UNIVERSIDADE FEDERAL DE SANTA CATARINA

MINESCRATCH: INTEGRAÇÃO MINETEST-SCRATCH PARA APOIAR O ENSINO DE PROGRAMAÇÃO

Jhonata da Rocha

Florianópolis – Santa Catarina

2016/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E

ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

Jhonata da Rocha

MINESCRATCH: INTEGRAÇÃO MINETEST-SCRATCH PARA APOIAR O ENSINO DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso para a obtenção do grau de Bacharelado no Curso de Sistemas de Informação na Universidade Federal de Santa Catarina.

Florianópolis – Santa Catarina

2016/2

Jhonata da Rocha

MINESCRATCH: INTEGRAÇÃO MINETEST-SCRATCH PARA APOIAR O ENSINO DE PROGRAMAÇÃO

Este Trabalho de Conclusão de Curso foi julgado para a obtenção do Título de “Bacharel em Sistemas de Informação”, e aprovado em sua forma final pelo Curso de Bacharelado em Sistemas de Informação.

Florianópolis – Santa Catarina, outubro de 2016.

_____

Prof. Dr. Renato Cislaghi
Coordenador

Banca Examinadora:

_____

Prof. Dr. Jean Carlo Rossa Hauck
Orientador

_____

Prof. Dra. rer. nat. Christiane Anneliese Gresse Von Wangenheim, PMP

_____

Giani Petri

Resumo

O ensino de programação para crianças e jovens tem se tornado um assunto cada vez mais importante, entretanto muitas escolas ainda não oferecem esse tipo de oportunidade aos estudantes. Diversas iniciativas têm sido criadas no intuito de estimular e apoiar o ensino de programação. O Scratch é uma dessas ferramentas que permite o ensino de programação de forma visual. Por outro lado, jogos como Minecraft são muito atrativos para crianças e jovens. Minetest é um jogo com propósito similar, porém com código livre. Nesse sentido, a utilização de jogos pode ser um importante aliado no ensino desse conteúdo. Pensando nisso, o objetivo deste trabalho é apresentar uma ferramenta para auxiliar a construção de jogos educacionais para o ensino de programação, utilizando o apelo visual disponível pelo software Minetest e a facilidade de codificação do Scratch. O trabalho objetiva ainda a construção de um protótipo de jogo educacional, utilizando a biblioteca construída. Uma integração de integração entre Minetest e Scratch é modelada, desenvolvida, testada e avaliada. Os primeiros resultados da avaliação indicam a viabilidade de utilização da biblioteca.

Palavras-chave: Biblioteca, Integração, Scratch, Minetest, Ensino de programação.

Abstract

Programming education for children and youth has become an increasingly important issue, however many schools do not offer this kind of opportunity to students. Several initiatives have been created in order to encourage and support educational programming. Scratch a such tool that allows visually programming education. On the other hand, games like Minecraft are very attractive to children and young people. Minetest is a game with a similar purpose, but with open source. In this sense, the use of games can be an important ally in teaching this content. Thinking about it, the aim of this study is to provide a tool to assist the construction of educational games for teaching programming, using the visual appeal available by the Minetest software and Scratch coding facility. The study aims to further the construction of a prototype educational game using the built library. An integration between Minetest and Scratch is modeled, developed, tested and evaluated. The first evaluation results indicate the feasibility of using the library.

Keywords: library, integration, Scratch, Minetest, programming education.

Lista de Figuras

Lista de Quadros

Sumário

1. Introdução

1.1 Contextualização

O Brasil possui o 5º maior mercado interno de tecnologia da informação segundo um estudo feito pela BRASSCOM (2013), tendo uma participação de 5,2% no PIB nacional. Entretanto, apesar de números positivos, um assunto é objeto de discussões: a carência de profissionais qualificados. Ainda segundo a BRASSCOM (2013), no ano de 2014 seriam requisitados 78 mil profissionais no setor de tecnologia nos estados de São Paulo, Rio de Janeiro, Bahia, Pernambuco, Rio Grande do Sul e Minas Gerais, mas apenas 33 mil estudantes teriam seus cursos concluídos para a ocupação de tais vagas.

Essa carência de profissionais qualificados tem relação com a alta taxa de evasão dos cursos de tecnologia. Segundo o observatório SOFTEX (2012), anualmente 21% dos estudantes matriculados abandonam seus cursos da área de TI. Uma possível justificativa para isso é o fato da maior parte das escolas não possuírem conteúdos relacionados a computação em sua grade de ensino, limitando-se, na maioria das vezes, a aulas de informática como edição de documentos de texto. Muitos alunos entram no curso de graduação sem possuir contato algum com o pensamento computacional, raciocínio lógico em programação ou mesmo conhecimentos básicos de computadores, o que pode tornar o aprendizado mais árduo e desestimular o estudante.

Uma hipótese para reduzir a escassez de profissionais na área de TI, o ensino de computação para crianças e jovens torna-se importante quando se percebe que atualmente a computação é onipresente. Por possuir muitos aspectos

interdisciplinares, sua compreensão pode ser relevante para profissionais de outras áreas.

Iniciativas como code.org[1] e Computação na Escola[2] surgiram para tentar levar o ensino de computação para crianças e jovens a um público maior. Existe também o Scratch[3], projeto criado no Instituto de Tecnologia de Massachusetts, que possui o intuito do ensino mais focado na programação, permitindo codificar por meio de blocos visuais, fazendo seu aprendizado mais intuitivo e atraente.

Além disso, considerando o mercado brasileiro de jogos digitais, que em 2012 teve aproximadamente 35 milhões de usuários (SEBRAE, 2012), é possível supor que uma forma de atrair usuários para o ensino de programação seria através do uso de jogos educacionais. Um jogo educacional é qualquer atividade com o intuito instrucional que envolva competição e que seja regulado por regras (Dempsey, Rasmussem e Luccasen,1996 apud. BOTELHO, 2004).

Um dos jogos digitais mais famosos é Minecraft[4], que em 2014 se tornou o terceiro jogo mais vendido da história[5]. Nele o jogador possui a liberdade de tomar suas próprias decisões, não havendo assim um objetivo único, e possuindo como principais características o estímulo a criatividade. É possível aproveitar-se do entusiasmo com o Minecraft para o ensino de programação. Contudo, existe um impasse: o Minecraft trata-se de um jogo proprietário, com custo de aquisição elevado para escolas públicas, por exemplo. Uma alternativa é a utilização de um jogo com

[1] https://code.org/
[2] http://www.computacaonaescola.ufsc.br/
[3] https://scratch.mit.edu/
[4] https://minecraft.net/pt-br/
[5] http://g1.globo.com/tecnologia/games/noticia/2014/06/minecraft-se-torna-o-3-game-mais-vendido-de-todos-os-tempos.html

proposta muito semelhante e que possui código aberto, permitindo evoluções e melhorias pela comunidade é o Minetest[6].

Assim, surge uma questão: estes dois softwares, Scratch e Minetest, poderiam ser utilizadas de forma integrada para apoiar o ensino de programação para crianças, aproveitando o apelo de jogabilidade do Minetest e o potencial de programação em blocos do Scratch? A integração é possível, entretanto não foi possível encontrar, atualmente, uma forma de trabalhar com as duas ferramentas em conjunto, pois elas utilizam protocolos de comunicação diferentes. Enquanto o Minetest trabalha com um protocolo de comunicação de baixo nível sobre UDP, controlando a sessão do jogo, movimentos, etc. o Scratch trabalha com um protocolo de mais alto nível, sobre HTTP, controlando os comandos possíveis.

Nesse sentido, este trabalho visa o desenvolvimento de uma integração o Minetest e o Scratch de forma a tornar possível o desenvolvimento de jogos educacionais a partir dessa integração.

1.2 Objetivos

Esta seção apresenta os objetivos geral e específicos do trabalho proposto.

1.2.1 Objetivo Geral

O objetivo geral do trabalho consiste em modelar, desenvolver e testar uma integração entre as ferramentas Scratch e Minetest, na forma de uma biblioteca, avaliada por meio de um protótipo de jogo educacional para ensino de programação.

[6] http://www.minetest.net/

1.2.2 Objetivos Específicos

Para o desenvolvimento desta integração são definidos os seguintes objetivos específicos:

- Levantar a fundamentação teórica sobre: bibliotecas e modularização de softwares, desenvolvimento de jogos educacionais, ensino de programação e as ferramentas a serem utilizadas neste trabalho.
- Realizar uma revisão sistemática da literatura sobre softwares integrando Minetest e Scratch para ensino de programação.
- Modelar e desenvolver uma biblioteca que faça a integração entre os protocolos de comunicação dos softwares Scratch e Minetest.
- Testar a biblioteca por meio do desenvolvimento de um jogo educacional que utiliza a biblioteca.
- Avaliar a biblioteca por meio do jogo desenvolvido.

1.3 Limites do escopo do trabalho

O presente trabalho limita-se a desenvolver uma biblioteca para a construção de jogos educacionais que faça a integração entre os protocolos de comunicação dos softwares Minetest e Scratch, e testá-la por meio da construção de um jogo educacional com intuito de ensino de programação.

O jogo educacional desenvolvido serve somente como prova de conceito da biblioteca da integração desenvolvida. O jogo não é avaliado por meio de métodos científicos, pois está fora do escopo deste trabalho. Sua aplicação será apenas uma forma de avaliar a biblioteca construída.

É utilizado no desenvolvimento deste trabalho o Minetest na versão 0.4.9 e o Scratch na versão 2.

1.4 Método de pesquisa

O trabalho é classificado como uma pesquisa aplicada, que segundo Barros e Lehfeld (2000), é motivada pela necessidade de gerar conhecimento com o intuito prático de utilizá-lo.

Seu desenvolvimento é baseado em quarto etapas, sendo estas:

Etapa 1: fundamentação teórica

Nesta etapa é feita uma análise da literatura dos conteúdos específicos referentes ao trabalho:

1. Conceito sobre modularização e bibliotecas.
2. Conceitos acerca do desenvolvimento de jogos educacionais.
3. Estudo acerca do ensino e aprendizagem de programação.
4. Estudo das ferramentas utilizadas neste trabalho, sendo elas o Scratch e o Minetest.

Etapa 2: revisão sistemática do estado da arte.

Nesta etapa é feita uma análise de propostas semelhantes ao objetivo deste trabalho seguindo a abordagem proposta por Kitchenham (2004), que consiste basicamente em:

1. Definir o protocolo de pesquisa.
2. Buscar por ferramentas com propostas semelhantes.
3. Extração e análise das informações.

Etapa 3: modelagem e desenvolvimento da biblioteca e do jogo educacional.

Nesta etapa é o desenvolvimento da integração entre os protocolos de comunicação na forma de uma biblioteca, assim como o desenvolvimento do jogo educacional que a utiliza. Assim, esta etapa se divide em:

Etapa 4: desenvolvimento da integração entre os softwares

Ela é feita seguindo o ciclo de vida em cascata, um modelo de construção de softwares em que seu desenvolvimento é dividido em 6 etapas, que ocorrem de maneira sequencial (BALAJI; MURUGAIYAN, 2012). Estas etapas são:

1. **Análise:** levantamento de requisitos e análise do software, incluindo o estudo dos protocolos de comunicação das ferramentas.
2. **Design:** modelagem do software.
3. **Implementação:** implementação do software;
4. **Teste:** testes de implementação;
5. **Implantação:** por se tratar de uma biblioteca, esta etapa se dá por meio do desenvolvimento de um software que a utilize.
6. **Manutenção**: esta etapa ocorre após a implantação, e propõe manter o software atualizado.

Etapa 5: desenvolvimento do Jogo

Nesta etapa é desenvolvido o jogo educacional que utiliza a biblioteca proposta. O desenvolvimento do jogo será feito utilizando parte do ENgAGED (BATTISTELA et. al., 2014), um processo para desenvolvimento que auxilia sistematicamente na produção de jogos educacionais para o ensino de computação (BATTISTELLA et. al., 2014). A Figura 1 mostra um esquema em alto nível do funcionamento do ENgAGED.

Figura 1- ENgAGED: Educational Games Development (BATTISTELLA et. al., 2014)

As etapas do ENgAGED representam:

1. **Análise:** nesta etapa é feita a análise do jogo educacional, definindo as metas de aprendizagem a serem atingidas, analisando o contexto em que o jogo está inserido e definindo os objetivos de desempenho a serem atingidos.
2. **Projeto:** nesta etapa é desenvolvido o projeto instrucional, definindo o conteúdo a ser abordado pelo jogo, além do desenvolvimento de um instrumento de avaliação da completude dos objetivos definidos na etapa de análise.
3. **Desenvolvimento:** nesta etapa é feita a implementação do jogo previamente projetado. Ela segue o ciclo de vida em cascata.
4. **Execução**: a execução da unidade instrucional, utilizando o jogo criado.

5. **Avaliação**: seguindo as métricas pré-estabelecidas na etapa de projeto.

Etapa 6: avaliação

Nesta etapa é apresentada a avaliação da integração proposta. Esta avaliação será feita de forma indireta, através da execução do jogo educacional que a utiliza, utilizando a metodologia GQM – *Goal/Question/Metric* (BASILI, 1994).

Esta etapa se divide em:

1. Planejamento da avaliação, definindo os aspectos a serem avaliados.
2. Definição da avaliação, apresentando a forma como ela será executada.
3. Execução da avaliação.

1.5 Estrutura do trabalho

O presente trabalho está estruturado da seguinte forma:

- No capítulo 2 são apresentados os conceitos fundamentais para a compreensão do trabalho:
    - Modularização e bibliotecas.
    - Desenvolvimento de jogos educacionais.
    - Algoritmos e programação.
    - Apresentação dos softwares a serem integrados.

- No capítulo 3 é feita uma revisão sistemática da literatura, utilizando o processo de Kitchenham (2004).
- O capítulo 4 apresenta as etapas de análise tanto da biblioteca proposta quanto do jogo educacional que a utiliza.
- O capítulo 5 apresenta a etapa de desenvolvimento da biblioteca e do jogo.
- No capítulo 6 é apresentada a avaliação da biblioteca desenvolvida, seguindo as métricas pré-estabelecidas.
- No capítulo 7 é apresentada a conclusão do trabalho.

2. Fundamentação Teórica

Como este trabalho trata do desenvolvimento de uma biblioteca de integração entre Minetest e Scratch e a implementação do protótipo de um jogo educacional para avaliá-la, este capítulo apresenta os conceitos de desenvolvimento baseado em módulos e de bibliotecas. Como a biblioteca implementada é avaliada por meio de um jogo educacional, são incluídos também conceitos relacionados ao desenvolvimento de jogos educacionais.

Por fim, são apresentadas as ferramentas que serão utilizadas para o desenvolvimento deste trabalho, sendo elas o Minetest e o Scratch.

2.1 Modularização e bibliotecas

A modularização de sistemas é um senso comum na engenharia de software moderna. É comum que a arquitetura de grande parte dos softwares atuais seja em módulos (SARKAR; RAMA; KAK, 2007). Módulos são um conjunto de subprogramas, funções, dados e implementação de determinadas regras de negócio. A modularização beneficia o software em (SARKAR; RAMA; KAK, 2007):

- Compreensibilidade: maior facilidade em compreender o código.
- Testabilidade: o encapsulamento promove uma maior facilidade em testar o código.
- Mutabilidade: o encapsulamento flexibiliza o código ao passo que é possível fazer alterações em partes do código sem afetar o funcionamento das demais.

- Analisabilidade: com alta modularização torna-se mais fácil analisar o código, visto que é possível observá-lo com maior nível de abstração.
- Manutenibilidade: facilita a manutenção do código, seja na mudança do comportamento atual como na adição de novas funcionalidades.

Ao meio de comunicação com um módulo se dá o nome de API, um acrônimo para Interface de Programação de Aplicações (Application Programming Interface em inglês) (SARKAR; RAMA; KAK, 2007).

2.1.1 Programação Modular

Inicialmente a programação modular vinha da ideia de construir programas a partir de programas menores, principalmente sub-rotinas. Contudo, apenas isso não beneficia o software. Ele deve ser desenvolvido de forma que estas pequenas partes sejam autônomas e bem organizadas, e estejam conectadas em uma estrutura simples e coerente (MEYER, 1988).

Meyer (1988) também define os seguintes critérios para que o desenvolvimento possa ser considerado modular:

1. **Decomposição**: um software deve poder ser decomposto em pedaços menores, conectados por uma estrutura simples e que podem trabalhar de forma independente. Cada subprograma pode, de forma recursiva, ser dividido em pedaços ainda menores. Além disso deve-se minimizar ao máximo a quantidade de dependências entre um sistema e outro. Um dos métodos para satisfazer o critério da decomposição é o modelo **top-down**. Ele diz que o problema deve ser pensado em sua forma mais abstrata, e então refinado sucessivamente em subprogramas até um ponto de baixo nível de

abstração, em que o seja viável a implementação da funcionalidade. A Figura 2 mostra uma representação gráfica do modelo *top-down*, do mais alto nível de abstração até o mais baixo.

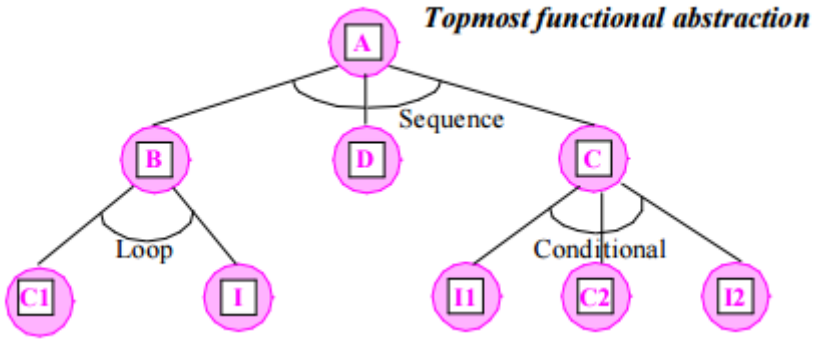


Figura 2- Modelo top-down (MEYER, 1988)

2. **Composição**: o software deve ser construído de forma que os diferentes módulos possam ser combinados entre si, produzindo sistemas diferentes. Como apresentado na Figura 3, podemos ver um conjunto de módulos que podem ser organizados de diferentes formas. Os critérios de composição e decomposição são usualmente difíceis de conciliar. O método *top-down* tende a criar programas difíceis de combinar com outros módulos de forma independente, já que ele é derivado do processo de refinamento de problemas específicos. Desta forma, é necessário conciliar estes dois critérios quando se está desenvolvendo softwares de forma modular. Exemplos de módulos que obedecem este critério são bibliotecas de programas.

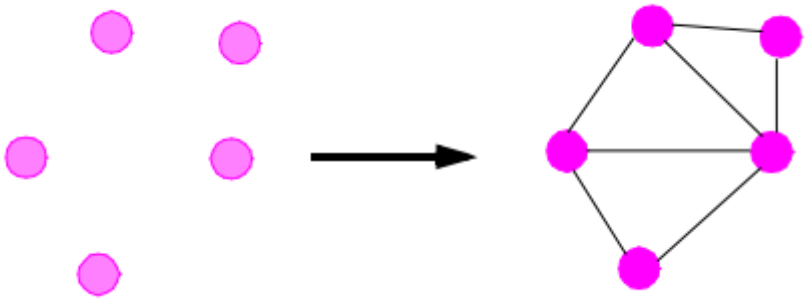


Figura 3 - Módulos podendo ser agrupados de várias formas (MEYER, 1988).

3. **Compreensibilidade**: um software deve ser construído de forma que um usuário humano possa entender o objetivo específico de cada módulo sem conhecer os outros.
4. **Continuidade**: softwares modulares devem ser construídos de forma que pequenas alterações nas especificações de um problema devem acarretar na mudança de poucos módulos.
5. **Proteção**: se durante a execução de um programa ocorrer um comportamento anormal em um módulo, este comportamento não deve afetar os demais módulos do sistema.

Os 5 critérios supracitados acarretam em 5 regras para assegurar a modularidade na construção de um software (MEYER, 1988):

1. **Mapeamento direto**: fazendo uma conexão entre a modelagem de um sistema criado para solucionar um problema e o problema de fato, esta regra diz que "*a estrutura modular idealizada no processo de construção do software deve ser compatível com a estrutura derivada da modelagem do domínio do problema*" (MEYER, 1988). Desta forma é possível rastrear elementos do domínio do problema na solução implementada.

2. **Poucas interfaces**: cada módulo deve conter poucos meios de comunicação com outros módulos. A Figura 4 mostra diversas formas de diferentes módulos se conectarem. Segundo Meyer (1988) o sistema B apresenta um sistema que deve ser evitado, com todos os módulos possuindo conexão com os outros. Meyer (1988) também afirma que o sistema A deve ser evitado, pois possui uma autoridade central, e isso dificulta a satisfação dos critérios da proteção e continuidade.

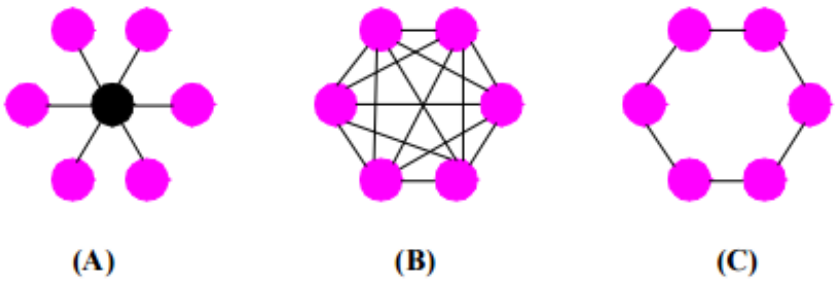


Figura 4- Conexões de diferentes sistemas compostos pelos mesmo módulos (MEYER, 1988).

3. **Interfaces pequenas**: além de possuir poucas conexões entre módulo, deve-se limitar a quantidade de informações passadas nestas conexões, minimizando assim a dependência de um módulo para outro. Esta regra também segue os critérios de proteção e continuidade.
4. **Interfaces explícitas**: a comunicação entre módulos deve ocorrer de forma explícita. Assim, quando um módulo precise ser alterado, deve ser fácil descobrir quais módulos serão afetados.

5. **Encapsulamento**: significa isolar o código, ocultando sua implementação e tornando público apenas o que seja necessário (SCOTT, 2000). O encapsulamento provém do critério de continuidade, já que é possível alterar a parte privada de um módulo sem afetar as demais.

Sarkar, Rama e Kak (2007) ainda atribuem à modularização os seguintes princípios:

- **Similaridade de propósito**: os módulos devem agrupar dados e funções que façam sentido entre si, oferecendo um serviço autocontido e bem definido. Deve-se minimizar a dispersão do objetivo, ou seja, o módulo deve apenas fazer a função para o qual foi planejado.
- **Compilabilidade**: objetiva tornar o módulo o menos dependente possível de outros módulos no momento da compilação. Desta forma é mais fácil o módulo ser testado de forma independente.
- **Extensibilidade**: no desenvolvimento de sistemas modularizados deve-se maximizar a extensibilidade de cada módulo individualmente, de forma que cada módulo possa crescer sem alterar o comportamento dos demais.
- **Testabilidade**: deve-se maximizar a testabilidade dos módulos de forma independente.
- **Dependências acíclicas**: um módulo deve evitar depender de módulos que direta ou indiretamente dependem dele. Ele deve ser projetado da forma mais unidirecional possível.

2.1.2 Bibliotecas

Um dos tipos de módulos existentes são as bibliotecas de aplicações (MEYER, 1988). Elas são um conjunto de funções implementadas em alguma linguagem de programação e possuindo uma API para a mesma.

As bibliotecas podem ser incorporadas ao software de duas maneiras, sendo denominadas (YACOUB; AMMAR; MILI, 1999):

1. **Bibliotecas estáticas**: são aquelas que possuem seus códigos invocados durante a construção do programa. Uma vantagem é o fato de não ser necessário incluir o código da biblioteca, apenas o resultado de sua compilação ou pré-compilação. Entretanto, possuem como desvantagem a necessidade de reassociação em cada atualização. Outra desvantagem é que em sistemas *multitasking* cada aplicação que utilize a biblioteca deverá possuir uma cópia sua.
2. **Bibliotecas dinâmicas**: possuem seus códigos invocados após a construção do programa, seja no início da execução ou no meio. Seus códigos são invocados em tempo de execução. Neste tipo de biblioteca várias aplicações podem utilizar uma mesma cópia da biblioteca.

Como a biblioteca desenvolvida neste trabalho tem o objetivo de integrar dois softwares para permitir o desenvolvimento de jogos educacionais, a próxima seção trata desse assunto.

2.2 Desenvolvimento de Jogos Educacionais

Como a Biblioteca desenvolvida neste trabalho será avaliada por meio de um jogo educacional, nesta seção são apresentados os conceitos envolvidos no desenvolvimento de jogos educacionais.

2.2.1 Jogos Educacionais

Jogos educacionais são atividades que envolvem competição, sejam regulados por regras e possuam um objetivo a ser alcançado, além de serem projetados especificamente para o ensino de algum conteúdo (PRENSKY, 2006).

Uma vantagem de sua utilização é a maior interatividade do aluno com o conteúdo a ser ensinado, pois ele propicia o aprendizado através da experiência (MCDONALD, 2004). Outro benefício de sua utilização no ensino é o fato de ele ser atrativo para o aluno, já que ele conecta o estudante ao conhecimento de uma forma divertida (SAVI, 2011).

Jogos educacionais podem ser encontrados no formato não digital (p.ex. tabuleiro, cartas) ou no formato digital. Os digitais são aqueles em que a interação do usuário com o jogo é feita através de um dispositivo virtual, como computadores, videogames e celulares (BALASUBRAMANIAN; WILSON, 2006).

É possível ampliar a eficiência do processo de ensino e aprendizagem através do uso de jogos educacionais digitais. Além de capturar a atenção do estudante, ele propicia o aprendizado através da descoberta, já que com jogos digitais tipicamente existe um feedback imediato de suas ações, é possível tomar atitudes e planejar ações de acordo com o resultado obtido em cada interação (SAVI, 2011). Além disso, os jogos educacionais digitais ainda podem servir como fator de socialização, aproximando os jogadores seja de forma competitiva ou cooperativa.

Os elementos básicos dos jogos digitais são (BALASUBRAMANIAN; WILSON, 2006):

- Papel ou personagem do jogador: entidade que representa o jogador dentro do jogo;
- Regras: delimitações do jogo;
- Metas e objetivos: delimita quando um jogo se encerra e a vitória é alcançada;
- Desafios: problemas dentro do jogo que devem ser solucionados para se atingir a vitória;
- História ou narrativa: contexto em que o jogo é situado, promovendo imersão do jogador;
- Iterações do jogador: ações que o jogador pode tomar de forma a controlar seu personagem;
- Estratégias: planejamento do jogador com relação à ordem e o momento das ações a serem tomadas para a conclusão de seus objetivos.
- Resultados: *feedback* que garante ao jogado ruma forma de avaliar seu desempenho ao longo do jogo.

O presente trabalho possui como objetivo a construção de uma biblioteca que fará a integração entre o Minetest e o Scratch, bem como uma prova de conceito na forma de um jogo educacional para o ensino de programação que a utilize. Por isso a seção seguinte apresenta a fundamentação teórica acerca dos aspectos relevantes ao ensino e linguagens de programação.

2.3 Algoritmos e programação

A biblioteca proposta por este trabalho será testada por meio de um jogo educacional para o ensino de programação. Por isso, nesta sessão é apresentada a fundamentação teórica acerca deste tema.

Um algoritmo é "*uma sequência finita de instruções que, se seguido corretamente, efetua determinada tarefa*" e possuem as seguintes características (HOROWITZ; SAHNI, 1978):

- **Entrada**: fator externo que influencia algoritmo. Um algoritmo pode ter zero ou mais entradas.
- **Saída**: resultado da execução de um algoritmo, podendo ter uma ou mais saídas.
- **Definição**: cada instrução deve ser clara e sem ambiguidade.
- **Finitude**: um algoritmo deve conter um conjunto finito de instruções.
- **Efetividade**: um algoritmo deve ser uma sequência de instruções simples.

Em computadores, uma forma de expressar os algoritmos é através do uso de linguagens de programação, que são projetadas para que cada instrução possua um único significado. Dessa forma, um programa é definido como sendo "*a expressão de um algoritmo em uma linguagem de programação*" (HOROWITZ; SAHNI, 1978).

2.3.1 Elementos básicos de algoritmos e programas

De forma geral, pode-se considerar que os principais elementos que compões um algoritmo são (LOPES, 2014):

- **Desvio incondicional**: desvio que permite que o fluxo de execução de um algoritmo ou programa seja alterado para qualquer outro trecho deste mesmo algoritmo ou programa (UFPR, 2015).

Tradicionalmente referencia-se a este desvio utilizando o termo em inglês “*go-to*”, que em uma tradução livre ficaria: “ir-para”.

- **Desvio Simples**: desvios que apenas executam uma instrução ou bloco de instruções caso alguma condição seja satisfeita (LOPES, 2014). Tradicionalmente referencia-se a este desvio utilizando o termo em inglês “*if-then”*, que em uma tradução livre seria algo como: “se-então” (**se** hoje não chover, **então** vou à praia). A Figura 5 mostra o funcionamento de um desvio simples.

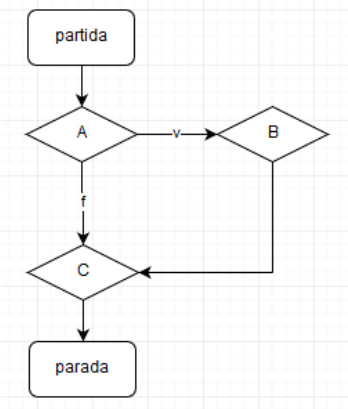


Figura 5 – Fluxo de um desvio simples

- **Desvio Composto**: desvios que executam instruções diferentes caso alguma condição não seja satisfeita (LOPES, 2014). Tradicionalmente referencia-se a este desvio utilizando o termo em inglês “*if-then-else*”, que em uma tradução livre ficaria: “*se-então-senão*” (**se** hoje não chover, **então** vou à praia. **Senão**, vou ao shopping). O fluxo de um desvio composto pode ser visto na Figura 6.

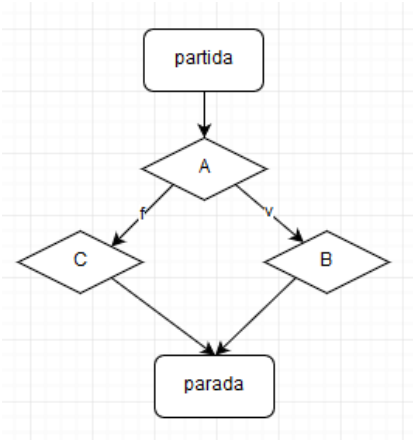


Figura 6 - Fluxo de um desvio composto

- **Desvios encadeados**: esta situação ocorre quando há um desvio dentro de outro, seja um desvio simples ou composto (LOPES, 2014). Usualmente isso ocorre quando é necessário fazer mais de uma verificação dentro de um algoritmo.
- **Operadores lógicos**: operadores lógicos em um algoritmo são utilizados usualmente para agrupar condições em desvios, evitando a utilização de desvios encadeados (LOPES, 2014). Os principais operadores lógicos utilizados em algoritmos são:
    - **Operador "E"**: Considera verdadeiro o teste para a condição se todas as comparações ou afirmações feitas são verdadeiras.
    - **Operador "OU"**: Considera verdadeiro o teste para a condição se alguma das comparações ou afirmações feitas são verdadeiras.
    - **Operador "não"**: Inverte o resultado de uma comparação ou afirmação: se uma comparação *A* resultava verdadeiro, o uso do operador "não" torna o resultado da comparação *A* falso.

- **Desvio de condição múltipla**: desvio múltiplo em que, dada uma pergunta, dependendo de seu resultado o algoritmo executa determinada instrução ou bloco de instruções, porém não se limitando a perguntas binárias (LOPES, 2014). O fluxo de um desvio de condições múltiplas é visto na Figura 7.

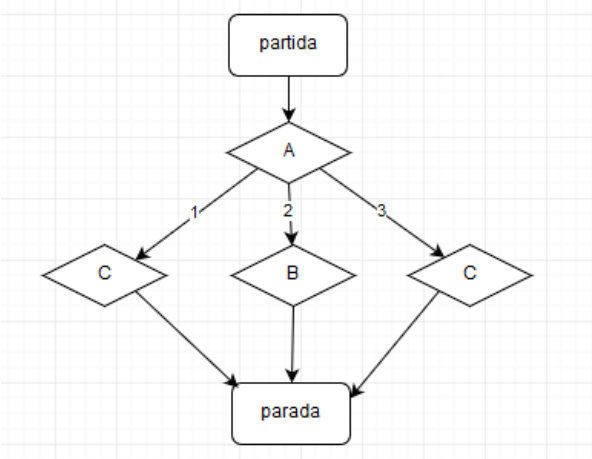


Figura 7 - Fluxo do desvio de condições múltiplas

- **Estruturas de repetição**: são estruturas que permitem a execução de um trecho do algoritmo por mais de uma vez, sendo elas (MOURA; FERBER, 2008):
    - ***While* (enquanto)**: estrutura que permite que a execução de um trecho do algoritmo enquanto determinada condição for verdadeira.

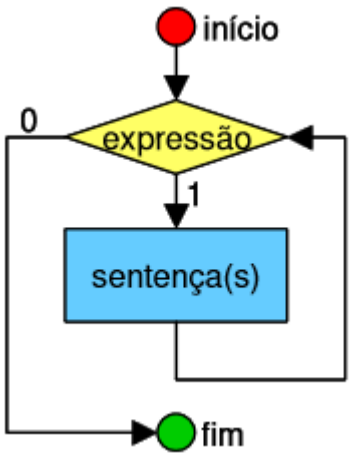


Figura 8- Estrutura de repetição while (enquanto) (MOURA; FERBER, 2008)

- ***For* (por)**: estrutura que permite a execução de um trecho do algoritmo por um número predeterminado de vezes.

Em um alto nível de abstração, algoritmos podem ser divididos em três grandes categorias (DIVERIO; MENEZES, 2009):

- Monolítico
- Recursivo
- Interativos

2.3.2 Algoritmos Monolíticos

São algoritmos baseados em desvios condicionais e incondicionais que não fazem uso de mecanismos auxiliares a fim de estruturar o algoritmo. A lógica é constituída de um único bloco de instruções, como apresentado na Figura 9 (DIVERIO; MENEZES, 2009).

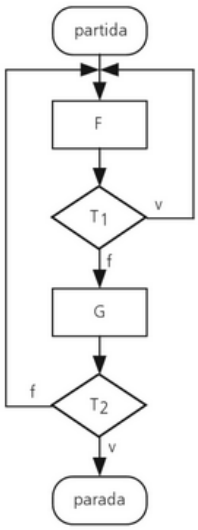


Figura 9 - Exemplo de algoritmo monolítico (DIVERIO; MENEZES, 2009)

2.3.3 Algoritmos recursivos e iterativos

Um algoritmo é considerado **recursivo** quando uma de suas instruções é uma chamada para o próprio algoritmo (HOROWITZ; SAHNI, 1978). Dentro dos algoritmos recursivos existem os:

- Recursivos diretos: quando um algoritmo *A* é chamado diretamente no corpo de *A*.
- Recursivos indiretos: quando um algoritmo *A* é chamado por um algoritmo *B*, mas uma das instruções de *B* é uma chamada ao algoritmo *A*.

A vantagem do uso de algoritmos recursivos é a sua clareza. Processos complexos e sofisticados podem ser expressados por poucas instruções de forma clara e objetiva (HOROWITZ; SAHNI, 1978).

Um exemplo de algoritmo recursivo pode ser visto na Figura 10, escrito na linguagem de programação Java.

```
public int fatorialDe(int x){
    if(x <= 1){
        return 1;
    } else {
        return x * fatorialDe(x - 1);
    }
}
```

Figura 10 - Algoritmo fatorial recursivo

A Figura 10 apresenta um algoritmo denominado *fatorialDe* que possui como entrada um número inteiro x e como saída o valor fatorial deste número. O algoritmo segue a sequência de instruções:

1. Verifica o valor de *x;*
2. Caso o valor seja menor ou igual a 1, o algoritmo terá como saída o número 1;
3. Caso o valor de *x* seja maior que 1, o algoritmo terá como saída:

- $x \times (saída\ do\ algoritmo\ fatorialDe\ (x-1))$

De forma recursiva, com apenas 3 instruções foi possível expressar a função fatorial em um algoritmo.

Já os algoritmos **iterativos** são aqueles que utilizam estruturas de repetições em seu corpo, de forma a evitar a dificuldade de entendimento de algoritmos monolíticos que possuem muitos desvios incondicionais, também conhecidos como "quebra de lógica" DIVERIO; MENEZES, 2009).

Assim é finalizada a fundamentação teórica sobre algoritmos e programação. Na seção seguinte são apresentadas as ferramentas utilizadas na construção da biblioteca.

2.4 Minetest

Minetest é um jogo de código livre disponível no estilo caixa de areia criado em 2010, composto por um ambiente virtual em 3D. Em sua versão 0.4.9 ele é disponível nas plataformas Windows, GNU/Linux e OSX.

Esse ambiente virtual é composto por diversos tipos de blocos, em sua grande maioria no formato cúbico, que compõem o cenário do jogo. Este cenário é referenciado no jogo com o termo mapa. Existem 2 tipos de blocos:

- Blocos de cenário: que são aqueles com os quais o personagem pode interagir. Exemplos são: terra, pedra, água.
- Blocos de personagem: que são aqueles que o personagem pode possuir e com os quais ele pode interagir com os blocos de cenário. Exemplos são: tochas, espadas, machados.

Cada bloco possui características diferentes, sejam meramente visuais, como possuindo funções e utilizações diferentes dentro do jogo. O jogador controla um

personagem, que por meio das ferramentas interage com o cenário, destruindo ou construindo blocos livremente, podendo criar até as construções mais complexas como prédios e casas.

Um mapa no Minetest possui um espaço de 62 mil blocos cúbicos, gerados de forma pseudoaleatória ou através de um seed informado pelo usuário no momento de sua criação, fazendo com que a experiência de cada um seja diferente durante o jogo. Ele também possui a opção de carregar um mapa já existente, tornando-os portáteis, já que é possível construir algo em um computador, e abrir este mesmo mapa em um outro. Com isso, é possível que o jogo educacional proposto neste trabalho possua o mesmo mapa em qualquer ambiente em que for aplicado. A Figura 11 mostra um exemplo de mapa.

Figura 11 – Mapa no Minetest (MINETEST, 2015)

O Minetest não possui um objetivo final, possibilitando ao jogador explorar livremente seu ambiente de forma indefinida. O jogo obedece às mecânicas da maioria dos jogos 3D da atualidade, em que o jogador pode se movimentar nos 3 eixos (x, y e z).

O Minetest possui uma API (Interface de Programação de Aplicações) para a modificação de elementos do jogo sem edição direta do código fonte. Essas

modificações, chamadas popularmente de *mods* (em tradução livre, “modificação”) utilizando a linguagem Lua[7] em sua versão 5.1.

O jogo pode ser jogado por um ou mais jogadores. No caso de um jogo *multiplayer*, que são aqueles com mais de um jogador, uma instância torna-se o servidor e as demais são os clientes. É possível fazer a comunicação com softwares de terceiros através do seu protocolo de comunicação de rede, através do recebimento e envio de pacotes a nível de rede.

Essa comunicação se dá na camada de transporte e acontece por meio do protocolo UDP. A Figura 12 apresenta como ocorre a autenticação entre um cliente e um servidor no Minetest em sua versão 0.4.9.

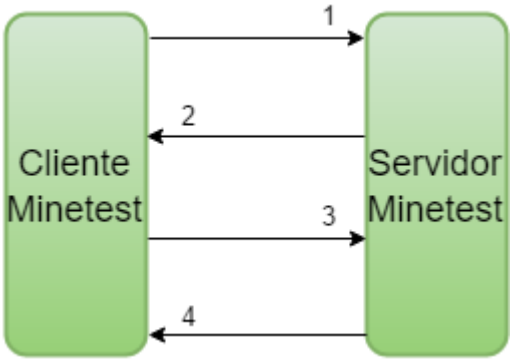


Figura 12 – Diagrama da autenticação no Minetest

Os eventos apresentados na Figura 12 representam:

1. O cliente envia ao servidor um comando para o início da autenticação.
2. O servidor indica ao cliente que está pronto para o início.
3. O cliente envia então o nome de usuário e um *hash* da senha.
4. O servidor faz as verificações de segurança e então autoriza a conexão, enviando informações como:
    - Usuários conectados.

[7] http://www.lua.org

- Dados da conexão: tempo de resposta, *ping*, etc.
- Cenário: o mapa onde o jogo ocorre.
- Dados do usuário: pontos de vida, localização, etc.

2.5 Scratch

Scratch[8] é uma plataforma de desenvolvimento de programas criada no Instituto de Tecnologia de Massachusetts. Seu objetivo é "*ajudar os jovens a aprender a pensar de maneira criativa, sistemática e trabalhar de forma colaborativa*" (SCRATCH, 2015). Nele é permitido a construção de algoritmos por meio de blocos de instruções visuais. Dessa forma é possível programar graficamente, algo que pode ser mais atrativo para crianças. O Scratch é utilizado em mais de 150 países e traduzido para mais de 40 idiomas diferentes.

Ele possui versões online[9] e off-line, chamadas de editores, que podem ser acessados através de um navegador ou instaladas diretamente no computador. Estes editores são compostos por:

- Atores: uma área que mostra objetos que podem ser controlados por programas desenvolvidos.
- Palco: área em que os atores executam o algoritmo desenvolvido.
- Sons: área de visualização e edição de sons a serem utilizados por meio da programação.
- Pano de fundo: área de criação e edição da imagem que fica atrás de onde os atores estão no palco.
- Scripts: área de criação e edição dos programas.

[8] https://scratch.mit.edu/
[9] https://scratch.mit.edu/projects/editor/

A imagem Figura 13 apresenta a Tela inicial do editor off-line do Scratch em sua versão em inglês.

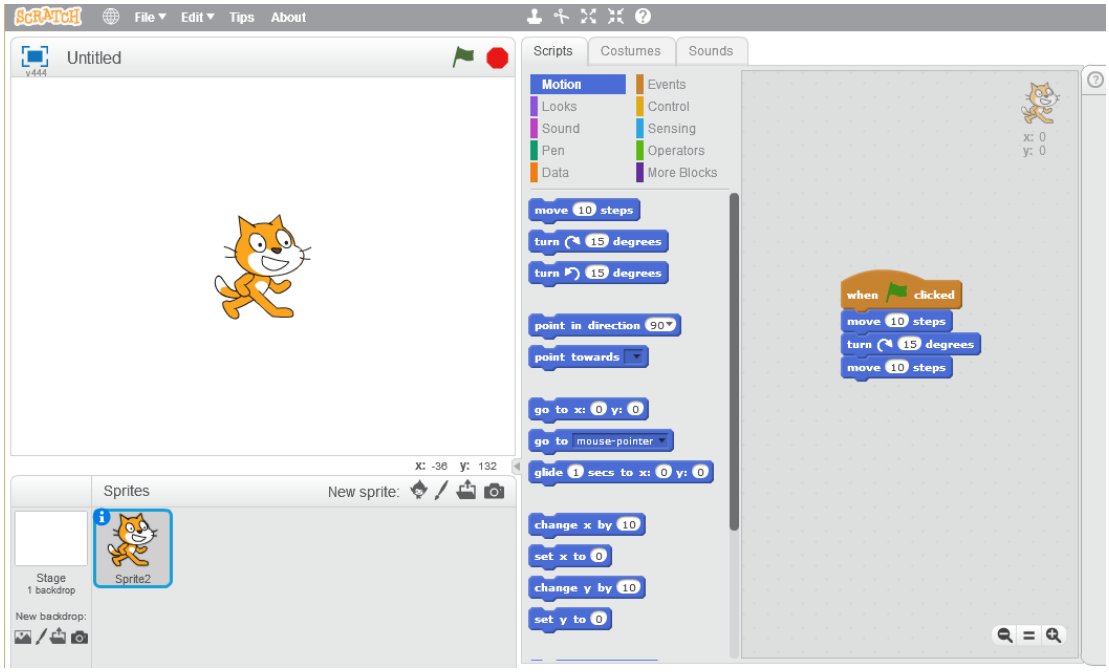


Figura 13- Tela inicial Scratch (SCRATCH, 2016)

As instruções presentes no Scratch podem ser consideradas uma linguagem de programação, pois são instruções não-ambíguas, podem possuir entradas e saídas e são um conjunto de instruções finitos (HOROWITZ; SAHNI, 1978). A sintaxe desta linguagem se dá através de instruções em blocos visuais, organizados em 9 categorias:

- Movimento: instruções que fazem com que o ator selecionado se movimente.
- Aparência: instruções que mudam o estado do palco de forma exclusivamente gráfica.
- Som: instruções que emitem e controlam sons criados na parte de criação de sons.
- Caneta: instruções que traçam linhas no palco.

- Variáveis: instruções para criação e modificação de variáveis.
- Eventos: instruções que são ativadas quando o determinado evento ocorrer.
- Controle: instruções de controle do programa criado. Nesta categoria estão presentes os desvios condicionais e as estruturas de repetição.
- Sensores: instruções que indicam determinado estado do computador, como por exemplo: instrução que indica se o mouse está sendo pressionado ou não.
- Operadores: instruções de operação tanto aritméticas como lógicas.

Existe ainda uma categoria denominada *Mais Blocos*, em que o usuário pode criar blocos personalizados.

A linguagem de programação do Scratch se baseia em blocos visuais, que representam as instruções presentes nas categorias previamente descritas. Através da junção desses blocos, é possível construir programas para serem executados pelo editor do Scratch. O exemplo da Figura 14, desenvolvido na linguagem de programação do Scratch, mostra um programa que, ao ser executado:

1. Dispara um evento quando o usuário clicar na bandeira verde, presente no editor do Scratch.
2. Repete por 10 vezes a instrução que move o ator por 10 passos.

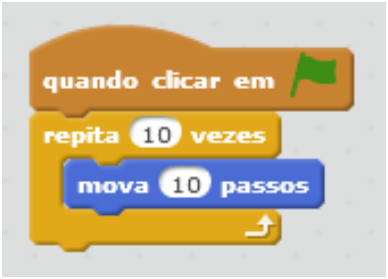


Figura 14 - Exemplo de programa construído no Scratch (SCRATCH, 2016)

No editor off-line, é existe a funcionalidade de importar uma extensão HTTP, que é uma forma de comunicar o Scratch com outros softwares por meio de requisições HTTP. Isso ocorre com a importação de um arquivo no editor off-line do Scratch. Este arquivo, denominado extensão HTTP, deve seguir o formato JSON e é composta pelos seguintes itens:

- **ExtensionName**: elemento que define o nome da extensão HTTP.
- **ExtensionPort**: a porta em que o Scratch enviará as requisições, fazendo assim a comunicação.
- **BlockSpecs**: especificação dos novos blocos de comandos. Estes novos blocos possuem os seguintes elementos:
    - Tipo do bloco.
    - Texto exibido no bloco, podendo ou não haver parâmetros de entrada.
    - Endereço a ser acessado quando o comando for executado.
    - Valores padrões para os parâmetros do comando, caso existam.
- **Menus**: lista de menus para a criação de comandos que apenas aceitem parâmetros pré-definidos.

Quando um dos comandos da extensão HTTP é executado no Scratch, ele gera uma requisição HTTP na porta definida na extensão e no caminho explicitado em sua respectiva especificação na extensão. Um exemplo de extensão HTTP pode ser visto na Figura 15, contendo apenas o comando girar_para, que possui como parâmetro um menu com as opções esquerda e direita.

```
{
    "extensionName": "Extensão Exemplo",
    "extensionPort": 12345,
    "blockSpecs": [
        [
          " ",
          "girar para %m.posicoes",
          "girar_para",
          "direita"
        ]
    ],

    "menus": {
      "posicoes": ["direita", "esquerda"]
    }
}
```

Figura 15 - Extensão HTTP

Após importada no Scratch, a extensão apresentada na Figura 15 pode ser encontrada no menu Mais Blocos, e seu resultado é o comando representado na Figura 16.

Figura 16 - Exemplo de uma Extensão HTTP

Ao ser executado, este comando irá gerar uma requisição no endereço localhost:12345/girar_para/direita. Este endereço representa:

- *Localhost*: o endereço da máquina em que o Scratch está sendo executado.
- *12345*: a porta em que a requisição vai ser feita.
- */girar_para*: o endereço do comando.
- */direita*: o parâmetro no comando do Scratch na Figura 16

Atualmente o Scratch apenas aceita comandos unilaterais. Ou seja, não é possível a criação de blocos que recebam resultados, portanto não é possível enviar informações do servidor para o Scratch.

3. Estado da Arte

Este capítulo apresenta o estado da arte das integrações existentes envolvendo os softwares Scratch e Minetest. A análise do estado da arte é realizada seguindo as principais etapas do método de revisão sistemática de literatura definido por Kitchenham (2004), adaptadas para uma pesquisa de trabalho de conclusão de curso de graduação. Esse método é realizado definindo a revisão da literatura disponível, executando a busca e fazendo a extração dos resultados, bem como sua análise.

3.1 Definição do protocolo de revisão

Para identificar o estado da arte do tema proposto por este trabalho, foi definida como pergunta de pesquisa de revisão: “Como são as bibliotecas de integração entre Minetest e Scratch para apoio ao ensino de programação? ”.

Para tal pesquisa, foi utilizado o mecanismo de buscas Google Scholar[10], que agrega diferentes bases de bibliotecas digitais neste domínio. As buscas foram realizadas utilizando os termos em português e sua tradução para o inglês. Os resultados analisados da busca foram apenas os que estavam nestes idiomas, e dentro do período de 2010 a 2016. Além disso, apenas os 50 primeiros resultados obtidos nas execuções das buscas foram analisados, devido às limitações de um trabalho de conclusão de curso de graduação.

[10] https://scholar.google.com.br/

3.1.1 Critérios de inclusão/exclusão

Foram utilizados os seguintes critérios para a inclusão da literatura:

- O material encontrado deve referir-se a uma ferramenta que faça utilização do Scratch (ou de qualquer forma de programação visual) ou do Minetest por meio de uma integração. Por questão de semelhança de propósito, o jogo Minecraft também será aceito.
- O material deve ser voltado ao ensino de programação.

3.1.2 Critério de qualidade

O critério de qualidade foi definido como:

- A profundidade como o material apresenta detalhes da implementação da integração com as ferramentas.

3.1.3 Dados a serem extraídos

Os dados extraídos dos trabalhos selecionados, de forma a procurar responder à pergunta de pesquisa, são:

- Tecnologias utilizadas para a integração.
- Informações sobre os protocolos utilizados.
- Forma como foi avaliado o trabalho.
- Plataformas disponíveis.

3.1.4 String de Busca

Os termos utilizados na busca são derivados da pergunta de pesquisa e são apresentados no Quadro 1:

Termo	Sinônimo	Tradução (inglês)
“Scratch”	“programação visual”	“visual programming”

“Minetest”	“Minecraft”	
“Biblioteca”	“Biblioteca de software”, “módulo”, “módulo de software”, “biblioteca estática”	“Library”, “software library”, “module” “software module”, “static library”
“Integração”	“Integração de software”, “integração de sistemas”	“integration”, “software integration”, “system integration”
“Ensino de Programação”	Ensino de lógica de programação”, “ensino de algoritmos”, “ensino de construção de algoritmos”	“Programming education”, “programming logic education”, “algorithm education”, “algorithm building education”

Quadro 1- Sinônimos e traduções dos termos de busca

As Strings de busca derivadas aparecem no Quadro 2, bem como o número total de resultados retornados:

Termos de Busca	Número de resultados
("scratch" OR "programação visual" OR "visual programming") AND ("minetest" OR "minecraft") AND("integração" OR "integração de software" OR "integração de sistemas" OR "software integration" OR "system integration") AND ("biblioteca" OR "biblioteca de software" OR "módulo" OR "módulo de software" OR "biblioteca estática" OR "library" OR "software library" OR "module" OR "software module" OR "static library") AND ("ensino de programação" OR "ensino de lógica de programação" OR "ensino de algoritmos" OR "ensino de construção de algoritmos" OR "Programming education" OR "programming logic education" OR "algorithm education" OR "algorithm building education") PERÍODO (2010 a 2016)	5
(("scratch" OR "programação visual" OR "visual programming") OR ("minetest" OR "minecraft")) AND ("integração" OR "integração de software" OR "integração de sistemas" OR "software integration" OR "system integration") AND ("biblioteca" OR "biblioteca de software" OR "módulo" OR "módulo de software" OR "biblioteca estática" OR "library" OR "software library" OR "module" OR "software module" OR "static library") AND ("ensino de programação" OR "ensino de lógica de programação" OR "ensino de algoritmos" OR "ensino de construção de algoritmos" OR "Programming education" OR "programming logic education" OR "algorithm education" OR "algorithm building education") PERÍODO (2010 a 2016)	795

Quadro 2- Termos de busca.

3.2 Execução da busca

As buscas foram feitas em setembro de 2016. No total foram necessárias 3 iterações de busca, apresentadas na sequência.

Primeira e segunda execuções

A primeira execução retornou um total de 5 resultados, nenhum, contudo atendeu aos critérios de inclusão.

No intuito de ampliar o número de resultados, foi feita a modificação do termo de busca, retirando a restrição de “AND” dos termos mais específicos “scratch” e “minetest”, vide Quadro 2, o que trouxe um total de 795 resultados. Foi encontrado apenas um resultado que se enquadrou nos critérios de inclusão: um protótipo de aplicação visual para controle de mecanismos e sensores (HEINEN et. al., 2014). Utilizando ainda o método *Snowball* (BIERNACKI; WALDORF, 1981), foi encontrado o s2a_fm (S2A_FM, 2016), uma extensão para o Scratch para o controle de micro controladores Arduino[11].

Terceira execução

Por conta da escassez de trabalhos relevantes no motor de busca selecionado, foi tomada a decisão de fazer a pesquisa no buscador do Google[12]. Analisando os 50 primeiros, foi identificado um trabalho para fazer a programação visual no Minecraft (PRUSS, 2016).

3.3 Extração das informações análise dos resultados

Nesta seção são apresentados os resultados extraídos das execuções das buscas. Os resultados obtidos são:

Resultado 1	
Trabalho	Um protótipo de aplicação, que utiliza a programação visual como forma de controle de mecanismos e sensores no Raspberry Pi[13] (HEINEN et. al., 2014).
Tecnologias	O projeto faz o uso da biblioteca para programação visual Blockly[14]. Uma

[11] https://www.arduino.cc/
[12] https://google.com.br
[13] https://www.raspberrypi.org/
[14] https://developers.google.com/blockly/

	biblioteca que gera código de programação através de blocos visuais como o Scratch
Protocolos de integração	O Blockly é uma biblioteca JavaScript que faz uso de recursos HTML, portanto deve ser utilizada em um navegador de internet. Para sua utilização, é necessário referenciá-lo no cabeçalho da página HTML.
Avaliação	O trabalho foi avaliado por meio da utilização da aplicação em um ambiente de testes
Plataformas	A ferramenta está disponível apenas na plataforma Raspberry Pi, já que sua intenção é o controle de seus equipamentos físicos por meio da programação visual.
Resultado 2	
Trabalho	s2a_fm: ferramenta para o controle de microprocessadores Arduino utilizando programação visual.
Tecnologias	o projeto é desenvolvido na linguagem de programação Python, e permite que os usuários programem os microcontroladores Arduino através de programação visual.
Protocolos de integração	Para fazer a comunicação com o Scratch, é utilizada a extensão HTTP previamente mencionada disponível para o Scratch (vide capítulo 2). O servidor é criado na linguagem Python e se comunica com o Scratch por meio de requisições HTTP.

Avaliação	Sua página oficial não disponibiliza informações sobre avaliações feitas utilizando a ferramenta.
Resultado 3	
Trabalho	Um mod que permite programar as ações do personagem no Minetest. *Visual Programming for Minetest* (PRUSS, 2015) é um *mod* para o Minetest que faz comunicação com um editor de códigos visual online. Também existe uma versão disponível para o Minecraft.
Tecnologias	Também utiliza o Blockly como forma de programação visual, e é desenvolvido nas linguagens de programação Python e Lua.
Protocolos de integração	O *mod* para o Minetest é feito em Lua, mas invoca comandos Python presentes nele. O interpretador Python se comunica com o editor por meio de envio de *sockets* disponíveis na página.
Avaliação	Sua página oficial não disponibiliza informações sobre a avaliação.
Plataformas	Disponível apenas para a plataforma Windows.

3.4 Discussão

As buscas revelam que há uma escassez de trabalhos científicos que utilizam as APIs de integração do Scratch e do Minetest para o ensino de programação, tanto em conjunto como em separado

Durante as buscas foi constatado que diversos trabalhos fazem menção ao Scratch como forma de ensino de programação. Porém a maioria era meramente a avaliação da aplicação do Scratch de forma pedagógica. Alguns exemplos são:

- A utilização do Scratch como estratégia de ensino e aprendizagem da programação: uma experiência no âmbito do ensino profissional (LOMBA, 2014);
- Utilizando o SCRATCH nas aulas de Lógica de Programação do Proeja: Um relato de experiência (ALENCAR, 2014)
- Dominando para não ser dominado: Autonomia tecnológica com o Projeto Jovem Hacker (AMIEL et. al., 2015)

Uma possível justificativa para isso é que o Scratch é uma ferramenta que por si só já apresenta um ambiente de desenvolvimento completo.

A partir das buscas o autor tomou conhecimento da existência do Blockly, uma outra alternativa na produção de código visual.

3.4.1 Ameaças a validade da revisão da literatura

Na intenção de estender o número de trabalhos a serem analisados, não foi incluído nos critérios de inclusão a necessidade do trabalho encontrado ser uma biblioteca para construção de softwares de apoio ao ensino de programação. Isso pode ameaçar a validade da revisão, já que este é um dos objetivos propostos pelo presente trabalho.

As alterações nas strings de busca também podem ameaçar a validade, pois alteram a semântica da pesquisa, podendo excluir trabalhos relevantes.

A última execução sendo realizada no mecanismo de buscas do Google pode ser uma ameaça, já que alguns dos trabalhos encontrados não são trabalhos acadêmicos publicados em meios de prestígio.

A análise apenas dos 50 primeiros resultados de cada busca também é uma ameaça, pois ela depende exclusivamente da ordem de prioridade definida pelo motor, podendo ocultar trabalhos relevantes.

De posse das informações extraídas, finaliza-se o capítulo da análise sistemática da literatura. No capítulo seguinte é apresentada a análise de desenvolvimento da integração entre o Scratch e o Minetest na forma de uma biblioteca para construção de softwares de apoio ao ensino de programação.

4. Desenvolvimento da biblioteca

Este capítulo apresenta o desenvolvimento da biblioteca proposta. As etapas do desenvolvimento de um software seguindo o ciclo de vida em cascata (PRESSMAN, 2016), que no contexto da biblioteca são:

1. Levantamento de requisitos para a biblioteca proposta por este trabalho.
2. Projeto da biblioteca.
3. Implementação da biblioteca.

4.1 Análise da biblioteca

Já que o objetivo da biblioteca proposta por este trabalho é fazer uma integração entre os softwares Minetest e Scratch, a ela foi dado o nome: MineScratch.

Com base nos protocolos de comunicação (vide capítulo 2 – Fundamentação Teórica) e nas funcionalidades existentes nos softwares que serão integrados, foi identificado um conjunto mínimo de funcionalidades que um módulo de integração entre essas ferramentas deveria suportar para poder apoiar o desenvolvimento de jogos educacionais para ensino de programação. Assim, em alto nível de abstração, as funcionalidades de uma biblioteca de integração são:

- Criar uma conexão com o Scratch.
- Realizar autenticação em um servidor Minetest.
- Possibilitar a criação de instruções que podem ser recebidas no Scratch.
- Possibilitar a criação de comandos para serem enviados ao Minetest.
- Enviar e receber de comandos do Minetest.

- Receber instruções vindas do Scratch.
- Manter registro das informações do personagem durante a conexão.

A partir destas funcionalidades identificadas como necessárias para que a integração seja possível, são definidos e priorizados os requisitos funcionais e não-funcionais apresentados no Quadro 3:

Requisito	**Descrição**	**Prioridade**		
		Alta	**Média**	**Baixa**
RF01	Estabelecer conexão com um servidor informado por parâmetro Minetest.	X		
RF02	Estabelecer conexão com uma instância do Scratch rodando na mesma máquina.	X		
RF03	Possibilitar a criação de instruções proveniente do Scratch e serem interpretadas pelo MineScratch.	X		
RF04	Possibilitar a criação de comandos a serem enviados para o servidor Minetest previamente autenticado.	X		
RF05	Manter registro das informações do personagem autenticado: • Pontos de vida. • Posição no cenário.		X	
RF06	Receber instruções vindas do Scratch.	X		
RF07	Receber/enviar comandos para o Minetest.	X		
RNF01	Compatibilidade com a plataforma Java.		X	
RNF02	A comunicação MineScratch-Minetest deve ser pelo protocolo UDP.	X		
RNF03	A comunicação MineScratch-Minetest deve ser por meio de requisições HTTP.	X		
RNF04	O MineScratch não deve possuir custo com licenças.			X

Quadro 3 - Requisitos funcionais da biblioteca

4.2 Projeto da biblioteca

A MineScratch pode ser classificada como uma biblioteca estática, pois a sua utilização existe por meio de um código pré-compilado pela aplicação.

O sistema segue a arquitetura cliente-servidor, em que uma instância da aplicação faz requisições e recebe serviços de um servidor (KUROSE; ROSS, 2006). Existe um servidor Minetest conectado pela rede com diversas instâncias do MineScratch (e a aplicação que a utilizará). Cada uma se comporta como um cliente Minetest e se comunica com uma instância do Scratch que deve estar rodando na mesma máquina. A Figura 17 mostra uma visão geral da arquitetura do sistema em alto nível de abstração.

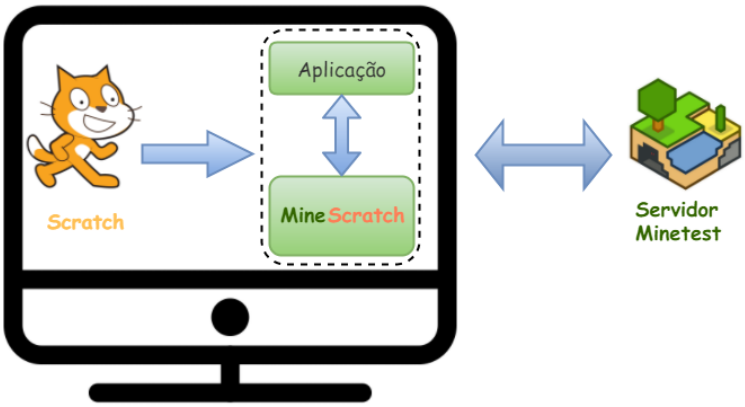


Figura 17 - Arquitetura geral do sistema

No contexto da arquitetura apresentada na Figura 17, no computador do usuário existe uma instância do editor do Scratch em execução e uma instância da aplicação que faz uso da biblioteca MineScratch, estes dois se comunicam por meio de requisições HTTP, no sentido Scratch-aplicação. Fora do computador, conectado

por via rede, existe um servidor Minetest em execução. A aplicação que faz uso do MineScratch se comunica com o servidor por meio do protocolo de rede UDP, enviando e recebendo informações.

A partir dos requisitos funcionais e do desenho geral da arquitetura, foram definidos os seguintes casos de uso (UC) e atores para o MineScratch:

Casos de uso:

- UC001 – Autenticar no Minetest.
- UC002 – Conectar com o Scratch.
- UC003 – Criar instruções Scratch.
- UC004 – Criar comandos Minetest.
- UC005 – Enviar comandos para o Minetest.
- UC006 – Receber comandos do Minetest.
- UC007 – Receber instruções Scratch.
- UC008 – Registrar informações do usuário ao longo da conexão.

Atores:

- **Ator Aplicação**: a aplicação que fará uso da biblioteca. Por exemplo, um jogo educacional para ensino de programação.
- **Ator Scratch**: instância do Scratch que deverá estar funcionando na mesma máquina da aplicação.
- **Ator Servidor Minetest**: instância do servidor Minetest em que a aplicação irá se conectar.

A Figura 18 mostra o diagrama de casos de uso para a biblioteca proposta.

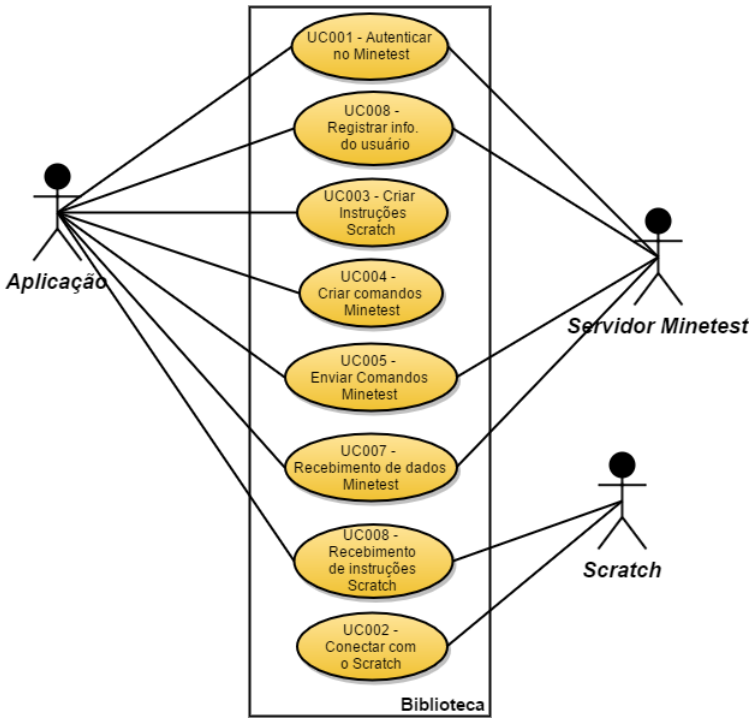


Figura 18 - Diagrama de casos de uso da biblioteca.

Após analisar a complexidade e prioridade dos casos de uso identificados, foi definido como necessário detalhar aqueles casos de uso essenciais e mais complexos. Assim, na sequência são apresentados o detalhamento dos casos UC001, UC002, UC006 e UC007, considerados os mais críticos da biblioteca, pois sem sua total compreensão seria difícil implementar a integração.

Caso de Uso	UC001
Descrição	Autenticar no Minetest
Atores	Aplicação, Servidor Minetest
Pré-condições	N/A
Pós-condições	Aplicação autenticada em um servidor Minetest
Fluxo Base	a) A Aplicação informa para a biblioteca o endereço de um servidor minetest em andamento, além de um usuário e uma senha para conexão b) A biblioteca faz o pedido para conexão (E1) (A1) (A2). c) O Servidor Minetest faz a autenticação e responde com informações acerca da conexão (E2).

	d) O caso de uso é encerrado.
Fluxos Alternativos	A1 - O servidor não é encontrado: a) Informa que um servidor não foi encontrado. b) Volta para o passo "a" do fluxo principal. A2 - O nome de usuário já existe na sessão a) Informa que o nome já está preenchido b) Volta para o passo "a" do fluxo principal.
Fluxos de Exceção	E1 – Não existe conexão de rede: a) Informa à aplicação que não foi possível estabelecer a conexão. E2 – A conexão cai: a) Informa à aplicação que não foi possível estabelecer a conexão.

Quadro 4- Detalhamento do caso de uso UC001 da biblioteca.

Caso de Uso	UC002
Descrição	Conectar com o Scratch
Atores	Scratch
Pré-condições	N/A
Pós-condições	A aplicação está pronta para receber comandos vindos do Scratch
Fluxo Base	a) O Scratch envia para a biblioteca uma requisição HTTP na própria máquina (localhost). b) A biblioteca envia como resultado uma sequência de caracteres pré-definida. c) Em forma de loop, o fluxo volta ao passo “a”, a fim de manter a conexão ativa.

Quadro 5- Detalhamento do caso de uso UC002 da biblioteca.

Caso de Uso	UC006
Descrição	Enviar comandos ao Minetest
Atores	Aplicação, Minetest
Pré-condições	Ter realizado finalizado o caso de uso UC001
Pós-condições	Um comando é enviado ao Minetest

Fluxo Base	a) Por meio de herança de classes, a aplicação ordena que a biblioteca envie um comando ao Minetest b) A biblioteca busca internamente os comandos existentes (A1). c) A biblioteca envia ao Minetest o comando referente à ordem vinda da aplicação (E1). d) O caso de teste é encerrado.
Fluxos Alternativos	A1 – O comando não é encontrado. a) Uma exceção é lançada da biblioteca para a aplicação. b) Volta ao passo “d” do fluxo principal.
Fluxos de Exceção	E1 – A conexão cai: a) Informa à aplicação que o comando não pôde ser enviado. b) Volta ao passo “d” do fluxo principal.

Quadro 6- Detalhamento do caso de uso UC006 da biblioteca.

Caso de Uso	UC007
Descrição	Receber instruções do Scratch.
Atores	Aplicação, Scratch
Pré-condições	Ter realizado a conexão com o Scratch, no caso de uso UC002
Pós-condições	A aplicação é informada de um recebimento de instrução do Scratch.
Fluxo Base	a) O Scratch envia para a biblioteca uma requisição HTTP, contendo informações sobre a instrução desejada b) A biblioteca verifica se a requisição foi feita em um endereço válido, por uma instrução Scratch já criada pela aplicação (A1). c) A aplicação é informada (por meio de herança de classes) que uma notificação foi recebida. d) O caso de teste é encerrado.
Fluxos Alternativos	A1 – O endereço da requisição não é encontrado a) A instrução é ignorada.

Quadro 7- Detalhamento do caso de uso UC007 da biblioteca.

Dada a importância e complexidade do UC001, percebeu-se como necessário um melhor entendimento do fluxo. Assim, a partir de seu detalhamento foi elaborado o caso de uso presente no apêndice 1.

Da mesma forma, percebeu-se necessário o entendimento do fluxo dos casos de uso UC006 e UC007, no entanto estes estão diretamente ligados. Por isso, tomou-

se a decisão de uni-los em um único diagrama de sequência, disponível no apêndice 2. O diagrama de classes da biblioteca se encontra no apêndice 3.

4.3 Implementação da biblioteca

Com base nos requisitos levantados, a biblioteca possui como funcionalidades:

- Fazer autenticação em um servidor Minetest, funcionando assim como um cliente. Ao fim da autenticação é criado no servidor um personagem referente ao cliente e são recebidas informações como:
  - Localização do personagem.
  - Pontos de vida.
  - Horário do dia no cenário.
  - Informações do mapa.
- Comandos básicos referentes ao personagem no Minetest:
  - Andar.
  - Girar no próprio eixo.
  - Teletransportar o personagem.
  - Enviar chat.
  - Clicar com o mouse na frente do personagem.
- Possibilidade de criação de novos comandos no Minetest.
- Criação de instruções a serem recebidas pelo editor do Scratch.

Atendendo ao requisito de composição (MEYER, 1988) a biblioteca é dividida basicamente em três módulos, explicados com mais detalhes a seguir:

- Gerenciador da comunicação com o Minetest.
- Gerenciador da comunicação com o Scratch.
- Funções úteis.

Cada módulo foi construído de forma autocontida, de forma que podem interagir com os outros sem expor seus componentes internos.

4.3.1 Módulo Minetest

Na comunicação com o Minetest existe a classe abstrata *AbstractMinetest*, que é a classe principal deste módulo. Ela possui um receptor que, ao ser ativado, fica ouvindo indefinidamente as mensagens provenientes do servidor Minetest. Ela também possui uma classe responsável pelo envio de pacotes ao servidor, lidando com toda a regras as regras impostas pela API de rede do Minetest.

Este módulo possui uma classe *Character*, onde as informações sobre o personagem são armazenadas e modificadas. Essas informações são recebidas do servidor ao fim da autenticação, e alteradas toda vez que um comando de movimento é enviado.

Ele ainda possui a classe abstrata *Command*, que possui o método abstrato *execute*. Este método deve ser implementado de forma a enviar os dados necessários ao servidor para realizar determinada ação no Minetest. Por padrão foi criado um conjunto de comandos iniciais de forma a facilitar a utilização da biblioteca, evitando inicialmente o contato com a API do Minetest. Esses comandos são:

- Andar: mover o personagem para frente.
- Girar: girar o personagem no eixo.
- Enviar mensagem: enviar mensagem de texto ao servidor Minetest conectado.

Para que desenvolvedores possam interagir de outras formas com o Minetest, como alterando o cenário, por exemplo, é necessária a criação de uma classe concreta que estenda a classe abstrata *Command*, e definindo os parâmetros que devem ser enviados ao servidor Minetest.

Por conta da arquitetura adotada pelo Minetest, foi identificado um problema para a construção da biblioteca. No software Minetest, quando há um servidor em andamento e um cliente conectado, o tráfego de informações referentes ao personagem é no sentido cliente-servidor. Dessa forma, o processamento das ações do personagem no cenário é feito apenas no cliente, que envia o resultado para o servidor. A Figura 19 traz uma representação em alto nível da sequência de ações supracitadas.

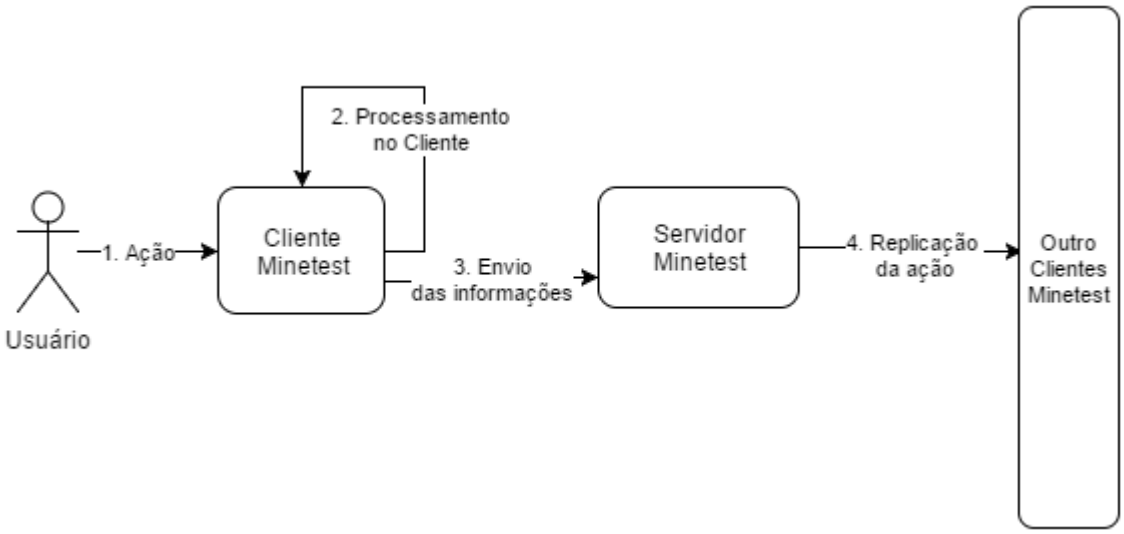


Figura 19- Fluxo geral de comunicação na arquitetura do Minetest

O fluxo a presentado na Figura 19 consiste em:

1. Alguma ação do usuário do Minetest (movimento, clique do mouse, etc.).
2. Após a ação do usuário, seu processamento é feito diretamente no cliente, alterando os dados do cenário internamente. Dessa forma, é o próprio cliente que gerencia se a ação do usuário é possível, como por exemplo se movimentar para uma coordenada que contém uma parede.
3. O cliente envia ao servidor Minetest o resultado do processamento, como a nova localização, alterações no cenário, interações.

4. O servidor faz a replicação das alterações nos demais clientes conectados.

Devido a essa arquitetura de rede do Minetest, pontos importantes como gravidade e colisões são gerenciados no cliente, algo que seria difícil de reproduzir na biblioteca, pois o processamento no cenário deveria então ser feito por ela. A solução adotada foi a possibilidade da criação de um arquivo de propriedades no formato JSON onde o desenvolvedor pode inserir informações importantes acerta do cenário do jogo. Essas informações são:

- **Initial:** Coordenadas para onde os personagens serão transportados ao fim da autenticação.
- **Allowed:** Coordenadas por onde o personagem pode andar.
- **Denied:** Coordenadas por onde o personagem não pode andar.
- **Objective:** Coordenadas que representam algum objetivo do jogo.

Cada coordenada deve estar contida em uma lista de números no arquivo JSON e seguir, com estes números representando os eixos X, Y e Z respectivamente.

O arquivo, se utilizado, deve possuir o nome *properties.json* e estar localizado no diretório raiz da aplicação. Na Figura 20 é apresentada um exemplo de conteúdo que pode ser inserido no arquivo de propriedades.

```
{
  "initial": [
    [-40, 61.5, 9]
  ],
  "allowed":[
    [-39, 61.5, 16]
  ],
  "objective": [
    [-42, 61.5, 15]
  ],
  "Denied": [
    [-44, 61.5, 9]
  ]
}
```

Figura 20 - Exemplo de código permitido no arquivo properties.json

4.3.2 Módulo Scratch

Para a comunicação com o Scratch, primeiramente é utilizada a biblioteca padrão do Java para recebimento e envio de pacotes, java.net[15]. Porém, por conta da complexidade, o código foi modificado, criando um servidor HTTP utilizando a biblioteca Apache Commons[16].

O módulo possui a classe abstrata *ScratchServer*, um servidor que ao ser acionado, se mantém ouvindo requisições na porta de rede 50210.

Neste módulo também é possível criar instruções novas, que são basicamente caminhos de rede que o *ScratchServer* deve ouvir, referente a instruções importadas na extensão HTTP do editor do Scratch. Essa criação se dá a partir da classe abstrata *Instruction*, que possui o método abstrato *execute*. Ele define o que deve ser feito quando for feita uma requisição no caminho definido pela instrução.

O código-fonte da biblioteca de integração está disponível em https://github.com/jhonata11/MineScratch.

[15] https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html
[16] https://commons.apache.org/

Com o desenvolvimento da biblioteca finalizado, é possível desenvolver um jogo educacional para o ensino de programação utilizando-a. Este desenvolvimento é apresentado no capítulo seguinte.

5. Desenvolvimento do jogo educacional

Este capítulo aborda o desenvolvimento do protótipo de jogo educacional construído utilizando a MineScratch. Seguindo o processo ENgAGED, as etapas apresentadas neste trabalho são:

1. Análise da unidade instrucional: levantamento dos dados necessários para a construção da unidade instrucional.
2. Projeto da unidade instrucional: planejamento da unidade instrucional, definindo o conteúdo e a ordem que ele deve ser apresentado.
3. Desenvolvimento do jogo: apontando o desenvolvimento do jogo utilizado na unidade instrucional.

Por se tratar do protótipo de um jogo, as etapas 1 e 2 são apresentadas em conjunto.

5.1 Análise/Projeto da Unidade Instrucional

Seguindo o processo ENgAGED, inicialmente é definido o objetivo do jogo. Este objetivo é a utilização do jogo dentro de uma unidade instrucional para o ensino de programação. Levando isso em consideração, assume-se que o público alvo sejam crianças e jovens de até 13 anos que já possuam contato com jogos digitais. O Quadro 8 apresenta a caracterização do seu público alvo.

Formação	Estudantes de ensino fundamental.
Faixa etária	Abaixo de 13 anos
Gênero	Ambos
Preferência de gêneros de jogo	Ação/aventura, *role-playing game*
Preferência de plataformas de jogo	Consoles ou computadores
Frequência que joga jogos digitais	Ao menos uma vez por semana.

Frequência que joga jogos não-digitais	Raramente
Jogos favoritos	Variados
Persona	Nome: Pedro Idade: 13 anos Descrição: criança do sexo masculino, jogador frequente. Estudante do ensino fundamental no colégio Energia em Florianópolis. Possui experiência com a utilização de computadores. Objetivo: aprender programação.

Quadro 8- Caracterização de aprendizes

A análise do contexto é apresentada no Quadro 9.

Análise do contexto	
Local de aplicação	O jogo será desenvolvido com objetivo de ser aplicado em laboratórios de informática de escolas de ensino fundamental
Recursos	Serão necessários ao menos 3 computadores que atendam aos requisitos de funcionamento do *Minetest* e do *Scratch* simultaneamente.
Sistema Operacional	Computadores que possuam Windows nas versões 7, 8, 8.1 ou 10 e Linux Ubuntu nas versões a partir de 14.04
Internet	Não é necessário a existência de internet, mas é necessário a existência de uma rede entre os computadores.
Recursos financeiros disponíveis	Serão apenas necessários os recursos humanos para o desenvolvimento deste jogo.

Quadro 9 - Análise do contexto

O jogo é projetado para ser aplicado em laboratórios de informática de escolas, para alunos do ensino fundamental. Seu objetivo é o ensino de lógica de programação, e já que este conteúdo não está presente na grade curricular obrigatória da maioria das escolas, assume-se que sua utilização seria em uma oficina ou curso oferecido pela escola.

A unidade instrucional possui como conteúdo programático alguns dos conceitos de lógica de programação. Sua especificação é apresentada no Quadro 10.

Informações específicas da unidade instrucional	
Nome	MineScratch no auxílio de programação.
Pré-requisito	Contato prévio com jogos digitais.
Conteúdo programático	Utilização de conceitos de lógica de programação: - Construção de algoritmos. - Encadeamento de instruções. - Estruturas de repetição. - Criação de métodos. - Parâmetros.
Duração	Aproximadamente 1 hora
Ambiente	Laboratório de informática

Quadro 10 - Informações específicas da Unidade Instrucional

Apesar de que, para este trabalho, não será desenvolvida uma unidade instrucional completa, é importante definir os objetivos para que o jogo possa ser implementado. São definidos como objetivos gerais da unidade instrucional:

- É esperado que o aluno consiga se lembrar dos conceitos apresentados na unidade instrucional, definidos no Quadro 10.
- O aluno deve compreender o funcionamento e a utilização dos conceitos apresentados na unidade instrucional, definidos no Quadro 10.
- A aluno deve ser capaz de aplicar estes conceitos em um outro ambiente, como no aprendizado de uma linguagem de programação, por exemplo.

Conforme os objetivos definidos, o conteúdo programático da unidade instrucional são os elementos de programação e algoritmos, sendo estes:

- Desvios condicionais simples.
- Desvios condicionais compostos.
- Operadores lógicos.

- Estruturas de repetição:
    - For (por);
    - While (enquanto);
- Atribuição de variáveis.

Por não ser o foco do presente trabalho, a sequência de apresentação do conteúdo e as estratégias da unidade instrucional não foram estabelecidas.

5.2 Desenvolvimento do jogo educacional

Esta seção apresenta a etapa de desenvolvimento da metodologia ENgAGED no contexto do jogo educacional que utiliza a biblioteca, seguindo o ciclo de vida em cascata de desenvolvimento de softwares. Nela são definidos os requisitos do jogo, definindo assim as regras do jogo e apresentando as etapas de sua construção.

5.2.1 Análise de Requisitos

Para atender às definições instrucionais estabelecidas na etapa do projeto do jogo educacional, foram identificados os elementos que devem ser desenvolvidos para a criação do jogo.

- Um conjunto de novas instruções no Scratch.
- Uma aplicação que utilize a biblioteca MineScratch para fazer a integração Scratch-Minetest.
- Um *mod* para o Minetest incluindo novas características ao jogo:
    - Um novo cenário onde todas as partidas serão realizadas.
    - Um bloco objetivo.
    - Um arquivo de *script* gerencia a pontuação dada a cada jogador.

A partir dos elementos necessários para utilizar a biblioteca de integração e atingir os objetivos educacionais propostos, são identificados os requisitos do jogo educacional que utiliza a biblioteca MineScratch, apresentados no Quadro 11.

Requisito	Descrição	Prioridade		
		Essencial	Importante	Desejável
RF01	Permitir o controle de um personagem no Minetest.	X		
RF02	Os controles definidos em RF01 são: • Andar para frente • Girar o personagem • Clicar no bloco da frente do personagem.	X		
RF03	Ao clicar no bloco objetivo, é dada uma pontuação ao jogador.	X		
RF04	Os controles definidos em RF02 devem ser executados através de algoritmos desenvolvidos no Scratch.	X		
RF05	O jogo deve possibilitar o login em um servidor previamente definido do Minetest.	X		
RNF01	Deve ser um jogo digital para a plataforma computador.	X		

RNF02	O jogo deve utilizar a biblioteca MineScratch.	X		
RNF04	O MineScratch deve ser compatível com as plataformas Windows e Linux Ubuntu em versões posteriores a 14.04.			X
RNF05	A inicialização do jogo deve ser feita por meio de uma interface gráfica.		X	
RNF06	O mapa deve ser construído de forma que diferentes comandos do Scratch devem ser executados para alcançar o objetivo.	X		

Quadro 11- Requisitos funcionais e não funcionais do jogo

O jogo é projetado para ser jogado por múltiplos jogadores, que devem movimentar seus personagens pelo cenário a fim de chegar antes que os outros em uma determinada posição. A partir disto foi elaborada a concepção do jogo, apresentada no Quadro 12, junto com suas regras.

Objetivos do jogo	O jogador deve levar seu personagem até determinada parte do cenário.
Gênero do jogo	Estratégia.
Modo de interação entre jogadores	Competição multilateral, de forma que cada jogador é um adversário de todos os outros

Regras do jogo:	1. Ao entrar em uma partida, o jogador é transportado a uma posição aleatória do cenário. 2. O jogador deve movimentar seu personagem utilizando blocos de instruções no Scratch. 3. Ele deve percorrer o cenário a fim de alcançar o bloco objetivo. 4. Ao clicar no bloco objetivo, o jogador recebe uma pontuação de 100 pontos. 5. Os demais jogadores que atingirem o objetivo recebem 15 pontos a menos que o anterior (85, 70, 55, etc.). 6. A pontuação mínima que um jogador pode atingir é 10. 7. Quando o personagem do jogador morre, ele é movido para a sua posição inicial, podendo tentar novamente. 8. Ao final de 5 rodadas, o jogador com maior pontuação vence.

Quadro 12- Concepção do jogo

O projeto do mapa onde o jogo ocorre pode ser visto na Figura 21

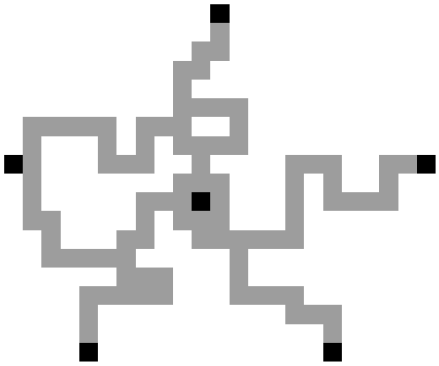


Figura 21- Projeto do mapa.

5.2.2 Design do Jogo

O jogo foi desenvolvido utilizando a biblioteca MineScratch para fazer a integração, fazendo uso de suas funções e implementando interfaces específicas. Ele possui a mesma arquitetura apresentada na Figura 17.

O jogo é um sistema que contém o Minetest, o Scratch e uma aplicação que fará a integração entre eles. Para esta aplicação, a partir dos requisitos funcionais e da concepção do jogo, foram identificados os casos de uso:

- UC001 - Entrar em uma partida.
- UC002 - Sair de uma partida.
- UC003 - Movimentar o personagem.
- UC004 - Enviar mensagens para o servidor.
- UC005 - Interagir com o cenário.

Da mesma forma, foram identificados os atores:

- Usuário.
- Biblioteca.

Estes casos de uso são apresentados no diagrama de casos de uso na Figura 22.

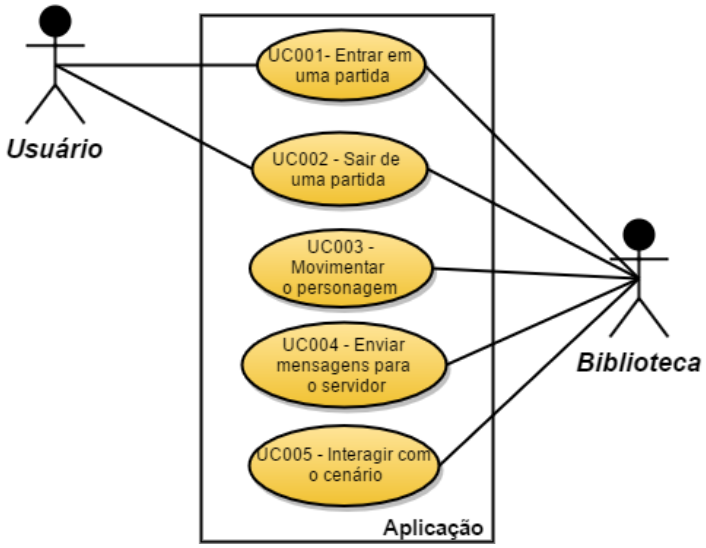


Figura 22 - Diagrama de casos de uso da aplicação.

Do Quadro 13 ao Quadro 15 são apresentados os detalhamentos dos casos de uso da aplicação, com exceção do UC004 e UC005 pois estes são meramente o uso de funções disponíveis na biblioteca.

Caso de Uso	UC001
Descrição	Entrar em uma partida.
Atores	Usuário, Biblioteca
Pré-condições	N/A
Pós-condições	O jogador estará em uma partida, em uma das posições iniciais.
Fluxo Base	a) O usuário informa um nome de usuário, senha, o endereço de um servidor Minetest e a porta. b) A aplicação seleciona uma das coordenadas possíveis. c) A aplicação envia à biblioteca as informações do usuário e a coordenada selecionada (A1) (A2) (E1). d) O caso de uso é encerrado.
Fluxos alternativos	A1 – A senha não corresponde: a) O jogador não é conectado. b) Volta ao passo “a” do fluxo principal. A2 – O nome de usuário já existe: a) O jogador não é conectado. b) Volta ao passo “a” do fluxo principal.

Fluxos de exceção	E1 – O endereço do servidor Minetest não é encontrado: a) O jogador não é conectado. b) Volta ao passo “a” do fluxo principal.

Quadro 13 - Detalhamento do caso de uso UC001 da aplicação.

Caso de Uso	UC002
Descrição	Sair de uma partida.
Atores	Usuário, Biblioteca.
Pré-condições	Ter concluído o caso de uso UC001
Pós-condições	O jogador terá saído de uma partida existente.
Fluxo Base	a) O usuário ativa a opção de sair de uma partida. b) A aplicação envia para a biblioteca uma requisição para encerrar a conexão. c) O caso de uso é encerrado.

Quadro 14 - Detalhamento do caso de uso UC002 da aplicação.

Caso de Uso	UC003
Descrição	Movimentar o personagem.
Atores	Usuário, Biblioteca.
Pré-condições	Ter concluído o caso de uso UC001
Pós-condições	A posição do personagem será alterada.
Fluxo Base	a) A biblioteca envia uma requisição do Scratch para movimentação do personagem, no *localhost*. b) A aplicação verifica a posição atual do personagem. c) A aplicação envia uma requisição à biblioteca para movimentar o personagem no Minetest. d) O caso de uso é encerrado
Fluxo de Exceção	E1 – A biblioteca lança exceção de servidor não encontrado: a) A aplicação exibe a informação na tela de logs. b) Volta ao passo “d” do fluxo principal.

Quadro 15 - Detalhamento do caso de uso UC003 da aplicação.

Em seguida é apresentado o diagrama de sequência do caso de uso UC001, considerado o mais crítico, na Figura 23.

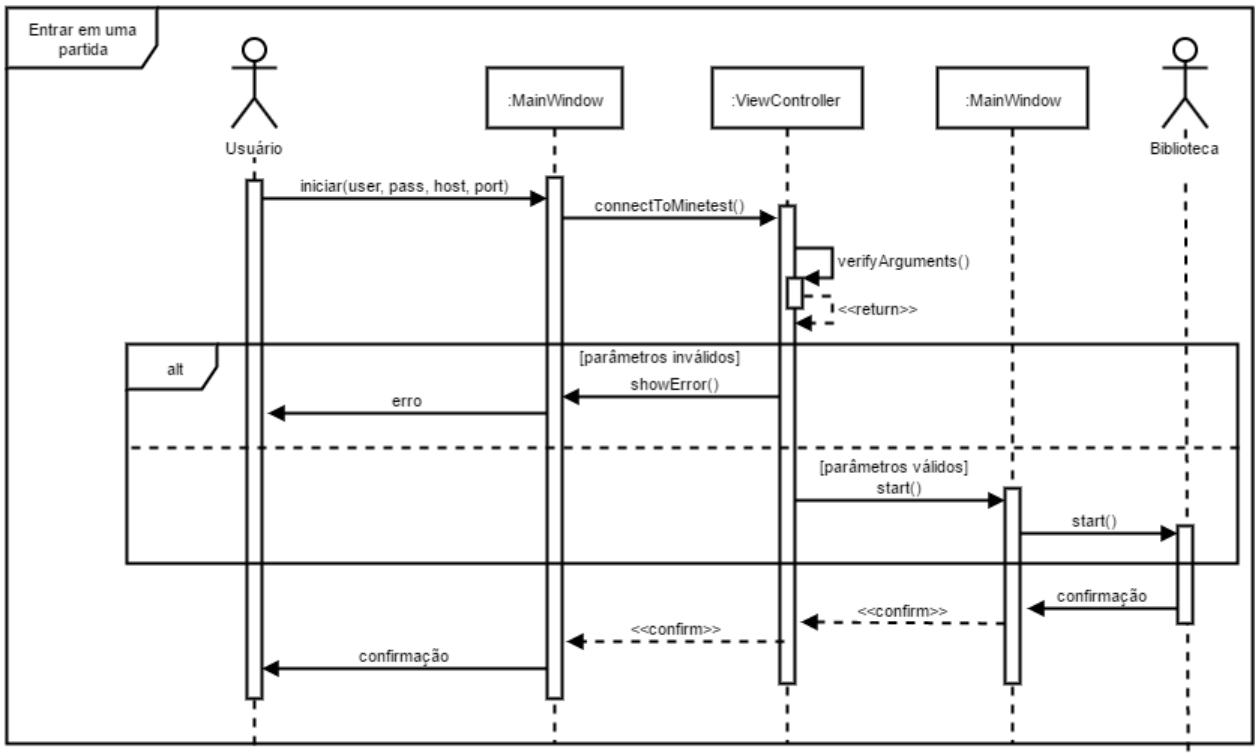


Figura 23 - Diagrama de sequência do caso de uso UC001 da aplicação.

A partir dos projetos apresentados no capítulo 4 foi feita a implementação do MineScratch e do jogo educacional, detalhada no capítulo 5.

A biblioteca utiliza como referência o Minetesting[17], uma implementação do protocolo de comunicação do Minetest realizada de forma experimental em Python[18] no grupo de pesquisas do INCoD[19]. Os motivos para o uso desta implementação como referência são a pobre documentação do protocolo de rede oficial do Minetest, assim

[17] https://github.com/boppreh/minetesting
[18] https://www.python.org/
[19] http://www.incod.ufsc.br/

como a falta de familiaridade do autor com a linguagem de programação utilizada no Minetest, C++.

O resultado da etapa de implementação deste trabalho foi:

- Uma biblioteca contendo um módulo de integração entre o Scratch e o Minetest, chamada MineScratch.
- Um jogo educacional que a utilize, composto por:
  - Uma extensão HTTP criada para o Scratch.
  - Uma *mod* criado para o Minetest.
  - Uma aplicação que integra todos os elementos previamente descritos.

5.2.3 Desenvolvimento do jogo

Conforme já mencionado, um protótipo de um jogo educacional voltado ao ensino de programação é criado com objetivo de testar a biblioteca MineScratch. Ele segue a modelagem estabelecida na fase de concepção do jogo descritas na seção anterior. O jogo é um sistema que consiste em:

- Um computador com o Minetest instalado funcionando como servidor, contendo:
  - Um mapa específico onde as partidas ocorrem.
  - Um *mod* para gerenciar a pontuação dos jogadores em cada partida.
- Até 5 computadores, cada um contendo:
  - Uma instância do Scratch funcionando.
  - Uma extensão HTTP para o Scratch.

- Uma aplicação que fará a comunicação entre o Scratch local e o servidor Minetest.
- Um arquivo JSON contendo as informações do mapa.

Inicialmente foi desenvolvido o mapa para o Minetest. Ele contém uma série de coordenadas onde o personagem deve iniciar o jogo. Também existe um ponto no mapa chamado de objetivo, onde os jogadores devem chegar a fim de alcançar uma pontuação alta, acumulando pontos. As coordenadas iniciais são separadas do objetivo por 22 blocos. Ou seja, sem errar movimentos, o jogador precisa se movimentar 22 vezes antes de alcançar o objetivo. A Figura 24 mostra o mapa criado para o jogo.

Figura 24 - Mapa criado para o jogo.

As coordenadas do mapa foram registradas no arquivo *properties.json*, e antes de movimentar o personagem, o jogo verifica se este é um movimento válido (se existe chão na coordenada onde o personagem escolheu se movimentar). Caso não seja, o personagem volta ao bloco inicial, tendo que começar de novo.

O bloco objetivo é representado por um baú no centro do mapa, que, ao ser clicado, atribui uma pontuação ao jogador. Isso é possível por conta de um script desenvolvido para o Minetest, na linguagem de programação Lua.

Na aplicação Java, que faz uso do MineScratch, são implementadas as classes e métodos abstratos provenientes da biblioteca. Ela é gerenciada por uma interface gráfica, em que o usuário pode iniciar a partida, finalizá-la. Por se tratar do protótipo de jogo, existe um painel que mostra logs do sistema, como instruções recebidas do Scratch, dados enviados ao Minetest, etc. A Figura 25 mostra a interface gráfica da aplicação.

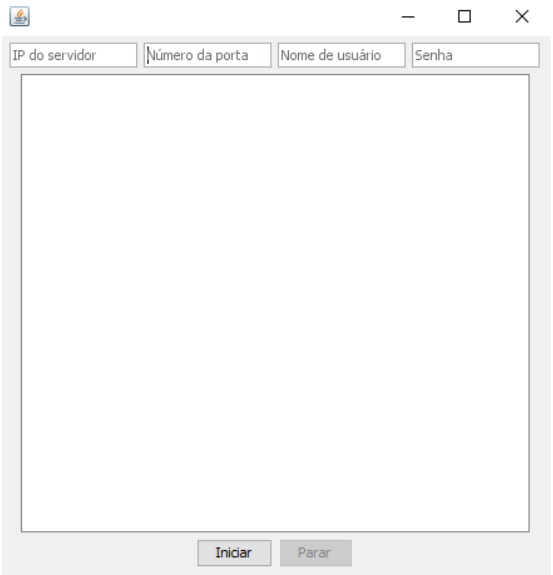


Figura 25- Interface gráfica da aplicação.

Para o Scratch, foi criada uma extensão HTTP contendo 3 ações que o jogador pode realizar no jogo. Estas ações, ao serem importadas pelo Scratch, dão origem aos blocos apresentados na Figura 26.

Figura 26 - Blocos de ações do jogo.

Na aplicação Java, foi criado o comando *MinetestClick*, que envia ao servidor Minetest um click no bloco imediatamente a frente do personagem. É com este comando que o objetivo é clicado no servidor.

Neste ponto, o sistema já representa um jogo funcional, contendo um servidor Minetest onde as partidas ocorrem e clientes que se conectam nele, além de instâncias do Scratch, que manipulam as ações do personagem no cliente.

O jogo pode ser encontrado no endereço https://github.com/jhonata11/MineScratch.

Assim, no capítulo seguinte é apresentada a avaliação da biblioteca proposta.

6. Aplicação e Avaliação

Neste capítulo é apresentada a aplicação e avaliação da biblioteca. O objetivo deste trabalho é a construção de uma biblioteca que integra o Scratch e o Minetest, por isso, avaliação da biblioteca é feita por meio de sua utilização em um jogo educacional para o ensino de programação. A partir desta aplicação é feita uma avaliação seguindo a abordagem GQM – *Goal/Question/Metric* (BASILI, 1994), uma abordagem de medição orientada a metas e medidas coletáveis, auxiliando na avaliação.

Na abordagem GQM devem ser definidos:

1. Objetivos da medição.
2. Perguntas que atendam o objetivo ao serem respondidas.
3. Para cada pergunta, definir as medidas a serem coletadas.

Considerou-se importante avaliar a biblioteca sob os aspectos de:

- Eficiência: ou seja, o desempenho de um software com relação ao tempo em comparação à quantidade de recursos usados por ele (ROCHA, 2001);
- Robustez: representa a capacidade de um software funcionar em condições adversas (FERNANDEZ et al., 2008);
- Funcionalidade: que representa se o conjunto de funcionalidades oferecidas pelo software satisfaz as necessidades do usuário (ROCHA, 2001).

Seguindo a abordagem GQM, são definidos assim os objetivos de medição sob o ponto de vista do desenvolvedor de software que utiliza a biblioteca:

- **Objetivo 1**: avaliar a eficiência da biblioteca e de sua utilização na construção de softwares no contexto de um jogo para ensino de programação sob o ponto de vista de um usuário do jogo.
- **Objetivo 2**: avaliar a robustez da biblioteca no contexto de um jogo para ensino de programação sob o ponto de vista de um usuário do jogo.
- **Objetivo 3**: avaliar a funcionalidade no contexto de um jogo para ensino de programação sob o ponto de vista de um usuário do jogo.

De acordo com a abordagem GQM, para cada objetivo de avaliação devem ser definidas perguntas e medidas.

Conforme se pode perceber pelas perguntas apresentadas no Quadro 16, os objetivos de medição são avaliados de forma indireta, pelo uso da biblioteca no contexto de um jogo, uma vez que ela não possui interface amigável para ser utilizadas por um usuário. Assim, as perguntas são direcionadas aos utilizadores (alunos) do jogo educacional que utiliza o MineScratch, avaliando-a de forma indireta.

As perguntas e medidas referentes ao objetivo 1 são apresentadas no Quadro 16.

Objetivo 1	Avaliar a eficiência da biblioteca e de sua utilização na construção de softwares, do ponto de vista do desenvolvedor.
Comportamento em relação ao tempo	
Pergunta Q1	No jogo início do jogo, quanto tempo levou, do momento do preenchimento dos dados de login até o personagem estar conectado no Minetest?
Medidas:	MQ1.1 - Tempo medido pelo usuário para realizar a autenticação no servidor Minetest. MQ1.2 - Tempo medido pelo usuário para realizar a conexão com o servidor Scratch.

	MQ1.3 - Tempo coletado nos logs da biblioteca para realizar a autenticação no servidor Minetest. MQ1.4 - Tempo coletado nos logs da biblioteca para realizar a conexão com o servidor Scratch.
Pergunta Q2	Durante a partida, qual era o tempo entre o envio de uma instrução do Scratch até o Minetest?
Medidas:	MQ2.1 - Tempo de recebimento de uma instrução do Scratch. MQ2.2 - Tempo de envio de comando específico de movimento. MQ2.3 - Tempo coletado nos logs da biblioteca para receber instruções do Scratch. MQ2.4 – Tempo coletado nos logs da biblioteca para o envio de comandos ao Minetest.
Utilização de recursos	
Pergunta Q3	Quanta memória era utilizada pela aplicação antes da partida ser iniciada?
Medidas:	MQ3.1 Uso de memória quando inativo.
Pergunta Q4	Quanta memória era utilizada após o início da partida?
Medidas:	MQ4.1 Uso de memória quando ativo.
Pergunta Q5	Qual o uso da CPU pela aplicação antes da partida ser iniciada?
Medidas:	MQ5.1 Uso de CPU quando inativo (antes da conexão com o Minetest).
Pergunta Q6	Qual o uso da CPU pela aplicação após o início da partida?
Medidas:	MQ6.1 Uso de CPU quando ativo.

Quadro 16 - Perguntas e medidas para o objetivo de medição 1.

Na sequência são apresentadas as perguntas e medidas referentes ao objetivo de medição 2, no Quadro 17.

Objetivo 2	Avaliar a robustez da biblioteca e de sua utilização na construção de softwares, do ponto de vista do desenvolvedor.
Perguntas	

Pergunta Q7	O jogo apresentou alguma falha durante sua execução?
Medidas	MQ7.1 Existência de falhas durante a execução.
Pergunta Q8	O sistema se comportou de forma positiva quando a falha ocorreu?
Medidas	MQ8.1 Tolerância a falhas.

Quadro 17 - Perguntas e medidas para o objetivo de medição 2.

Na sequência são apresentadas as perguntas referentes ao objetivo de funcionalidade, no Quadro 18.

Objetivo 3	Avaliar o critério de funcionalidade da biblioteca e de sua utilização na construção de softwares, do ponto de vista do desenvolvedor.
Perguntas	
Pergunta Q9	As funcionalidades oferecidas pela integração entre Scratch e Minetest são suficientes?
Medidas	MQ9.1 Funcionalidades oferecidas.
Pergunta Q10	Você sentiu falta de alguma funcionalidade na integração entre Scratch e Minetest?
Medidas	MQ10.1 Ausência de funcionalidades.

Quadro 18 - Perguntas e medidas para o objetivo de medição 3.

Adicionalmente são acrescentadas as perguntas apresentadas no Quadro 19.

Perguntas	
Pergunta Q11	Cite os pontos positivos sobre a integração entre o Minetest e o Scratch.
Medidas	MQ11.1 - Pontos positivos da integração.
Pergunta Q12	Cite os pontos negativos sobre a integração entre o Minetest e o Scratch
Medidas	MQ12.1 – Pontos negativos da integração.

Quadro 19 - Perguntas e medidas adicionais.

A partir das perguntas e medidas apresentadas no Quadro 16, foi elaborado o questionário Q-01, disponível no apêndice 4. Todas as medidas são coletadas pelo autor do trabalho através do questionário Q-01.

6.1 Aplicação

A aplicação realizada no laboratório de Materiais Magnéticos, na Universidade de Federal de Santa Catarina, pois o autor possui acesso facilitado, e este possui computadores disponíveis para o uso.

Com o interesse de aplicar a integração com usuários que não fossem, necessariamente, da área de computação, para evitar a tendência na avaliação, foram buscados possíveis usuários de outros cursos. Por proximidade do autor, a amostra de usuários para a aplicação foi selecionada dentre alunos do curso de engenharia de materiais.

A aplicação seguiu a seguinte estrutura:

- 3 jogadores, todos estudantes de engenharia de materiais na Universidade de Federal de Santa Catarina. Apenas 1 possuía contato prévio com programação, em nível básico. Todos possuíam a disposição um computador individual. Cada computador possuía o Scratch, o Minetest, a aplicação que utiliza a biblioteca MineScratch, e as extensões desenvolvidas para o jogo: *mod* e mapa do Minetest, e extensão HTTP do Scratch.
- 1 computador que fazia o papel de servidor. Ele possuía apenas o Minetest instalado.
- Todos os computadores estavam conectados em uma mesma rede.

Antes do início das partidas foi dada uma breve introdução sobre o Scratch aos participantes, mostrando os tipos de instruções existentes, e como elas funcionam.

Foram realizadas 3 partidas, que ocorreram da seguinte forma:

1. Na primeira execução, nenhum computador possuía o Minetest em execução, com exceção do servidor. Dessa forma, cada jogador só podia visualizar seus respectivos personagens através da tela do servidor. Isso não se mostrou muito prático, pois o servidor possuía uma tela pequena, e se tornou difícil visualizar os personagens. É possível que em um laboratório de informática, ou em uma sala de aula em que o servidor esteja ligado a um projetor, esse problema se solucione.
2. Em uma segunda execução, para solucionar o problema de visualização, cada jogador executou em sua máquina o Minetest, com a intenção apenas de visualizar seu personagem no jogo. Esta execução aconteceu de forma livre, sem que os jogadores se preocupassem em observar os dados referentes ao questionário. A Figura 27 apresenta uma imagem da segunda execução.

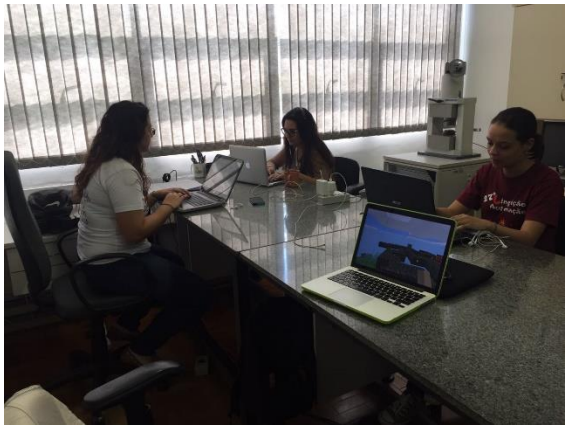


Figura 27 - Execução de uma partida do jogo educacional que utiliza a MineScratch

3. Na terceira execução, os jogadores foram orientados a anotar as informações referentes ao questionário. A rede onde os computadores se conectavam foi propositalmente desligada, a fim de verificar o comportamento do sistema durante as falhas. Os jogadores não foram informados que isto aconteceria.

Ao final da última execução, foi entregue aos jogadores o questionário Q-01.

6.2 Avaliação

As medidas coletadas para responder às perguntas, são apresentas a seguir, bem como a sua análise e interpretação, separadas de acordo com o objetivo de medição definido.

6.2.1 Objetivo 1: **eficiência**.

Medidas: MQ1.1, MQ1.2, MQ2.1 e MQ2.2,

As medidas foram obtidas durante a execução da aplicação pelos participantes da avaliação. A aplicação possui uma tela que informa sempre que uma instrução é recebida do Scratch e sempre que um comando é enviado ao Minetest, sendo assim mais fácil separar o tempo de comunicação com cada módulo. As medidas são apresentadas na Figura 28.

Verifica-se que são necessários em média 2,19 segundos para completar a conexão com os softwares, e em média 3,64 segundos do momento que uma instrução era enviada do Scratch até que ela surtisse efeito no Minetest.

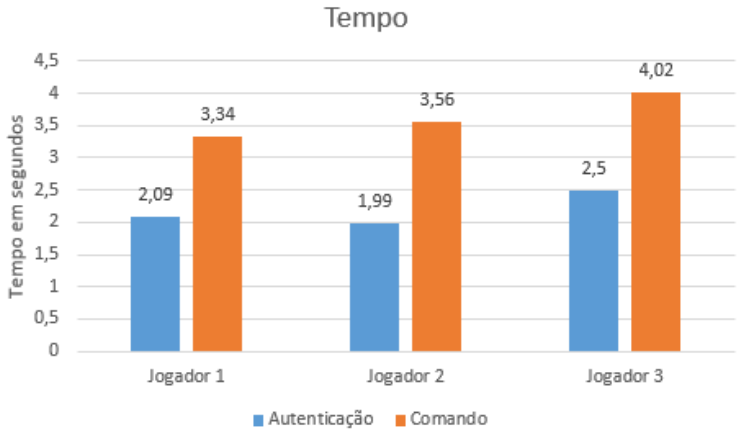


Figura 28 - Respostas referentes ao tempo de execução.

Medidas: MQ1.3 e MQ1.4, MQ2.3 e MQ2.4.

Com intenção de obter mais dados, as medidas foram obtidas em um ambiente de testes controlado, sem qualquer relação à execução previamente descrita.

O ambiente de testes possui um computador com o servidor Minetest e um único computador conectado a este servidor. Os dados foram coletados por meio dos logs da aplicação e do Minetest. 8 execuções foram feitas e os resultados são apresentados na Figura 29 e Figura 30. É possível observar um tempo maior no módulo do Minetest, tanto na autenticação quanto na tradução de comandos.

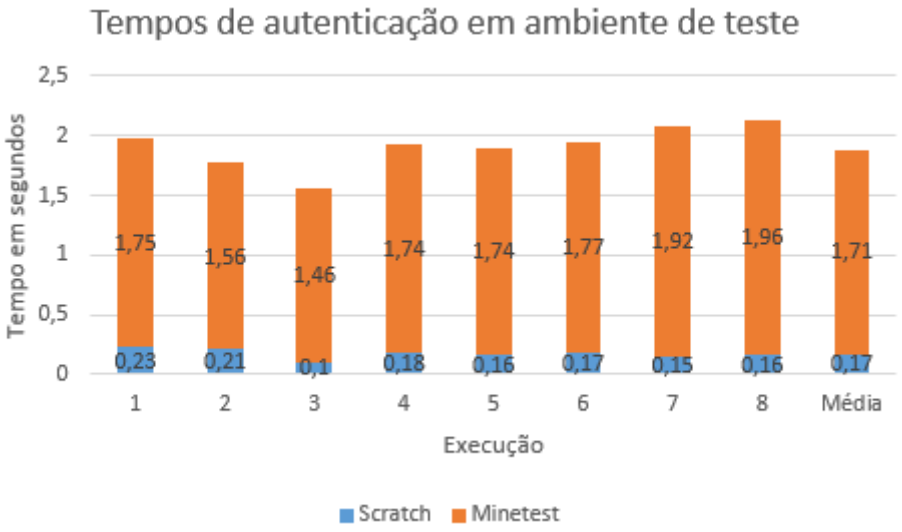


Figura 29 - Medidas do tempo de autenticação em ambiente de teste.

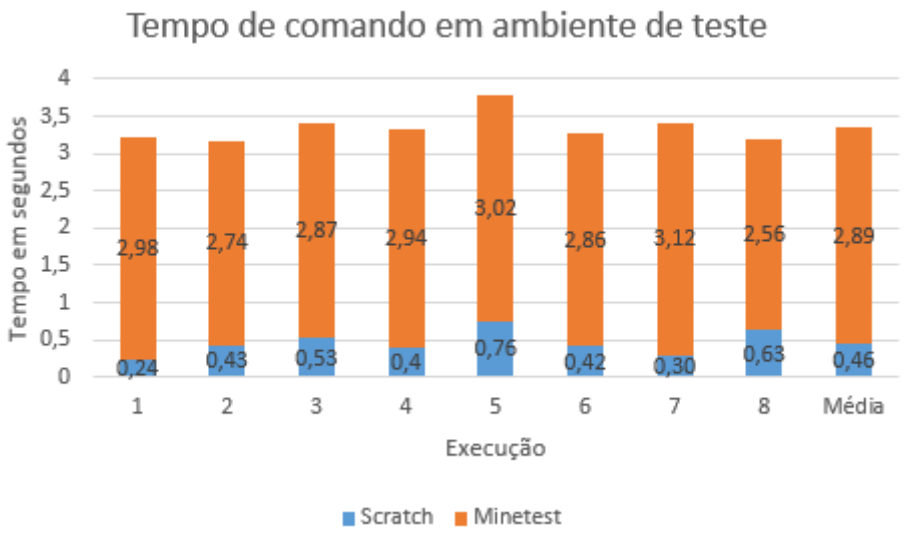


Figura 30 - Medidas do tempo de execução de comandos em ambiente de teste

Uma hipótese para essa maior eficiência do módulo de comunicação com o Scratch é a utilização da biblioteca Apache Commons, que faz todo o gerenciamento das requisições HTTP de forma independente. Para o Minetest, toda a comunicação era feita por envio e recebimento de pacotes da rede, em baixo nível de abstração, lidando com concorrência de threads. Da forma como foi construída, a biblioteca está a todo tempo ouvindo o servidor Minetest, e ao ter que enviar informações a ele, é necessário interromper a thread que está ouvindo. É possível que a aplicação esteja perdendo tempo nesse interrupção e ativação de thread.

Medidas: MQ3.1 e MQ4.1

Os resultados coletados acerca do uso de memória são apresentados na Figura 31. O termo *inativo* se refere a antes da aplicação ter iniciado a conexão com os softwares Scratch e Minetest, já o termo *ativo* se refere a após esta situação. É possível ver que o sistema não possui problemas com o uso de memória, tanto com a aplicação ativa quanto inativa.

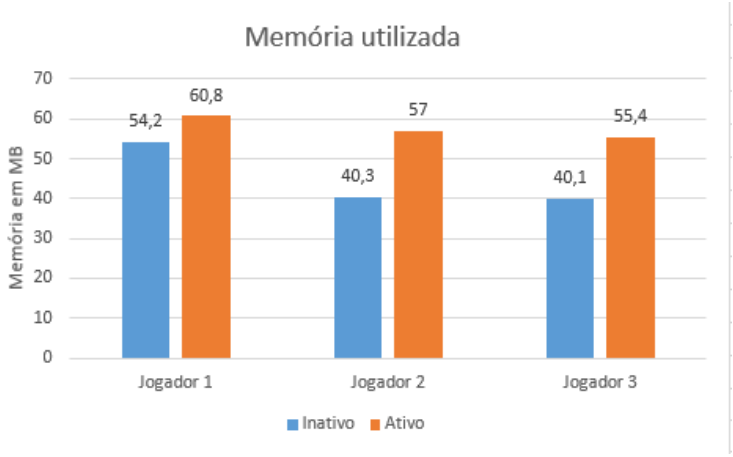


Figura 31 - Respostas referentes ao uso de memória.

Medidas: MQ5.1 e MQ6.1

Os resultados referentes ao uso de CPU pela aplicação, tanto com ela ativa quanto inativa, são exibidos na Figura 32. Diferente da memória, nota-se que o uso de CPU é relativamente alto após o início da partida. Uma hipótese para este alto consumo é o fato da biblioteca possuir um servidor HTTP em funcionamento para se comunicar com o Scratch, além de constantemente monitorar o servidor Minetest.

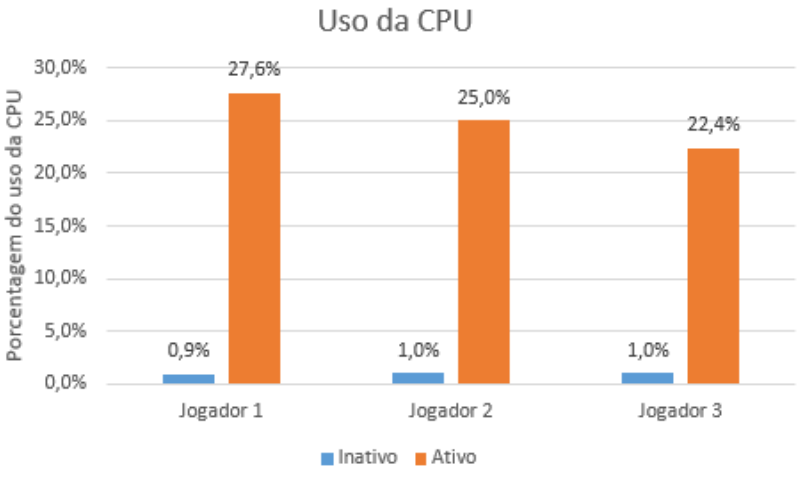


Figura 32- Respostas referentes ao uso da CPU.

6.2.2 Objetivo 2: robustez

Medidas: MQ7.1 e MQ8.1

Conforme mencionado na seção 6.1, ao longo da terceira execução do jogo a rede em que os computadores se conectava foi desligada, com intuito de avaliar a robustez da aplicação e – indiretamente – a biblioteca.

Nas respostas obtidas pelo questionário Q-01 foi constatado que a aplicação exibe a informação de que a conexão foi perdida. Essa informação é exibida por meio da tela de logs presentes na aplicação. Foi ainda constatada uma satisfação geral com relação ao comportamento durante a falha. Contudo, para um dos participantes, um comportamento mais adequado seria a tentativa de reconexão após certo período de tempo.

6.2.3 Objetivo 3: funcionalidade

Medidas: MQ9.1 e MQ10.1

Com relação aos resultados referentes ao quesito de funcionalidade, dois participantes da pesquisa não gostaram da forma como o personagem é visualizado, tendo que alternar entre as telas do Minetest e do editor do Scratch. Este problema talvez seja resolvido caso o servidor Minetest esteja rodando em um computador com projetor, de forma a mostrar para todos os jogadores.

Acerca dos pontos positivos da integração, todos citaram a facilidade de aprender o jogo. Isso se refere ao sistema Scratch-Minetest-Integração, mas é possível supor que a utilização da biblioteca MineScratch de uma forma mais complexa pode ser uma boa forma de ensinar programação. Os principais pontos negativos foram a lentidão na tradução de instruções Scratch para Minetest e também a falta de interação entre os jogadores. A falta de interação é um problema meramente do jogo educacional que utiliza a biblioteca, não do MineScratch em si.

6.2.4 Discussão

No geral, a partir da avaliação realizada é possível perceber que a biblioteca uma eficiência média, com poucos recursos da máquina sendo utilizados, contudo com tempos de resposta relativamente altos.

Com relação a robustez, a sugestão de tentativa de reconexão após uma queda de rede, dada por um dos participantes, foi observada como uma possível melhoria.

No quesito de funcionalidade, percebe-se que o maior problema apontado pelos participantes foi a forma de visualização do jogo, algo que não está ligado diretamente à biblioteca.

Observa-se que para a utilização em um ambiente de produção seria necessário otimizar a comunicação com o Minetest, pois como a utilização de jogos educacionais é uma forma de prender a atenção do aluno, altos tempos de resposta podem dispersá-los, tornando o jogo ineficaz no ensino. Essa melhora na eficiência deve ser feita melhorando o gerenciamento das múltiplas *threads* disponíveis na biblioteca, pois provavelmente esta parte do código está causando a demora presenciada nas execuções.

6.3 Ameaças à validade da avaliação

A principal ameaça à validade da avaliação é o fato da biblioteca ser avaliada indiretamente, por meio de seu uso na construção de um jogo educacional. De fato, uma forma de melhor avaliá-la seria aplicá-la na construção de aplicações por outros desenvolvedores que não o autor, porém, pela complexidade de desenvolvimento de um software, se tornaria inviável no escopo de um projeto de conclusão de curso.

Outra ameaça é o tamanho da amostra e a uniformidade de execuções. Com uma amostragem baixa e execuções sempre em um mesmo ambiente, é possível que

não diferentes cenários não tenham sido presenciados, o que poderia afetar os resultados com relação a eficiência e robustez.

Em relação ao ambiente de aplicação pode não ser exatamente o que seria encontrado em uma aplicação em sala de aula ou laboratório de informática. Mas, para simular uma rede instável, esta foi desconectada durante os testes.

A partir das informações coletadas com a avaliação foi elaborada a conclusão do projeto, apresentada no capítulo seguinte.

7. Conclusão

Este trabalho apresenta uma integração entre as ferramentas Scratch e Minetest na forma de uma biblioteca de software. Também foi desenvolvido o protótipo de um jogo educacional para o ensino de programação que utiliza esta biblioteca como forma de testá-la.

Foi então inicialmente levantada a fundamentação teórica:

- Modularização de software e bibliotecas: por se tratar de uma integração na forma de uma biblioteca.
- Desenvolvimento de jogos educacionais: já que a biblioteca será validade por meio de um jogo educacional, foi levantada a fundamentação teórica sobre este tema.
- Programação e algoritmos: o jogo educacional criado tem como intuito o ensino de programação por meio da biblioteca proposta. Por isso foi levantada a fundamentação teórica sobre este tema.
- Ferramentas da integração: foram analisados os softwares integrados

Para conhecer o estado da arte foi realizada uma revisão sistemática da literatura sobre jogos integrando Minetest e Scratch para ensino de programação. A partir desta revisão foi constatado que provavelmente não existem outras integrações entre estas duas ferramentas. Contudo foram extraídos dados de integrações com cada uma das ferramentas em separado.

Na sequência foi apresentada a modelagem e o desenvolvimento, tanto da biblioteca como do jogo educacional que a utiliza. A extensão HTTP se mostrou

limitada, apenas enviá-las, limitando a gama de instruções de código que podem ser criados.

Por se tratar de uma biblioteca, sua avaliação de forma direta seria difícil no escopo de um trabalho de conclusão de curso. Por isso a integração foi avaliada indiretamente por meio da aplicação do jogo educacional que a utiliza por uma amostra. A biblioteca se mostrou relativamente eficiente, porém nos testes foi percebido que poderia ser otimizada na comunicação com o Minetest.

Assim, é possível perceber que os objetivos deste trabalho foram atingidos, entretanto alguns aspectos poderiam ser melhorados.

7.1 Trabalhos futuros:

Levando em consideração as informações obtidas durante o desenvolvimento deste trabalho, bem como os resultados apresentados no capítulo 6, são definidos como possíveis trabalhos futuros:

- Melhorar da eficiência na comunicação com o Minetest, por meio da otimização nos algoritmos de rede e do gerenciamento de múltiplas *threads*.
- Verificar a possibilidade de alteração da utilização do Scratch para a biblioteca Blockly, diretamente integrada no Minetest.
- Avaliar a biblioteca de forma direta, aplicando-a no desenvolvimento de jogos e avaliando este desenvolvimento.
- Desenvolver um jogo educacional completo utilizando a biblioteca seguindo a abordagem ENgAGED na totalidade.

8. Referências

ALENCAR, Gersica A.; FREITAS, Ana K.; DANIELLE, J. S. **Utilizando o SCRATCH nas aulas de Lógica de Programação do Proeja:** Um relato de experiência. 2014.

AMIEL, Tel et al. **Dominando para não ser dominado**: Autonomia tecnológica com o Projeto Jovem Hacker. 2015.

BALAJI, S.; MURUGAIYAN, M. Sundararajan. Waterfall vs. V-Model vs. Agile: **A comparative study on SDLC**. International Journal of Information Technology and Business Management, v. 2, n. 1, p. 26-30, 2012.

BALASUBRAMANIAN, Nathan; WILSON, Brent G. Games and simulations. In: **Society for information technology and teacher education international conference**. 2006.

BARROS, A. J. S. e LEHFELD, N. A. S. **Fundamentos de Metodologia: Um Guia para a Iniciação Científica. 2 Ed.** São Paulo: Makron Books, 2000.

BATTISTELLA, P.; GRESSE VON WANGENHEIM, C. **ENgAGED**: **Processo de Desenvolvimento de Jogos para Ensino em Computação**. Relatório técnico. Instituto Nacional para Convergência Digital, Universidade Federal de Santa Catarina, Florianópolis, 2015.

BIERNACKI, Patrick; WALDORF, Dan. **Snowball sampling**: Problems and techniques of chain referral sampling. Sociological methods & research, v. 10, n. 2, p. 141-163, 1981.

BOTELHO, L. **Jogos educativos aplicados ao e-learning**. 2003. Disponível em: < http://www.elearningbrasil.com.br/home/artigos/artigos.asp?id=1921>

BRASSCOM, 2013. **Mercado Brasileiro de TIC.** Disponível em: <http://www.brasscom.org.br/brasscom/Portugues/detInstitucional.php?codArea=3&codCategoria=21>. Acesso em: 21 jul. 2015.

BRASSCOM, 2013. **Procuram-se profissionais de TI.** Disponível em: <http://www.brasscom.org.br/brasscom/Portugues/detNoticia.php?codArea=2&codCategoria=26&codNoticia=400>. Acessado em: 21 jul. 2015.

CODE.ORG, disponível em <http://code.org/>. Acessado em 21 jul. 2015.

DIVERIO, Tiarajú A.; MENEZES, Paulo B. **Teoria da Computação: Máquinas Universais e Computabilidade-Vol. 5**. Bookman Editora, 2009.

FERNANDEZ, Jean-Claude; MOUNIER, Laurent; PACHON, Cyril. A model-based approach for robustness testing. In: **IFIP International Conference on Testing of Communicating Systems**. Springer Berlin Heidelberg, 2005. p. 333-348.

HEINEN, Eduarth; LAUER, André George; FILHO, Pedro Lealdino. **PROTÓTIPO DE APLICAÇÃO DE PROGRAMAÇÃO VISUAL PARA CONTROLE DE MECANISMOS E SENSORES COM RASPBERRY-PI E NODE. JS**. 2014

HOROWITZ, Ellis; SAHNI, Sartaj. **Fundamentals of computer algorithms**. Computer Science Press, 1978.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. UK: NICTA Technical Report 0400011T.1, Keele University, 2004.

KUROSE, James F.; ROSS, Keith W. Redes de Computadores e a Internet. São Paulo: Person, 2006.

LOPES, Abrahão. **Desvios Condicionais*.*** Disciplina de Algoritmos, 2014. Disponível em <http://docente.ifrn.edu.br/abrahaolopes/2014.1-subsequente/1.2411.1v-algoritmos/03-desvios-condicionais> Acesso em: 7 mar. 2016

MCDONALD, J. **Exam Review Strategies**, 2004. Disponível em: <http://www.wlu.ca/documents/107/Exam_Review_Strategies_Packages.pdf>. Acesso em 20 de junho de 2014.

MEYER, Bertrand. **Object-oriented software construction**. New York: Prentice hall, 1988.

MINETEST, disponível em <http://www.minetest.net/>. Acesso em 21 jul. 2015.

MINETEST Network API, disponível em <http://dev.minetest.net/Network_Protocol>. Acessado em 21 Jul. 2015.

MOURA, V. Arnaldo; FERBER, Daniel, F. **Curso C: Estruturas de repetição**, 2008. Disponível em <http://olimpiada.ic.unicamp.br/extras/cursoC/Cap06-RepeticaoControle-texto.pdf> Acesso em: 7 mar. 2016.

NITZKE, Julio Alberto. Estratégias de Ensino. Disponível em: <http://penta.ufrgs.br/~julio/tutores/estrateg.htm>. Acesso em: 27 out. 2015.

OBSERVATÓRIO SOFTEX. **Software e Serviços de TI:** A indústria brasileira em perspectiva. 2012. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/2012-Observatorio-Softex-Industria-Brasileira-Software-Servicos-TI-em-perspectiva-Versao-Completa-Portugues.pdf>. Acessado em 21 jul. 2015.

PRESSMAN, Roger S. **ENGENHARIA DE SOFTWARE:** Uma abordagem profissional. 6. ed. Porto Alegre: Mcgrawhill, 2006.

PRUSS, Alexander. Raspberryjammod-minetest, disponível em: <https://github.com/arpruss/raspberryjammod-minetest/>. Acessado em 14 out. 2016.

PRENSKY, M. **Don't bother me Mom - I'm learning**. Minneapolis: Paragon House Publishers, 2006

ROCHA, Ana Regina Cavalcante da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software**. São Paulo: Prentice Hall, 2001.

S2A_FM, disponível em: <https://github.com/MrYsLab/s2a_fm>. Acessado em 14 out. 2016.

SAVI, R. **Avaliações de jogos voltados para a disseminação de conhecimentos**. Tese (Doutorado em Engenharia e Gestão do Conhecimento), EGC/UFSC, Florianópolis/Brasil, 2011.

SCOTT, Michael Lee. **Programming language pragmatics**. Morgan Kaufmann, 2000.

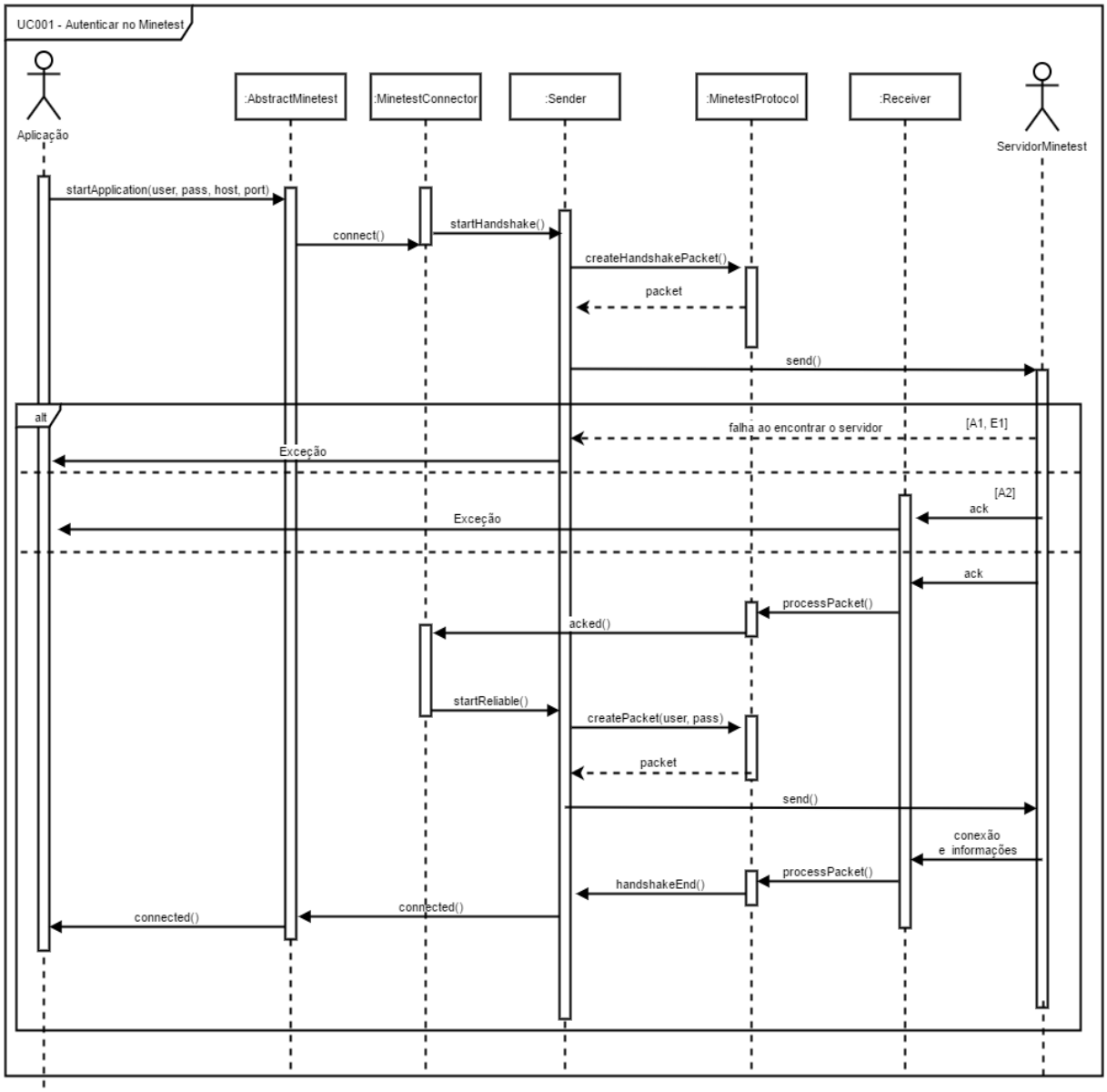
SCRATCH, disponível em <http://scratch.mit.edu/>. Acessado em 21 jul. 2015.

SELLDIN, Håkan. Design and Implementation of an Application. **Programming Interface for Volume Rendering**. 2002.

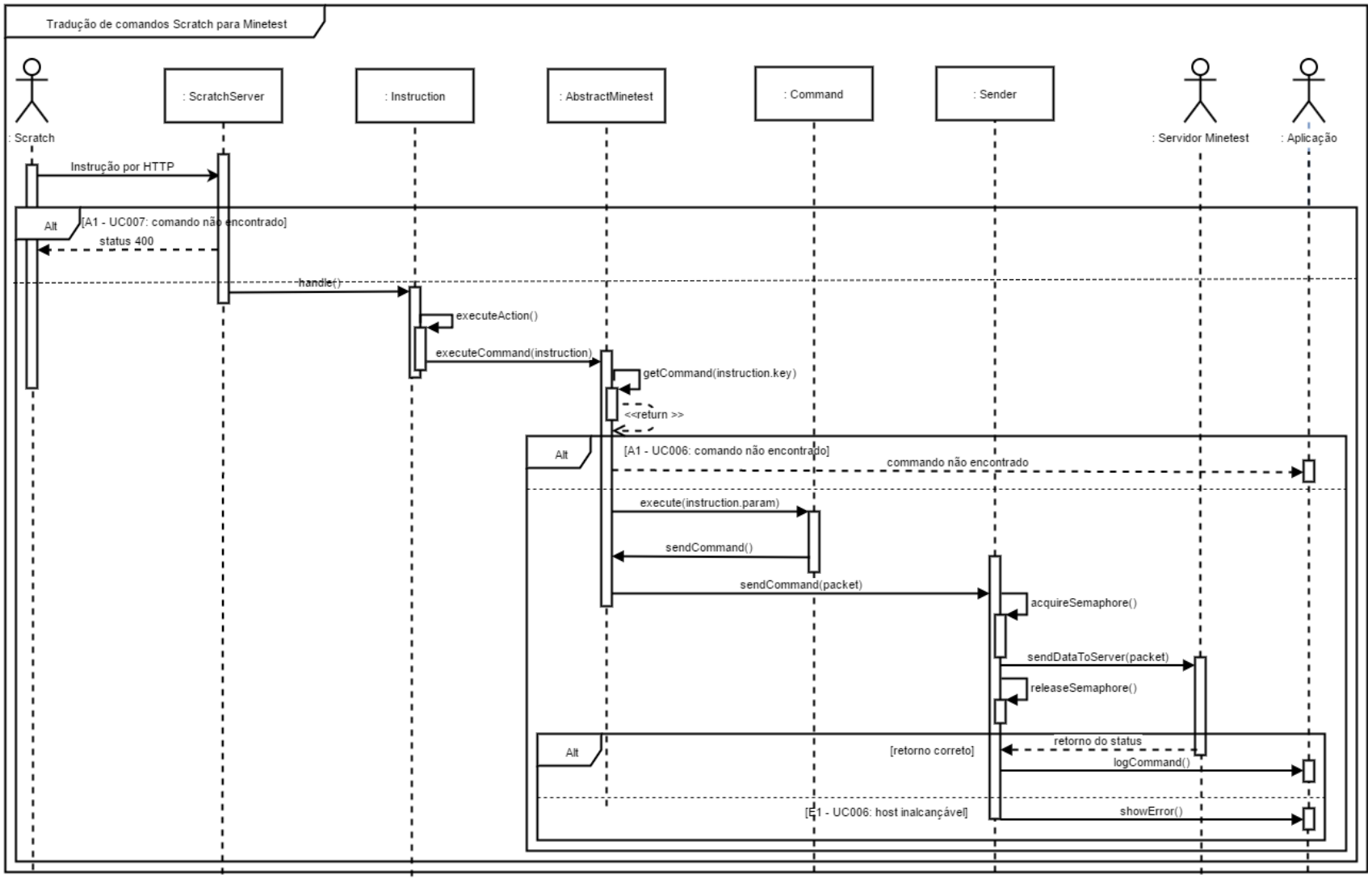
STYLOS, Jeffrey; MYERS, Brad. **Mapping the space of API design decisions**. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). IEEE, 2007. p. 50-60.

ZIMBARDO, Philip G.; LEIPPE, Michael R. **The psychology of attitude change and social influence**. Mcgraw-Hill Book Company, 1991.

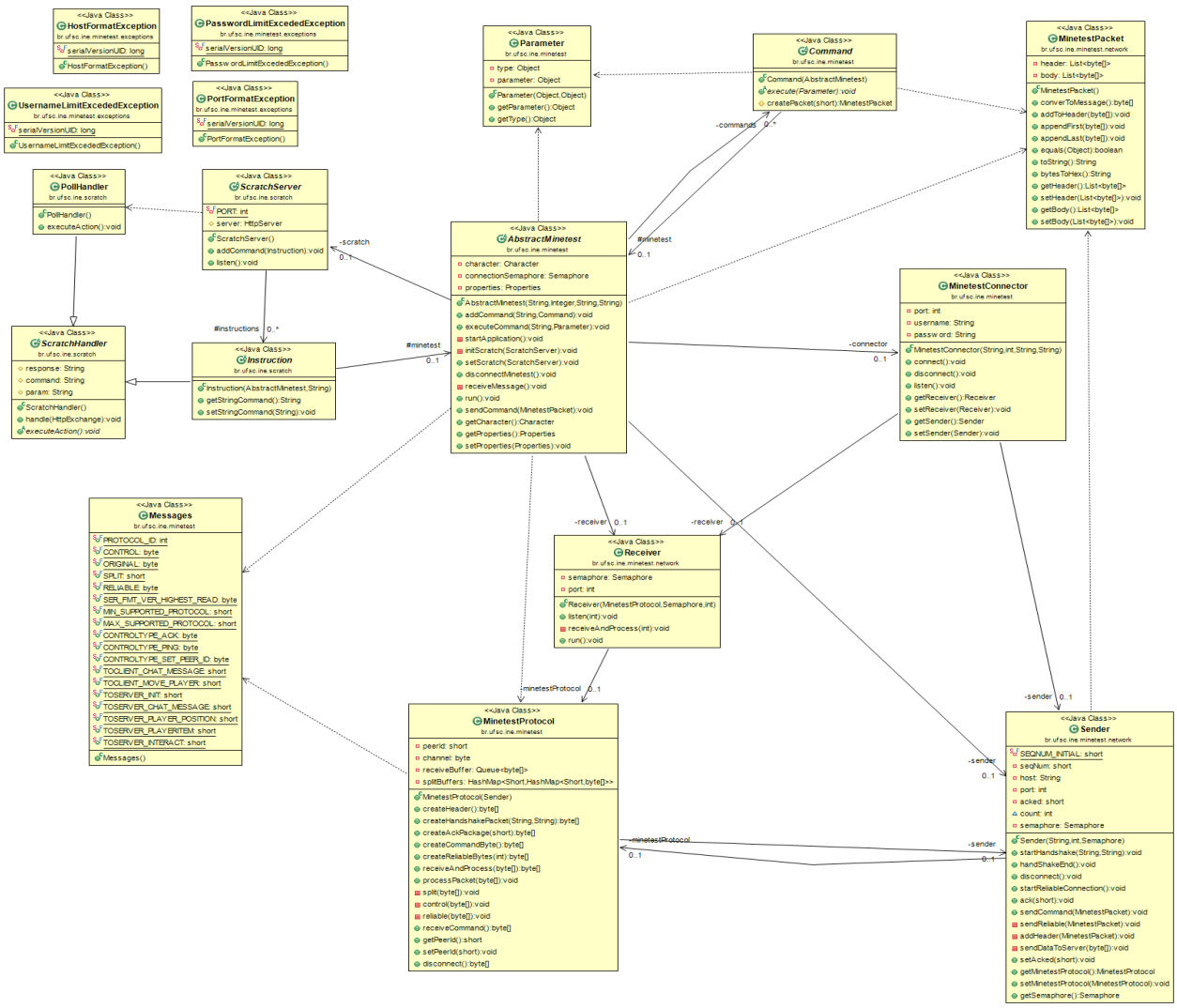
Apêndice 1 – Diagrama de sequência para o caso de uso UC001 da aplicação.



Apêndice 2 – Diagrama dos casos de uso UC006 e UC007 da biblioteca.



Apêndice 3 – Diagrama de classes do MineScratch



Apêndice 4 – Questionário Q01

Questionário: Q-01	**Nome**:
Especificações do Computador S.O.: Memória: CPU:	
01 - No jogo início do jogo, quanto tempo levou do momento do preenchimento dos dados até o personagem estar conectado no Minetest?	
Resposta:	
02 - Durante a partida, qual era o tempo entre o envio de uma instrução do Scratch até o Minetest?	
Resposta:	
03 - Quanta memória era utilizada pela aplicação: A. Antes do início da partida? B. Após o início da partida?	
Resposta: A. B.	
04 - Qual era o uso da CPU pela aplicação: A. Antes do início da partida? B. Após o início da partida?	
Resposta: A. B.	
05 - O jogo apresentou alguma falha durante sua execução?	
Resposta:	
06 - O sistema se comportou de forma positiva quando a falha ocorreu?	

Resposta:
07 - As funcionalidades oferecidas pela integração entre Scratch e Minetest são suficientes?
Resposta:
08 - Você sentiu falta de alguma funcionalidade na integração entre Scratch e Minetest?
Resposta:
09 - Cite os pontos positivos sobre a integração entre o Minetest e o Scratch.
Resposta:
10 - Cite os pontos negativos sobre a integração entre o Minetest e o Scratch.
Resposta: