%)	0/3 (0%)	

Anexo C – Artigo sobre o Trabalho de Conclusão de Curso

Desenvolvimento de um analisador de código para Snap! voltado ao ensino de Computação na Educação Básica

Rafael Pelle

Departamento de Informática e Estatística, Universidade Federal de Santa Catarina

Florianópolis – SC – Brasil

rafapelle@gmail.com

Resumo. *O ensino de computação na Educação Básica, utilizando ambientes de desenvolvimento como Snap! auxilia os alunos a aprender competências básicas de programação e pensamento computacional. Uma das principais etapas dentro de uma unidade instrucional de ensino de computação é a avaliação da aprendizagem do aluno. Tipicamente esta tarefa requer um esforço considerável, além da necessidade da capacitação do professor do Ensino Básico. Nesse contexto, o objetivo deste trabalho é o desenvolvimento de uma ferramenta web para automatizar o processo de avaliação da aprendizagem por meio da análise do código de projetos desenvolvidos com a linguagem de programação Snap!.*

1. Introdução

Apesar da importância da computação e ótimas perspectivas de carreira, os cursos de graduação na área de Tecnologia da Informação (TI) ainda têm baixa concorrência por vagas e altas taxas de desistência. Em 2010, a evasão nos cursos da área de TI foi de 87% em todo o Brasil (BRASSCOM, 2012). É inevitável que isso se reflita no mercado de trabalho. O déficit de 408 mil profissionais de TI estimado para 2022 pode significar uma perda de receita de R$ 167 bilhões entre 2010 e 2020 para o setor (SOFTEX, 2014). Assim, existe uma necessidade urgente de melhorar o nível de compreensão pública de da computação como um campo acadêmico e profissional (CSTA, 2011).

Um dos objetivos do ensino dos conceitos de computação no Ensino Básico é popularizá-la e motivar os alunos a seguirem carreira na área, melhorando assim a situação atual do mercado de trabalho. O ensino destes princípios e práticas tem benefícios que vão além de uma formação profissional. Estudantes de computação aprendem raciocínio lógico, pensamento algorítmico, concepção e resolução de problemas estruturados, todos esses conceitos e habilidades são valiosos muito além da sala de aula de ciências da computação (CSTA, 2011). Estudar computação pode preparar um estudante para uma carreira em muitas áreas, tanto dentro quanto fora da computação (CSTA, 2011).

Uma alternativa para ensinar computação na Educação Básica é o uso de linguagens de programação visual, baseadas em blocos como a linguagem Snap! (SNAP!; BERKELEY, 2013). Com Snap! é possível criar jogos e animações, usando laços de repetição, execução condicional, eventos, operadores matemáticos e *booleanos*, listas entre outros recursos. Snap! é uma linguagem de programação amplamente convidativa para crianças e adultos e também uma plataforma para estudar computação (SNAP!; BERKELEY, 2013).

Uma questão que dificulta a inserção do ensino de computação é a falta de tempo dos professores para preparação e acompanhamento das aulas, além das suas outras responsabilidades. Também se observa dificuldade especificamente na avaliação da aprendizagem dos alunos, parte essencial do ensino. Corrigir e avaliar os trabalhos desenvolvidos pelos alunos é uma tarefa árdua, além de despender muito tempo se for realizada manualmente.

Nesse contexto, este trabalho visa facilitar a avaliação da aprendizagem dos alunos em relação a projetos de programação, desenvolvendo uma ferramenta web para automatizar o processo de avaliação da aprendizagem por meio da análise do código de projetos desenvolvidos com a linguagem de programação Snap!, a ser utilizada em unidades instrucionais no ensino de computação no Ensino Básico. Espera-se que a ferramenta aumente as chances de o ensino de computação ser adotado mais amplamente nas escolas.

2. Fundamentação teórica

Neste capítulo são apresentados conceitos sobre a teoria de ensino e aprendizagem, design instrucional e avaliação da aprendizagem. São abordados tópicos sobre ensino de computação no Ensino Básico. É apresentada uma visão geral sobre a linguagem de programação Snap!.

2.1. Aprendizagem e Ensino

Formalmente, a aprendizagem é definida como “Processo por meio do qual uma nova competência é incorporada à estrutura cognitiva do indivíduo” (MICHAELIS, 2016). Essas competências podem ser entendidas como a união entre conhecimentos, habilidades e atitudes Novas competências podem ser adquiridas por meio de estudo, experiência, raciocínio e observação e pode ser transmitida de forma sistemática. Neste contexto, o ensino sistematiza a aprendizagem de novas competências. (MICHAELIS, 2016).

O ensino pode ser, então, definido como uma atividade sistemática de interação entre seres, interação esta que se configura numa ação exercida sobre o sujeito ou grupo de sujeitos visando provocar neles mudanças tão eficazes que os tornem elementos ativos desta própria ação exercida (LIBÂNEO, 2009). Presume-se, a interligação de três elementos (Figura 3): um agente (professor, alguém, um grupo, etc.), uma competência transmitida (conhecimentos, atitudes e habilidades) e um educando (aluno, grupo de alunos, uma geração) (LIBÂNEO, 2009).

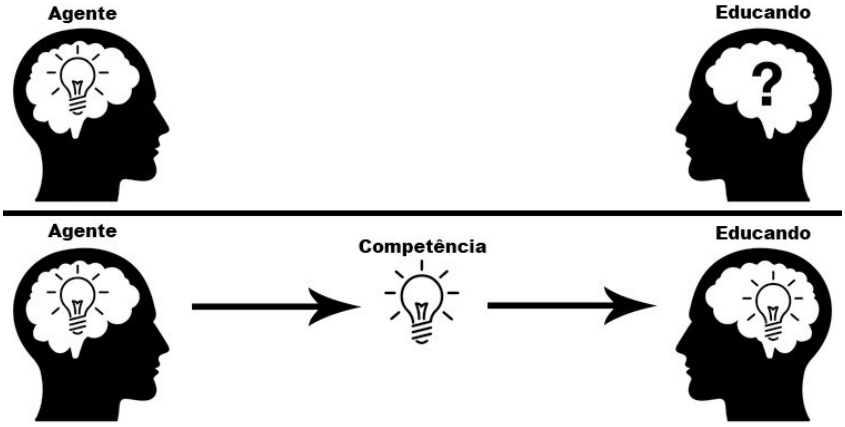


Figure 1 - Estrutura sistemática do Ensino

2.2. Design Instrucional

Design Instrucional é se refere à engenharia pedagógica (BASQUE, 2010) e trata-se de uma metodologia para fazer com que a transmissão de competências seja eficiente, efetiva e motivadora (MERRILL et al., 1996). O Design Instrucional corresponde à: ação intencional e sistemática de ensino, que envolve o planejamento, o desenvolvimento e a utilização de métodos, técnicas, atividades, materiais, eventos e produtos instrucionais em situações de ensino específicas, a fim de facilitar a aprendizagem humana a partir dos princípios de aprendizagem e instrução conhecidos (FILATRO, 2004). Design instrucional é um processo iterativo para planejar os objetivos de desempenho, selecionar estratégias instrucionais, escolher mídia, selecionar e/ou criar materiais e aplicar avaliação (BRANCH, 2009).

Um dos principais modelos de design instrucional é o modelo ADDIE[2] (Figura 2).

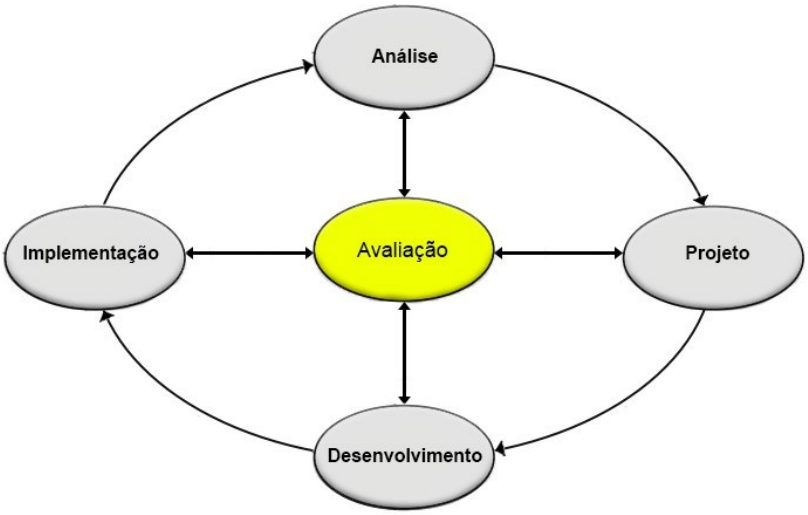


Figure 2 - Modelo ADDIE (BRANCH, 2009)

[2] O nome ADDIE é um acrônimo para as palavras: ***A**nalyze, **D**esign, **D**evelop, **I**mplement e **E**valuate* (Análise, Projeto, Desenvolvimento, Implementação e Avaliação) que representam cada uma das cinco fases do modelo.

2.3. Avaliação da aprendizagem

A avaliação da aprendizagem é uma tarefa fundamental no processo de ensino. Ela deve acompanhar todas as etapas do processo de ensino e aprendizagem. É por meio da avaliação da aprendizagem que é possível determinar se os alunos atingiram os objetivos de aprendizagem. A avaliação no design instrucional tem foco na medição da capacidade do aluno de realizar a sua nova competência (BRANCH, 2009).

A avaliação pode ser feita de forma formativa ou somativa. A avaliação formativa deve ser interpretada como todas as atividades empreendidas por professores e/ou por alunos, que forneçam *feedback* para modificar as atividades de aprendizagem e ensino (BLACK; WILIAM, 1998). É utilizada durante todo o processo de ensino e aprendizagem, avalia o nível de entrada do aluno e avalia se o aluno está progredindo adequadamente para atingir os objetivos de aprendizagem. A avaliação somativa é feita ao final de uma unidade instrucional. É projetada para avaliar a aprendizagem de um aluno em relação aos objetivos de aprendizagem da unidade instrucional, com o propósito de dar nota, certificação ou avaliação do progresso (BLACK; WILIAM, 1996). A avaliação somativa é comumente aplicada usando provas, trabalhos práticos, exercícios ou apresentações.

2.4. Ensino de computação no Ensino Básico

Existem no mundo diversos currículos de referência para o ensino de computação no Ensino Básico, entre eles um dos mais reconhecidos é o CSTA/ACM K-12 (CSTA, 2011). Conforme esse framework de currículo de referência os objetivos de qualquer curso da computação, no Ensino Básico, devem ser (CSTA, 2011):

- Introduzir os conceitos fundamentais da computação para todos os estudantes, a começar no Ensino Fundamental;
- Apresentar a computação no nível do ensino secundário de uma maneira que possa ser tanto acessível quanto digna de crédito no currículo escolar;
- Incentivar as escolas a oferecerem cursos de ciência da computação adicionais de nível secundário, o que permitirá aos estudantes interessados a possibilidade de estudar aspectos da ciência da computação com mais profundidade e prepará-los para a entrada no mercado de trabalho ou faculdade;
- Aumentar a disponibilidade de aprendizado da ciência da computação para todos os alunos, especialmente aqueles que pertencem às minorias.

Conforme este framework de currículo cinco áreas essenciais devem ser abordadas no ensino de computação no Ensino Básico (CSTA, 2011):

- **Programação:** trata-se de uma parte essencial da computação, é a competência de criar programas de software. Alunos devem aprender a projetar, desenvolver e publicar produtos (*websites*, aplicações móveis, animações e jogos) utilizando recursos tecnológicos. Eles devem compreender o que são algoritmos e qual a sua aplicação prática. Como parte da prática, também devem implementar software utilizando uma linguagem de programação.
- **Pensamento computacional:** é uma abordagem para a resolução de problemas de forma que pode ser implementada com um computador. Os alunos tornam-se não somente utilizadores de ferramentas, mas construtores de ferramentas. Os alunos fazem uso de um conjunto de conceitos, tais como abstração, recursão e iteração, para processar e analisar dados e criar artefatos reais e virtuais. O pensamento computacional é uma metodologia de resolução de problemas que pode ser automatizada, transferida e aplicada para diferentes áreas. A

relevância do pensamento computacional se dá pela facilidade de aplicá-lo a qualquer outro tipo de raciocínio.

- **Colaboração:** a computação é uma disciplina intrinsecamente colaborativa. Portanto, é importante que sejam desenvolvidas habilidades de colaboração, tais como trabalho em equipe, crítica construtiva e comunicação eficaz.
- **Computadores e dispositivos de comunicação:** os alunos devem compreender os elementos do computador moderno e de dispositivos e redes de comunicação. Os alunos devem usar a terminologia apropriada e precisa quando se comunicam acerca de tecnologia.
- **Impactos éticos globais e na comunidade:** princípios de privacidade, segurança de rede, licenças de software e direitos autorais devem ser ensinados a fim de preparar os alunos a se tornarem cidadãos responsáveis no mundo moderno. Os alunos também devem ser capazes de avaliar a confiabilidade e a precisão das informações na Internet. É essencial que os alunos compreendam o impacto dos computadores na comunicação internacional e identificar até que ponto os problemas de acesso impactam nossas vidas.

2.5. Ambiente de desenvolvimento Snap!

Uma alternativa para ensinar computação na Educação Básica é o uso de linguagens de programação visual baseada em blocos. Um exemplo é a linguagem Snap! (SNAP!; BERKELEY, 2013).). Com Snap! é possível criar jogos e animações, e visualizar sua execução. Para a criação destes pode-se usar laços de repetição, execução condicional, eventos, operadores matemáticos e *booleanos*, listas entre outros recursos.

O Snap! utiliza elementos característicos de ambientes/linguagens de programação baseada em blocos. A Tabela 1 apresenta os principais elementos e suas respectivas descrições.

Tabela 1 - Principais elementos da linguagem Snap!

Elemento	Tradução (Português)	Descrição
Sprite	Ator	Um objeto individual que pode mudar a aparência de acordo com as fantasias, emitir sons e executar outras ações em um palco.
Palette	Categoria	Uma divisão entre blocos para agrupá-los de forma que faça sentido.
Script	Roteiro	Um ou mais blocos que realizam ações como movimentar, alterar aparência de um ator, etc.
Stage	Palco	Onde um ou mais atores executam um ou mais roteiros. Os atores utilizam sons e fantasias que estão no Palco.
Block	Bloco	Uma instrução que pode ou não fazer parte de um roteiro.
Costumes	Fantasias	Imagens que um ator ou palco pode assumir para alterar sua aparência. O usuário pode usar fantasias predefinidas ou criar suas próprias fantasias.
Sounds	Sons	Sons que um ator ou palco pode emitir.
Sprite Corral	Curral	Local onde ficam todos os atores e palcos do projeto.

Um programa Snap! consiste de um ou mais roteiros. Cada roteiro é feito de um ou mais blocos de comando. A Figura 3 mostra um exemplo de roteiro de um programa Snap! que faz um ator se mover e desenhar um quadrado no palco, emitindo um som para cada aresta desenhada, quando a tecla “espaço” é pressionada.

Figure 3 - Exemplo de um roteiro Snap!

3. CodeMaster – Snap!

A proposta apresentada pelo presente trabalho é desenvolver uma ferramenta *web* semelhante à ferramenta Dr.Scratch, que automatize a análise e avaliação de projetos desenvolvidos com Snap!, a ser utilizada em unidades instrucionais para ensinar a computação no Ensino Básico – A CodeMaster. A ferramenta proposta deve atender as necessidades de dois tipos de usuários, alunos e professores. Os alunos, sem necessidade de cadastro, devem poder realizar *upload* de apenas um projeto por vez, a ferramenta deve analisá-lo, avaliá-lo e apresentar o resultado de forma completa. Os professores, previamente cadastrados, devem poder realizar upload de múltiplos projetos de uma vez, a ferramenta deve analisá-los, avaliá-los e apresentar os resultados de forma resumida, apenas uma visão geral das notas.

3.1 Modelo Conceitual

As principais funcionalidades da ferramenta são:

- Permitir o *upload* de projetos Snap!, em formato .XML, via interface web.
- Analisar projetos Snap! mapeando os *tokens* e suas respectivas frequências de uso.
- Avaliar projetos Snap! em relação aos conceitos: lógica, paralelismo, interatividade com o usuário, representação de dados, controle de fluxo, sincronização e abstração.
- Apresentar avaliação dos projetos Snap! para o professor. Incluindo em uma tabela as pontuações de cada conceito, pontuação total, nota e nível do projeto, de cada projeto, além das médias de cada conceito entre os projetos submetidos para avaliação.
- Apresentar avaliação de projeto Snap! para o aluno, de forma detalhada, incluindo as pontuações de cada conceito, pontuação total, nota e nível do projeto. A interface deve apresentar também sugestões de como melhorar a pontuação.

3.2. Processo de Análise e Avaliação

Neste subcapítulo o processo de análise e avaliação de código realizada na ferramenta CodeMaster é apresentado detalhadamente. A partir do *upload* de um projeto Snap! (arquivo

XML) a ferramenta CodeMaster analisa, avalia e apresenta os resultados da avaliação do projeto (Figura 4).

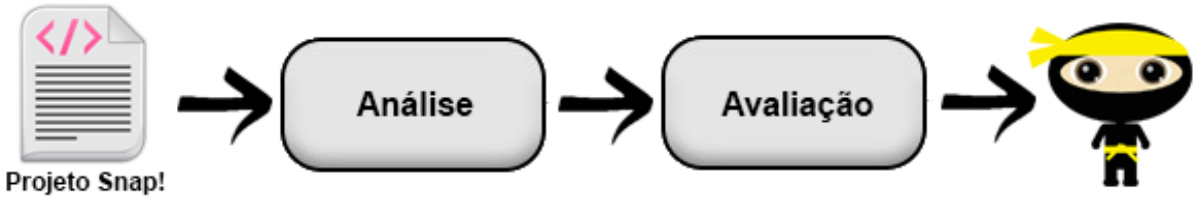


Figura 4 - Processo de análise e avaliação

3.2.1. Processo de análise de código

A ferramenta deve analisar projetos desenvolvidos com a linguagem de programação Snap! versão 4.0. Os projetos devem ser enviados em formato XML por meio de interface web. A análise deve ser realizada em três etapas:

4. O código do projeto, em XML, deve ser lido e convertido em uma *string*, para ser manipulado com maior facilidade.
5. A *string* resultante da etapa 1 deve ser dividida nas ocorrências dos caracteres “aspas”, “espaço em branco”, “nova linha”, “maior que”, “menor que”, resultando em uma lista de *tokens*.
6. A ferramenta deve percorrer a lista de *tokens*, resultado da etapa 2, realizando uma contagem da frequência uso de cada *token*, resultando em uma tabela de *tokens* e frequências de uso.

A Figura 5 exemplifica o processo de análise de um projeto Snap!.

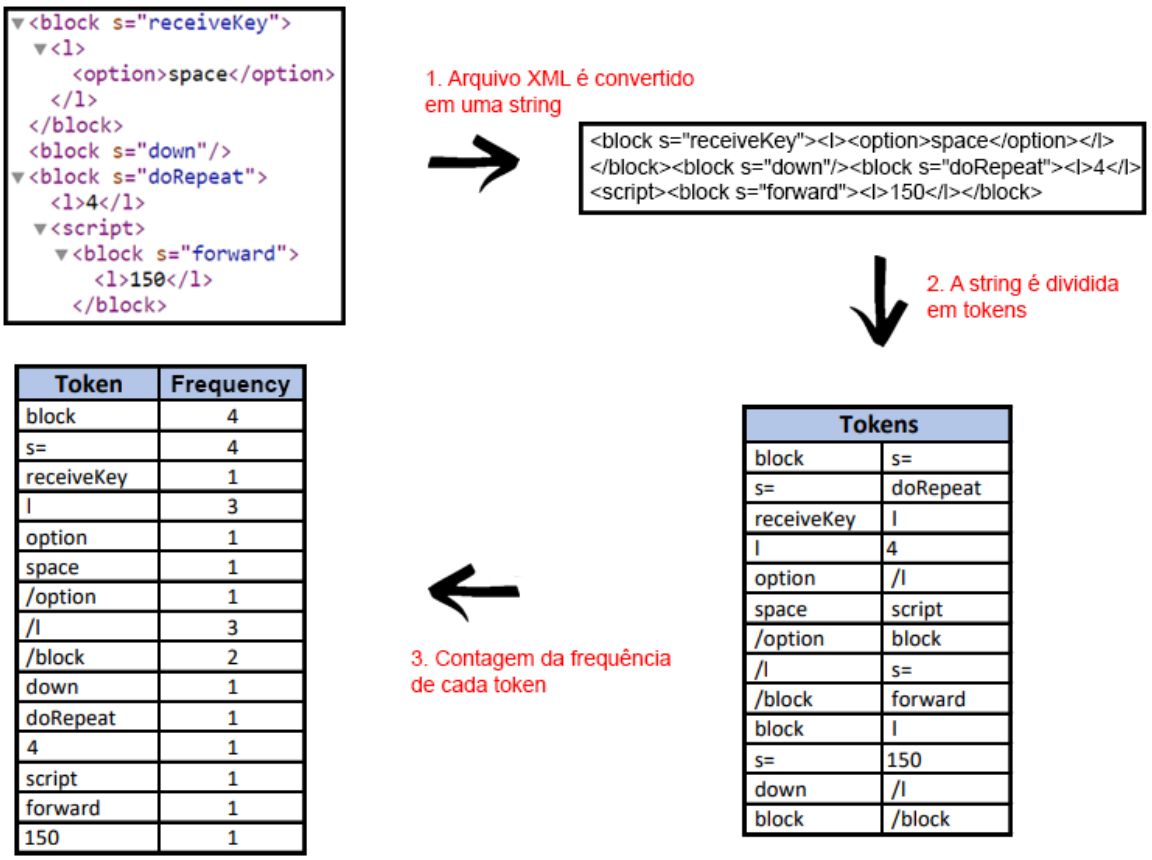


Figura 5 - Processo de análise de um projeto Snap!

3.2.2. Processo de avaliação

Com o objetivo de avaliar a aprendizagem do pensamento computacional, a ferramenta utiliza os dados extraídos do código para avaliar o projeto desenvolvido. Esta avaliação é feita em relação às dimensões-chave do Framework de pensamento computacional apresentado por Brennan & Resnick (2012): conceitos de pensamento computacional, práticas de pensamento computacional e perspectivas de pensamento computacional (Tabela 2), que também foi adotado como base por Dr.Scratch (Moreno-León; Robles, 2015).

Tabela 2 - Dimensões-chave de avaliação

***Framework* de pensamento computacional (BRENNAN; RESNICK, 2012)**	**CodeMaster (Snap!)**
Conceitos de pensamento computacional	
Sequencias	Controle de fluxo
Laços de repetição	Controle de fluxo
Eventos	Interatividade com o usuário
Paralelismo	Paralelismo
Condicionais	Lógica
Operadores	Operadores
Dados	Representação de dados
	Sincronização
Práticas de pensamento computacional	
Ser incremental e iterativo	
Testes e depuração	
Reutilização e remixagem	
Abstração e modularização	Abstração
Perspectivas de pensamento computacional	
Expressando	
Conectando	
Questionando	

A partir destas dimensões chave é definida uma rubrica como base para a ferramenta CodeMaster realizar avaliação de projetos Snap! (Tabela 3).

Tabela 3 - Rubrica de avaliação da ferramenta CodeMaster (Snap!)

Critérios	**Nível de performance**			
	0	**1**	**2**	**3**
Lógica	Não utilizou nenhum comando de lógica.	Utilização do comando "Se, então".	Utilização do comando "Se, então; senão".	Utilização de comandos de operações lógicas que combinam as condições.
Paralelismo	Não utilizou nenhum comando de paralelismo.	2 scripts iniciando com "bandeira verde".	2 scripts com o comando "quando a tecla for pressionada" utilizando a mesma tecla ou 2 scripts com o comando "quando eu for clicado".	2 scripts de recebimento de mensagens ou criação de clones ou 2 scripts de sensores.
Interatividade com o usuário	Não utilizou nenhum comando de interatividade com o usuário.	Utilização do comando da "bandeira verde".	Utilização de comandos de teclas/atores pressionadas, perguntas ao usuário, e botão do mouse.	Utilização de comandos de reprodução de sons.
Representação de dados	Não utilizou nenhum comando de representação de dados.	Modificação de propriedades dos atores como coordenadas, tamanho e aparência.	Operações com variáveis.	Operações com listas.
Controle de fluxo	Não utilizou nenhum comando de controle de fluxo.	Programação de uma sequência de blocos.	Utilização dos comandos "repita x vezes" e "sempre".	Utilização do comando "repita até que".
Sincronização	Não utilizou nenhum comando de sincronização.	Utilização do comando "espere".	Utilização de comandos de envio de mensagens com duração de tempo.	Utilização do comando "espere até".
Abstração	Não utilizou nenhum comando de abstração.	Programação com mais de 1 *script*.	Utilização e programação de funções por meio de blocos personalizados.	Utilização de clones.

Operadores	Não utilizou nenhum tipo de operador.	Utilização de operador de um tipo.	Utilização de operadores de dois tipos diferentes.	Utilização de mais do que dois tipos de operadores diferentes.

Referente ao critério de operadores, a Tabela 4 detalha os tipos de operadores.

Tabela 4 - Tipos de operadores

Tipos	Operadores
Comparações	=, <, >.
Operações matemáticas básicas	+, -, *, /.
Operações matemáticas avançadas	*mod, round, sqrt* e *pick random.*
Operações com *strings*.	*join, split by, letter of, length of, is identical to, unicode of* e *unicode as letter.*

O resultado esperado de uma avaliação inclui uma pontuação para cada conceito avaliado, uma pontuação total, uma nota e um nível de expertise. Ao avaliar um projeto a ferramenta utiliza a rubrica apresentada na Tabela 3 e a tabela de *tokens* e frequências de uso resultante do processo de análise para atribuir **pontuações individuais** para cada conceito. A **pontuação total** de um projeto Snap! é calculada somando cada uma das pontuações individuais.

$$PontuaçãoTotal = \sum PontuaçãoIndividualDeCadaConceito$$

A maior pontuação possível, chamada de **pontuação máxima**, é calculada multiplicando a quantidade de conceitos avaliados (por padrão oito) pela pontuação máxima de cada conceito (três).

$$PontuaçãoMáxima = QuantidadeConceitosAvaliados * 3$$

No Brasil não há uma escala determinada para avaliação dos alunos no Ensino Básico, porém, observa-se que a escala entre 0 e 10 é amplamente utilizada. Visando facilitar o entendimento do professor e do aluno em relação à avaliação de um projeto, a pontuação total deve ser mapeada em uma nota, que pode variar entre 0 e 10. A nota de um projeto é calculada da seguinte forma:

$$Nota = \left(\frac{PontuaçãoTotal}{PontuaçãoMáxima}\right) * 10$$

Considerando a aplicação da ferramenta CodeMaster no contexto educacional com crianças e/ou jovens, o resultado é apresentado também de forma mais lúdica adotando uma gamificação. O termo "gamificação" refere-se ao uso de elementos baseados em jogos em contextos não relacionados a jogos, que visam envolver as pessoas, motivá-las, melhorar a aprendizagem e resolver problemas (DETERDING; DIXON; KHALED; NACKE, 2011). Visando esses

benefícios, a ferramenta atribui e apresenta ao usuário um **nível de expertise** gamificado do projeto, além das pontuações e nota. Para isto são utilizadas imagens de Ninjas com faixas de diferentes cores (Figura 6).

Figura 6 - Ninjas do CodeMaster

A cor da faixa utilizada pelo ninja é atribuída com base na nota, como mostra a Tabela 5.

Tabela 5 - Mapeamento entre nota e nível de expertise

Nota	Nível de expertise
De 0,0 até 0,9	Faixa branca
De 1,0 até 1,9	Faixa amarela
De 2,0 até 2,9	Faixa laranja
De 3,0 até 3,9	Faixa vermelha
De 4,0 até 4,9	Faixa roxa
De 5,0 até 5,9	Faixa azul
De 6,0 até 6,9	Faixa turquesa
De 7,0 até 7,9	Faixa verde
De 8,0 até 8,9	Faixa marrom
De 9,0 até 10	Faixa preta

A Figura 7 exemplifica o processo de avaliação de um projeto Snap!.

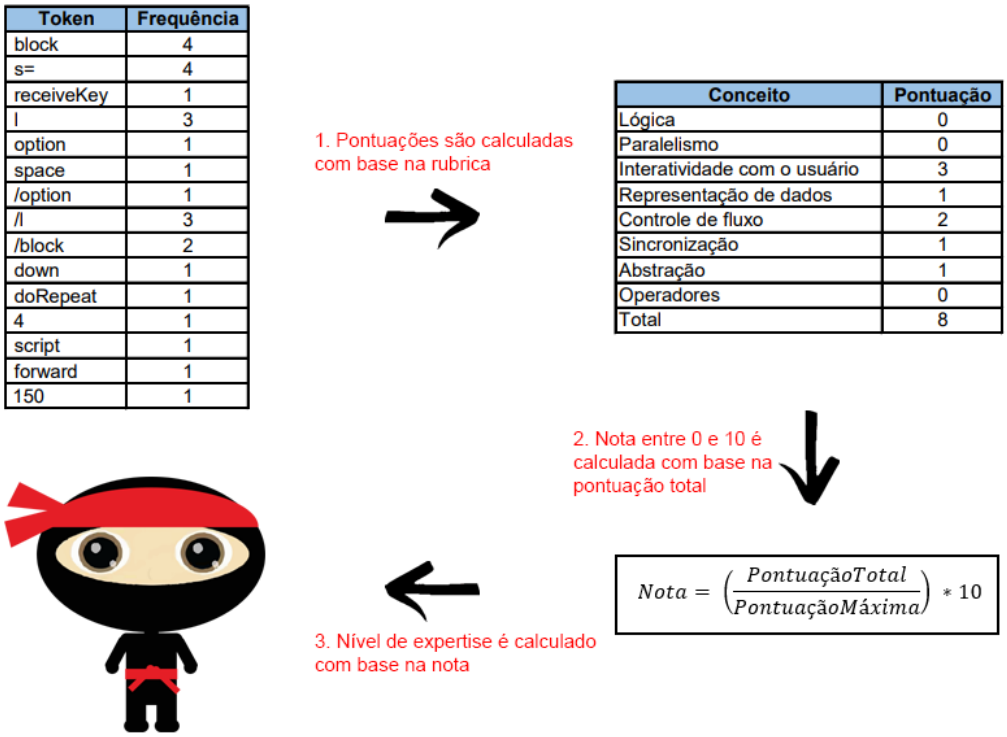


Figura 7 – Processo de avaliação de um projeto Snap!

3.3 Casos de uso

Atores do sistema são usuários e elementos computacionais que interagem com o sistema. Os atores da ferramenta proposta pelo presente trabalho são apresentados na Tabela 6.

Tabela 6 – Atores do sistema

Ator	Descrição
A01 – Aluno	Envia projeto Snap! para avaliação e vê o resultado da avaliação.
A02 – Professor	Envia um conjunto de projetos para avaliação e vê o resultado das avaliações. Também acessa avaliações de projetos que enviou no previamente.
A03 – Pesquisador	Acessa as estatísticas de todas as avaliações presentes no banco de dados.

Analisando o modelo conceitual da seção 4.1, os requisitos da seção 4.2 e os atores do sistema apresentados na Tabela 19, foram elaborados os casos de uso do sistema. A Figura 8 apresenta o diagrama de Casos de Uso da ferramenta CodeMaster (Snap!).

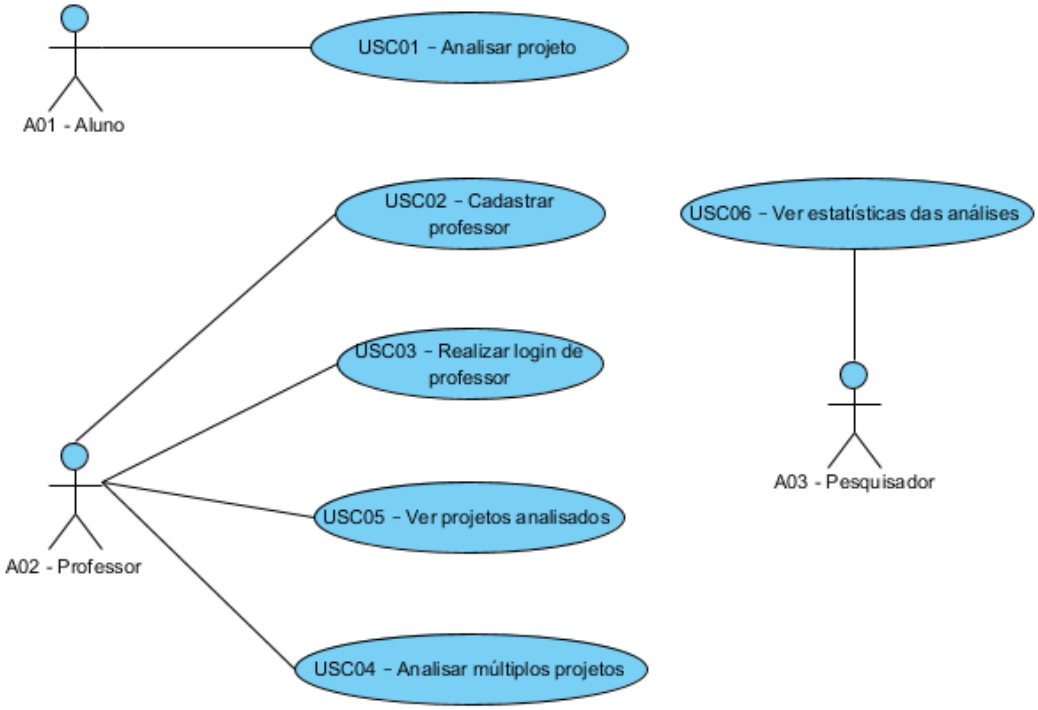


Figura 8 - Diagrama de Casos de Uso

Caso de Uso 1 – USC01 Analisar projeto

Atores:

- A01 – Aluno

Fluxo principal:

7. Aluno acessa o site do CodeMaster.
8. Aluno clica no botão “escolher arquivo”.
9. Aluno escolhe arquivo de projeto Snap! em seu computador.
10. Aluno clica no botão “avaliar”.
11. Sistema realiza a análise/avaliação do código do projeto.
12. Sistema retorna o resultado da análise do projeto.

Fluxo de exceção:

5a. Aluno clica no botão “avaliar” sem escolher projetos. Sistema informa uma mensagem de erro: “Nenhum projeto foi selecionado para análise!”.

Caso de Uso 2 – USC02 Cadastrar professor

Atores:

- A02 – Professor

Fluxo Principal:

7. Professor acessa site do CodeMaster.
8. Professor clica no menu superior no botão “professores”.
9. Professor clica na opção “Ainda não sou cadastrado” abaixo dos campos de login e senha.

10. Professor informa, nome completo, e-mail, instituição de ensino, cidade, estado, país, senha, confirmação de senha e marca o *checkbox* “Li e aceito os termos de contrato”.
11. Professor clica no botão “cadastrar”.
12. Professor é redirecionado para a página principal dos professores.

Fluxo de exceção:

5a. Professor não preenche um dos campos do formulário de cadastro e o sistema apresenta a mensagem “Todos os campos devem ser preenchidos”.

6b. Professor não marca o *checkbox* “Li e aceito os termos de contrato” e o sistema apresenta a mensagem “Você deve ler e aceitar os termos de contrato”.

7b. E-mail informado já existe cadastrado, sistema apresenta a mensagem “Já existe uma conta com este e-mail, verifique o e-mail informado!”.

Caso de Uso 3 – USC03 Login de professor

Atores:

- A02 – Professor

Pré-condição:

- USC02

Fluxo Principal:

7. Professor acessa o site do CodeMaster.
8. Professor clica no botão “professores” no menu superior da tela.
9. Professor informa seu e-mail e senha nos campos indicados.
10. Professor clica no botão “login” e seu e-mail e senha estão corretos.
11. O sistema autentica o professor no sistema.
12. Professor é redirecionado para a página principal dos professores.

Fluxo de exceção:

6a. O e-mail e senha do professor estão incorretos. O sistema informa uma mensagem de erro: “Usuário e senha estão incorretos”.

Caso de Uso 4 – USC04 Analisar múltiplos projetos

Atores:

- A02 – Professor

Pré-condição:

- USC03

Fluxo Principal:

6. Professor seleciona o tipo de projeto (App inventor ou Snap!) e nome da turma.
7. Professor seleciona os conceitos que serão avaliados.
8. Professor escolhe múltiplos arquivos de seu computador que correspondem a uma turma.
9. Professor clica no botão “avaliar”.
10. Professor recebe o resultado da análise de todos os projetos em uma tabela.

Fluxo de exceção:

4a. Professor clica em avaliar sem escolher projetos. Sistema informa uma mensagem de erro: “Nenhum projeto foi selecionado para análise!”.

Caso de Uso 5 – USC05 Ver projetos analisados

Atores:

- A02 - Professor

Pré-condição:

- USC03

Fluxo Principal:

5. Professor clica em ver projetos avaliados no menu lateral da área de professores.
6. Professor seleciona a turma que deseja ver as análises.
7. Sistema consulta e Banco de Dados e retorna o resultado das análises.
8. Professor visualiza as análises de todos os projetos daquela turma em forma de tabela.

Fluxo de exceção:

Nenhum projeto foi enviado para análise anteriormente pelo professor. Sistema apresenta mensagem: “Nenhum resultado foi encontrado”.

4. Conclusão

O objetivo geral do presente trabalho foi desenvolver uma ferramenta web para analisar e avaliar automaticamente código desenvolvido com a linguagem de programação para iniciantes Snap!, a ser utilizada em unidades instrucionais no ensino de computação no Ensino Básico. Neste contexto foi feita a análise da fundamentação teórica sobre aprendizagem, ensino de computação no Ensino Básico, sobre a ferramenta de desenvolvimento Snap! e sobre análise de código (O1). Em seguida foi realizado um estudo de mapeamento da literatura e identificado que existem poucos analisadores de código de linguagens de programação visual baseadas em blocos (O2).

Desta forma, com base na fundamentação teórica e na análise do estado da arte foi desenvolvida uma ferramenta web para análise e avaliação automatizada de projetos Snap!, nomeada de CodeMaster – Snap! (O3). A ferramenta proposta atende as necessidades de dois tipos de usuários, alunos e professores. Os alunos, sem necessidade de cadastro, podem realizar *upload* de apenas um projeto por vez, a ferramenta realiza a análise, avaliação e apresenta o resultado de forma completa. Os professores, previamente cadastrados, podem realizar upload de múltiplos projetos de uma vez, a ferramenta realiza a análise, avaliação e apresenta os resultados de forma resumida, apenas uma visão geral das notas.

Posteriormente, a ferramenta CodeMaster foi avaliada (O4). Avaliações com usuários, para obter dados sobre eficácia das tarefas bem como sobre a qualidade percebida do ponto de vista de professores e alunos e testes de corretude comparando o resultado de avaliações feitas com a ferramenta CodeMaster com o resultado de avaliações manuais. As avaliações sobre sua utilidade no ensino de computação no ensino básico, seu *feedback* apresentado como resultado das avaliações, a clareza de seus elementos, sua performance e sua facilidade de utilização foram muito positivas. Nos testes de corretude a ferramenta também foi bem avaliada, apenas uma falha foi detectada e corrigida.

Espera-se que a automatização do processo de análise e avaliação de projetos facilite a avaliação da aprendizagem dos alunos em relação a projetos de programação desenvolvidos com Snap!, reduzindo o tempo despendido e esforço dos professores, aumentando as chances de o ensino de computação ser adotado mais amplamente nas escolas.

Como trabalho futuro é sugerida a adição de um novo critério de avaliação à ferramenta, para avaliação de jogos desenvolvidos com Snap!, além da criação de uma galeria de projetos, para que professores e alunos possam compartilhar projetos, facilitando a avaliação e o recebimento dos projetos por parte do professor. Também é sugerida a alteração do sistema para bloquear o *upload* de arquivos com formatos inválidos (diferentes de .xml ou .aia). A expansão do sistema para permitir que pesquisadores realizem avaliação de projetos e recebam um feedback mais detalhado (p. ex. quais blocos foram programados, más práticas de programação, etc.) também é sugerida.

Referências

AULD, R., BELFIORE, P., & SCHEELER, M. (2010). Increasing pre-service teachers' use of differential reinforcement: Effects of performance feedback on. Journal of Behavioral Education, 19(1), pp. 169–183.

BALL, M. λ - An Autograder for Snap!, 2016. Disponível em:<https://www.gitbook.com/book/cycomachead/thesis/details>. Acesso em: Março 2017.

BASQUE, J. Ingénierie Pédagogique Et Technologies Éducatives. Québec: Télé-université, 2010.

BILAL, M., CHAN, P., MEDDINGS, F., & KONSTADOPOULOU, A. (2012). SCORE: An advanced assessment and feedback framework with a universal marking scheme in higher education. In Proc. of the Int. Conf. on Education and e-Learning Innovations, (pp. 1-6). Sousse/Tunísia

BINKLEY, D. 2007. Source Code Analysis: A Road Map. In *2007 Future of Software Engineering* (FOSE '07). IEEE Computer Society, Washington, DC, USA, 104-119. DOI=http://dx.doi.org/10.1109/FOSE.2007.27

BLACK, P.; WILIAM, D. (1996). Meanings and Consequences: A Basis for Distinguishing Formative and Summative Functions of Assessment? British Educational Research Journal, 22(5), 537-548.

BLACK, P.; WILIAN, D. (1998). Assessment and classroom learning. Assessment in Education: Principles, Policy & Practice, 5(1), pp. 7–74.

BOE, B.; HILL, C.; LEN, M.; Dreschler, G.; CONRAD, P.; Franklin, D. 2013. Hairball: lint-inspired static analysis of scratch projects. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (SIGCSE '13). Denver, USA.

BRANCH, R. M. Instructional Design: The ADDIE Approach. New York: Springer, 2009.

BRASSCOM. Índice BRASCOM de Convergência Digital, 2012. Disponível em: <www.brasscom.org.br/brasscom/Portugues/download.php?cod=437>. Acesso em: Setembro 2016.

BRENNAN, K.; RESNICK, M. (2012). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada (pp. 1-25).

BROOKE, J. et al. Sus-a quick and dirty usability scale. Usability evaluation in industry, London, v. 189, n. 194, p. 4–7, 1996.

CISCO. Networking Skills in Latin America, 2016. Disponível em: <http://www.cisco.com/assets/csr/pdf/IDC_Skills_Gap_-_LatAm.pdf>. Acesso em: Setembro 2016.

CSTA. ACM. CSTA K –12 Computer Science Standards, 2011. Disponível em: <https://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf>. Acesso em: Setembro 2016.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, p. 319–340, 1989.

DEMETRIO, M. F. (2017) Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação. 2017. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Department of Defense Handbook. 1988. *Instructional System Development/ System Approach to Training and Education (part 2 of 5)*. MIL-HDBK-29613-2A. 31 Oct 1988.

DETERDING, S.; DIXON, D.; KHALED, R.; NACKE, L. 2011. From game design elements to gamefulness: defining "gamification". In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland.

EMANUELSSON, P.; NILSSON, U. 2008. A comparative study of industrial static analysis tools. Electronic notes in theoretical computer science, v. 217, p. 5-21.

FILATRO, A. Design instrucional contextualizado: educação e tecnologia. São Paulo: SENAC, 2004.

GOHN, M. 2006. Educação não-formal, participação da sociedade civil e estruturas colegiadas nas escolas. Rio de Janeiro, Brasil. 14(50), p. 27-38.

GOMES, I.; MORGADO, P.; GOMES, T.; MOREIRA, R. 2009. An overview on the Static Code Analysis approach in Software Development. Faculdade de Engenharia da Universidade do Porto, Portugal. Disponível em: < https://web.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf>. Acesso em: Junho 2017.

HADDAWAY, N. R. et al. The role of Google Scholar in evidence reviews and its applicability to grey literature searching. PloS one, v. 10, n. 9, 2015.

HALLBERG, C. 2017. Comunicação pessoal.

HUANG, W. H. Y.; SOMAN, D. 2013. Gamification of education. Research Report Series: Behavioural Economics in Action, Rotman School of Management, University of Toronto. Disponível em: <https://pdfs.semanticscholar.org/c1df/e1970305f257b08a9f2b9844b346452eb869.pdf>. Acesso em: Junho 2017.

INEP. Sinopses Estatísticas da Educação Superior – Graduação. Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Brasília. 2010 a 2014.

ISO/IEC-25010 (2011). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

ISSO/IEC-9241-210:2010 - Ergonomicsof human-system interaction – Part 210: Human-centred design for interactive systems. 1. ed. [S.l.], 2010.

JAX-RS API, Java™ API for RESTful Web Services (JAX-RS) delivers API for RESTful Web Services development in Java SE and Java EE. 2017. Disponível em: < https://github.com/jax-rs>. Acesso em: julho de 2017.

JERSEY, RESTful Web Services in Java. 2017. Disponível em: < https://jersey.github.io/ >. Acesso em: julho de 2017.

JONES, T. Static and dynamic testing in the software development life cycle, 2013. Disponível em: <https://www.ibm.com/developerworks/library/se-static/se-static-pdf.pdf>. Acesso em: Novembro 2016.

Jonsson, A. Svingby, G. 2007. The use of scoring rubrics: Reliability, validity and educational consequences. Educational Research Review, 2(2), pp.130-144.

JOHNSON, D., & JOHNSON, R. (1993). Cooperative learning and feedback in technology-based instruction. In: J. DEMPSEY, Interactive instruction and feedback (pp. 133–157). Englewood Cliffs: Educational Technology Publications.

KITCHENHAM, B. 2004. Procedures for Performing Systematic Reviews. Joint Technical Report, TR/SE-0401 and NICTA 0400011T.1: Keele University, Keele, Reino Unido.

LIBÂNEO, J. C. 2009. Didática. 29 ed. Cortez. 264 p.

LIU, C. (2010). The comparison of learning effectiveness between traditional faceto-face learning and e-learning among goal-oriented users. In Proc. of the 6th Int. Conf. on Digital Content, Multimedia Technology and its Applications, Seoul/South Korea.

LYE, S. Y., KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 2014, 41, 51-61.

MERRILL, M. D. et al. Reclaiming instructional design. Educational Technology, 36(5), p. 5-7, 1996.

MICHAELIS. Dicionário Brasileiro da Língua Portuguesa. Disponível em: <http://michaelis.uol.com.br/>. Acesso em: Setembro 2016.

Moreno-León, J.; Robles, G. 2015. Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. In: Proceedings of the 10th Workshop in Primary and Secondary Computing Education, London, UK.

MYSQL, 2017. Disponível em: < https://www.mysql.com/>. Acesso em: agosto de 2017.

NATHALIERUN, Galerie de mes projets Snap!. Disponível em: <https://nathalierun.net/snap/Snap.Galerie/>. Acesso em: Outubro 2017.

NIELSON, F.; NIELSON, H. R.; HANKIN, C. Principles of Program Analysis. 2 ed. Berlin: Springer, 2015. 452 p.

OTA, G.; MORIMOTO, Y.; KATO, H. 2016. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Cambridge, UK.

PCN, Parâmetros Curriculares Nacionais, Terceiro e Quarto ciclos do Ensino Fundamental. Disponível em: <http://portal.mec.gov.br/seb/arquivos/pdf/introducao.pdf>. Acesso em: Outubro de 2016.

PETERSEN, K. et al. Systematic Mapping Studies in Software Engineering. In:

EASE. 2008, p. 68-77.

PRESSMAN, R. Engenharia de software: Uma abordagem profissional. 8 ed. Porto Alegre: Bookman, 2016. 968 p.

PROGRAMAMOS; DR.SCRATCH. Dr.Scratch – Analise seus projetos Scratch aqui. Disponivel em: <http://www.drscratch.org/>. Acesso em: Dezembro 2016.

RICHARDS, M., & SCHIFFEL, J. (2005). A distance learning framework for automatic instructor replies: articulable tacit knowledge used for feedback upon request. In Proceedings of the IEEE SoutheastCon, pp. 611-620, Lauderdale/USA.

SCRATCH; MIT. Scratch, 2013. Disponivel em: <http://scratch.mit.edu>. Acesso em: Novembro 2016.

SNAP!. Snap! Reference Manual Version 4.0, 2013. Disponível em: <http://snap.berkeley.edu/SnapManual.pdf>. Acesso em: Novembro 2016.

SNAP!; BERKELEY. Snap!, 2013. Disponível em: <http://snap.berkeley.edu>. Acesso em: Setembro 2016.

SOFTEX. Relatório Anual, 2014. Disponível em: <http://www.softex.br/wp-content/uploads/2015/04/Relatorio_Anual_2014.pdf>. Acesso em: Setembro 2016.

STRIEWE, M.; GOEDICKE, M. 2014. A Review of Static Analysis Approaches for Programming Exercises. In: International Computer Assisted Assessment Conference. Zeist, Holanda. Springer International Publishing. p. 100-113.

WAZLAWICK, R. Engenharia de software: conceitos e práticas. Elsevier Brasil, 2013.

WING, J. M. Computational thinking. Communications of the ACM, v. 49, n. 3, 2006.

WIRTH, N. Compiler Construction. 1 ed. Redwood City, USA: Addison-Wesley Pub (Sd), 1996. 131 p.

WOLZ, U.; HALLBERG, C.; TAYLOR, B. 2011. Scrape: A tool for visualizing the code of Scratch programs. In: Proceedings of 42nd ACM Technical Symposium on Computer Science Education, Dallas, USA. 2011.