Instituto de Matemática e Estatística - IME/USP
Bacharelado em Ciência da Computação - BCC

Uma Ferramenta de Simulações Interativas para Ensino de Computação para Crianças

Relatório Semestral de Iniciação Científica
Aluna: Marília Takaguti Dicezare
Orientadora: Profa. Dra. Kelly Rosa Braghetto

Resumo do Projeto Proposto

O projeto de pesquisa inicial propôs a criação de um protótipo de uma ferramenta web de simulações interativas de conceitos de lógica de programação para o ensino de computação no ensino fundamental. O ambiente de cada simulação e a sua tradução em pseudocódigo devem permitir que os estudantes as explorem livremente e também possam ter um primeiro contato com a estrutura de um código de programação.

Dessa forma, iremos desenvolver um arcabouço robusto que contenha pelo menos uma simulação de conceitos de lógica de programação, e que sirva como base para a implementação de novas simulações no futuro, juntamente com uma documentação.

O sistema será arquitetado com o estilo arquitetural Portas e Adaptadores ou Arquitetura Hexagonal (*Ports and Adapters* ou *Hexagonal Architecture*), onde a comunicação entre drivers e a aplicação é feita através de adaptadores específicos em portas da aplicação (Cockburn, 2005). Além disso, o arcabouço será desenvolvido com base no padrão arquitetural MVC (*Model-View-Controller*), o qual se apresenta em camadas (modelo, visão e controlador) e proporciona o isolamento entre as regras de negócio e a interface gráfica do usuário.

Pretendemos avaliar a usabilidade do protótipo proposto realizando validações com educadores e estudantes através de questionários de usabilidade, como o SUS (*System Usability Scale*) e o emoti-SAM (*Self-Assessment Manikin*).

Descrição das Atividades Realizadas

A seguir, fazemos uma descrição das atividades desenvolvidas até o momento e algumas conclusões iniciais obtidas.

1. Estudo dos *frameworks* e bibliotecas JavaScript

Inicialmente, foram realizadas comparações de três principais *frameworks* JavaScript de código aberto: Angular, React e Vue. No geral, as três ferramentas possuem características muitos semelhantes com respeito ao desenvolvimento, como organização em componentes, adição de responsividade, possibilidade de desenvolvimento multiplataforma, entre outras.

Em termos de popularidade e estabilidade, o React está em primeiro lugar, seguido do Angular e do Vue. A pesquisa do Stackoverflow de 2022 (Stackoverflow, 2022), com 45.297

respostas, mostra que entre os desenvolvedores profissionais o React é o *framework web* mais popular (44,31%), o Angular tem 23,06% de popularidade e o Vue (19,9%). Entretanto, os repositórios no GitHub sugerem que React e Vue são mais populares em termos de *watchers*, *stars* e *forks*. A estabilidade dos *frameworks* foi atribuída em termos das versões atuais deles (17.x - React, 13.x - Angular, 3.x - Vue).

Um ponto importante e decisivo em relação ao Angular é o seu desempenho. Ele tende a criar aplicações mais pesadas do que os demais *frameworks*. Como a ferramenta proposta envolve diversos recursos visuais e interativos, optamos por não seguir com o Angular para o desenvolvimento.

Já o React e o Vue são *frameworks* parecidos, que apresentam funcionalidades similares. Pela sua alta popularidade, o React apresenta uma enorme e bem estabelecida comunidade na internet, o que auxilia no desenvolvimento. Em contrapartida, ele pode apresentar uma curva de aprendizado maior que a do Vue, o qual possui uma documentação muito detalhada e extensa.

Dessa forma, o Vue foi escolhido para o desenvolvimento do projeto, por apresentar uma boa documentação, ser escalável e fácil de aprender. Queremos manter o foco no desenvolvimento das simulações e não no aprendizado de um *framework*.

Em relação às bibliotecas JavaScript que apresentam suporte para Vue, há muitas delas disponíveis na internet com as mais diversas facilidades. Por isso, após analisar algumas delas, como konva.js, Snap.svg e outras[1], percebemos que a maioria apresenta facilidades para lidar com eventos que serão usados nas simulações, como o *drag and drop*, por exemplo. Dessa forma, concluímos que o melhor momento para analisar mais detalhadamente os recursos oferecidos por estas bibliotecas será durante o desenvolvimento das simulações, através de testes, estudo de suas respectivas documentações e integração com o Vue.

2. Projeto das classes do sistema

Para o projeto da arquitetura das classes do sistema, inicialmente, estudamos os códigos e arquiteturas de ferramentas de ensino que pudessem ter alguma semelhança com a proposta neste projeto, como o Phet (Phet Interactive Simulations, 2023), Blockly (Blockly, 2023) e App Inventor (App Inventor, 2023). Em seguida, pensamos nas abstrações que gostaríamos de representar a nível de implementação, e as dividimos em módulos: estrutura do *framework*, das simulações e do gerador de pseudocódigo, entidades (ou elementos de simulação) e controlador.

Ainda, pensamos em um fluxo de desenvolvimento padrão para a criação das simulações, o qual iniciaria com um *template* da página com uma área de trabalho vazia e, depois, seria possível adicionar elementos com determinadas propriedades (estáticas ou dinâmicas), além do gerador de pseudocódigo.

Dessa forma, o diagrama de classes foi projetado com os detalhes de implementação descritos acima. A arquitetura hexagonal do sistema apresentada na proposta do projeto também foi redesenhada para melhor representar as separações das abstrações. As Figuras 1 e 2 mostram os esquemas da arquitetura hexagonal da ferramenta e do diagrama de classes, respectivamente.

A arquitetura hexagonal do sistema (Figura 1) continua estruturada para separar a lógica de negócios de entidades externas, apenas detalhamos mais a aplicação em si e o nível primário

[1] https://www.javascripting.com/images/2d-graphics/

de portas e adaptadores referentes a parte do usuário, conforme a divisão em módulos descrita anteriormente.

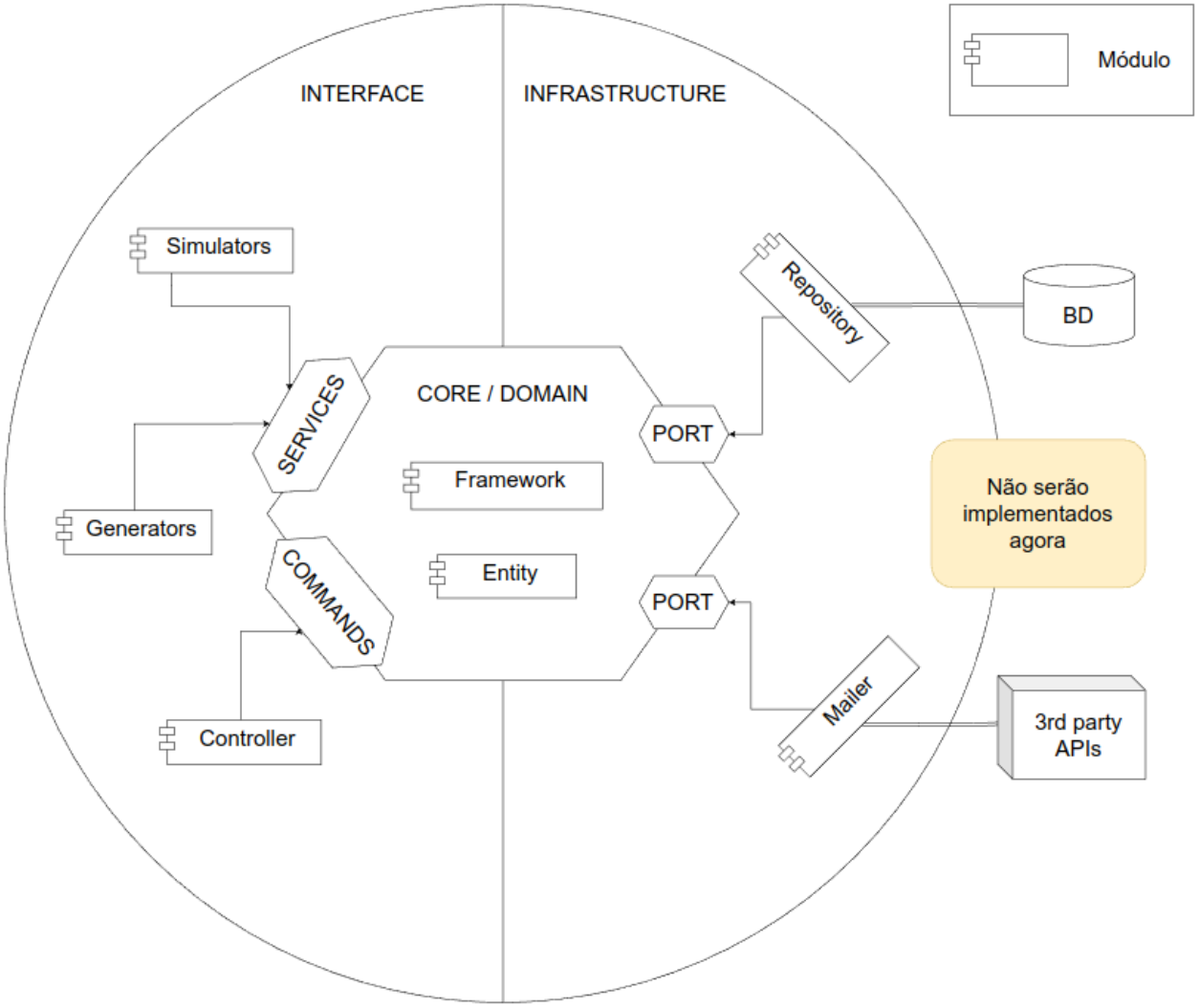

Figura 1 - Esquema da arquitetura hexagonal para a ferramenta proposta neste projeto, com duas portas referentes à parte do usuário, e outras duas referentes ao banco de dados e a notificações.

Assim, na parte central da nossa arquitetura (*Core/Domain*), teremos os módulos relacionados às regras de negócio da aplicação, os quais devem ser desacoplados dos módulos externos. Portanto, nessa divisão teremos o código do arcabouço (*Framework*) e das entidades (*Entity*), objetos relacionados às simulações que serão explicados mais adiante. Esse código deve ser a base para a criação do nosso sistema, apresentando funcionalidades comuns a todas as simulações, facilitando o seu desenvolvimento.

A interface da aplicação apresenta duas portas, uma para serviços (*Services*) na qual estão conectados os adaptadores da simulação (*Simulator*) e do gerador de código (*Generator*), e outra para comandos (*Commands*), onde se liga o controlador (*Controller*). Dessa forma, em serviços, teremos o desenvolvimento de cada simulação e do gerador do pseudocódigo correspondente. Em comandos, teremos a interação com o usuário através da emissão/recepção de eventos feita pelo controlador, seguindo o padrão arquitetural MVC (*Model-View-Controller*). Já as camadas de visão e modelo estarão embutidas no código de cada simulação, por serem específicas de cada uma.

A parte de infraestrutura da aplicação se mantém a mesma descrita inicialmente no projeto, onde o sistema irá acessar um banco de dados através de um adaptador de serviço de acesso aos dados (*Repository*), e um notificador, por meio de um adaptador de e-mail (*Mailer*) para a porta de notificações. Entretanto, a ideia dessas portas e adaptadores de nível secundário está relacionada a uma funcionalidade de conta pessoal de usuário, a qual não será implementada neste momento.

A partir da definição da arquitetura do sistema, projetamos suas classes (Figura 2) a nível de implementação. Para visualizar melhor cada módulo, eles foram separados nas Figuras 3 e 4.

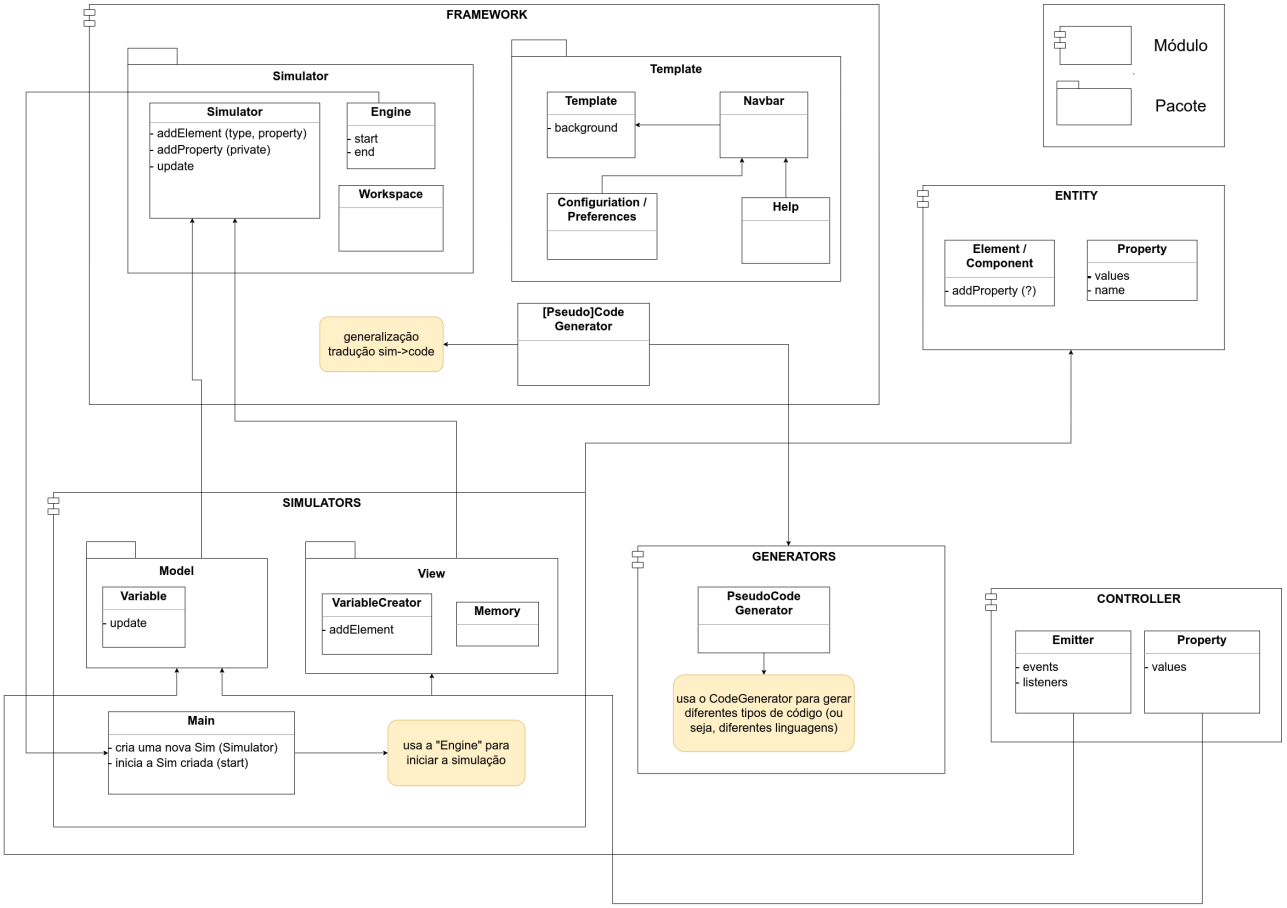

Figura 2 - Diagrama de classes UML da arquitetura do sistema.

Na Figura 3, é possível observar os módulos presentes no *Core* da aplicação. O *Framework* contém dois pacotes, um referente às simulações (*Simulator*) e outro ao *template* das páginas (*Template*), além de uma interface para a tradução da simulação em código. O pacote *Simulator* contém todas as classes comuns à implementação das simulações, desde a sua criação até a sua inicialização, e o *Template* inclui as classes para a construção das páginas web. Já o módulo *Entity* contém os objetos relacionados às simulações, ou seja, cada elemento que fará parte delas e suas propriedades estáticas (que não estão associadas a eventos). A ideia dessa parte central do sistema é prover interfaces, classes, objetos e métodos para o desenvolvimento de cada simulação. Portanto, estes serão acessados pelos módulos da interface, por exemplo, como indicam as setas na Figura 2.

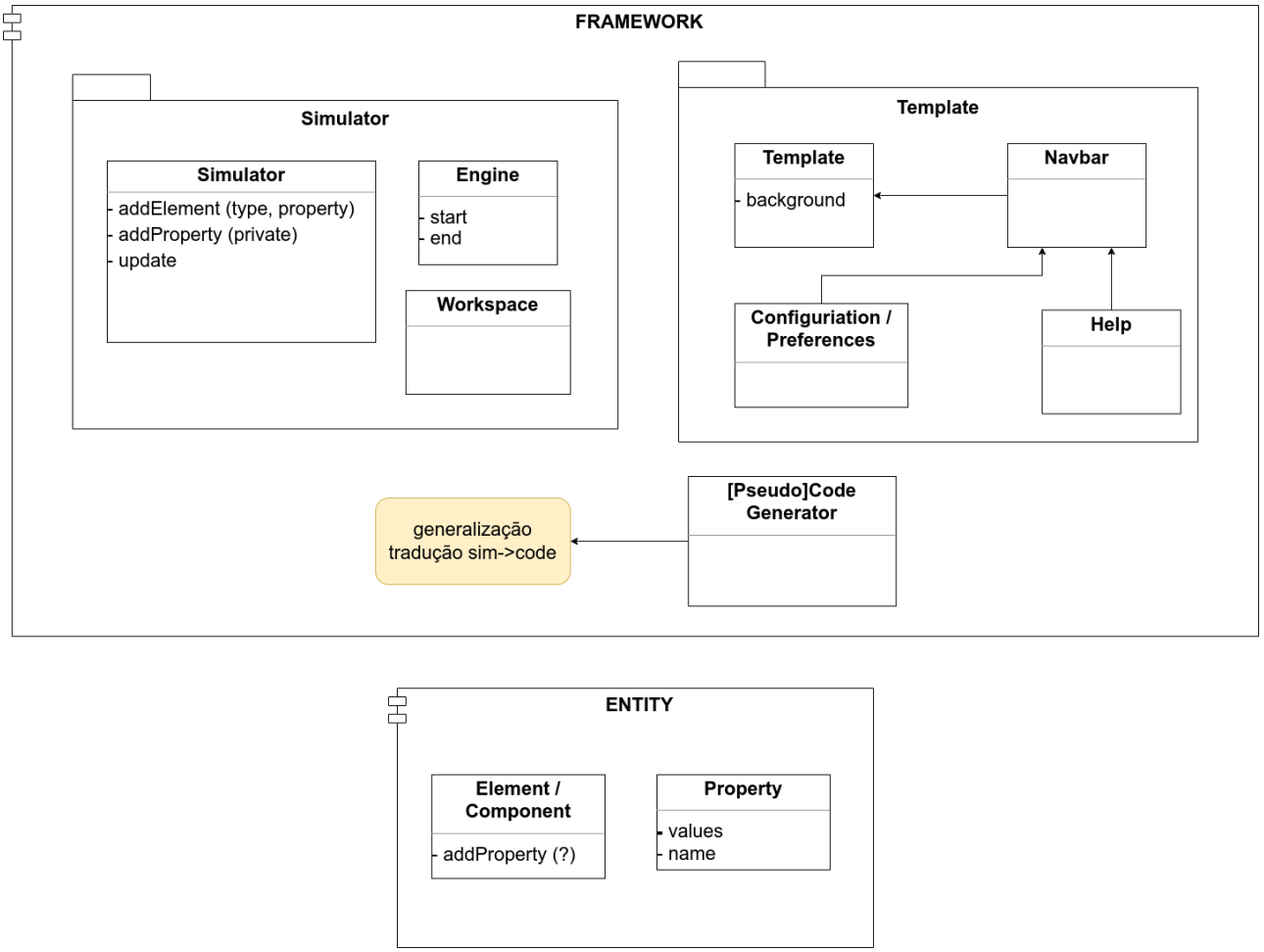

Figura 3 - Módulos contidos no *Core* da aplicação.

Os módulos referentes à interface da aplicação podem ser observados na Figura 4. Os adaptadores da porta de serviços devem conter todas as especificidades de cada simulação. Dessa forma, no módulo *Simulators*, temos a implementação das simulações em si, com os pacotes de visão e modelo do MVC e uma classe *Main*, a qual deve criar e inicializar a simulação. O *Generators* compreende a classe que de fato implementa a tradução da simulação para pseudocódigo. A forma como essa funcionalidade foi projetada, no *Core* e na *Interface*, possibilita a criação de geradores para diferentes linguagens de código, caso seja desejado.

Ainda na Figura 4, o módulo *Controller* inclui as classes que irão lidar com a emissão/recepção de eventos e as propriedades dinâmicas dos elementos das simulações, como os movimentos de arraste e click, por exemplo.

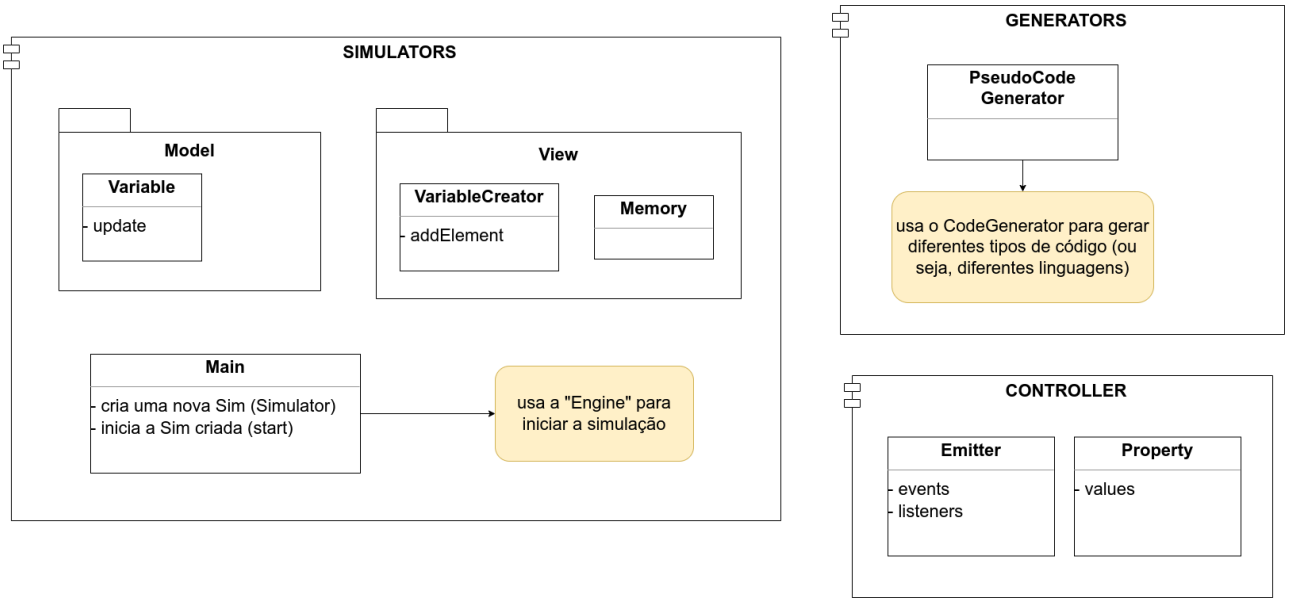

Figura 4 - Módulos contidos na interface da aplicação.

É importante lembrar que um sistema de software, assim como sua arquitetura, está em constante desenvolvimento e evolução. Por isso, os diagramas projetados são um excelente ponto de partida para o desenvolvimento da ferramenta, mas podem ser alterados conforme necessidade durante a implementação.

3. Desenvolvimento do arcabouço de simulações

O desenvolvimento do arcabouço de simulações ainda não foi iniciado, porém foram realizados alguns testes e tutoriais de exploração e manipulação de eventos em Vue para maior familiarização com o *framework* e com alguns tipos de interação que a ferramenta irá oferecer. Ademais, a projeção da arquitetura de classes do sistema possibilitou criar um ponto de partida para a implementação, a qual estava um pouco abstrata.

Após a entrega deste relatório, será criado o repositório na plataforma de controle de versionamento GitHub para o início do desenvolvimento do arcabouço junto à simulação de variáveis.

Próximos Passos

Com base no desenvolvimento atual do projeto, as atividades restantes e o novo cronograma se encontram na Tabela 1.

Tabela 1 - Atividades planejadas e cronograma.

Atividade	Mai/23	Jun/23	Jul/23	Ago/23	Set/23	Out/23
3. Desenvolver o arcabouço de simulações	x	x	x	x		
4. Desenvolver pelo menos uma simulação com ajuda de profissionais na área de educação em informática	x	x	x	x		
5. Avaliar a ferramenta junto a educadores e estudantes					x	x
6. Elaborar relatório e artigo para apresentar resultados					x	x

Referências

Blockly. (2023). Blockly. Disponível em: https://github.com/google/blockly. Último acesso em: 12/04/2023.

Cockburn, A. (2005). Hexagonal architecture. Disponível em: https://alistair.cockburn.us/hexagonal-architecture/. Último acesso em: 26/09/2022.

MIT App Inventor. (2023). Understanding an App's Architecture. Disponível em: http://www.appinventor.org/Architecture2. Último acesso em: 12/04/2023.

Phet Interactive Simulations. (2023). PhET creates free online interactive educational simulations that benefit STEM literacy worldwide. Disponível em: https://github.com/phetsims. Último acesso em: 12/04/2023.

Stackoverflow. (2022). 2022 Developer Survey. Disponível em: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies. Último acesso em: 21/11/2022.