Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Desenvolvimento de um componente web para auxílio na criação de um ambiente de ensino inspirado na linguagem Logo

Autores: Guilherme Augusto Nunes Silva e Thalisson Barreto de Melo Silva

Orientadores: Prof. Dr. Fábio Macêdo Mendes e Prof. Dr. Renato Coral Sampaio

Brasília, DF

2023

Guilherme Augusto Nunes Silva e Thalisson Barreto de Melo Silva

Desenvolvimento de um componente web para auxílio na criação de um ambiente de ensino inspirado na linguagem Logo

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Fábio Macêdo Mendes e Prof. Dr. Renato Coral Sampaio

Brasília, DF

2023

Guilherme Augusto Nunes Silva e Thalisson Barreto de Melo Silva

Desenvolvimento de um componente web para auxílio na criação de um ambiente de ensino inspirado na linguagem Logo/ Guilherme Augusto Nunes Silva e Thalisson Barreto de Melo Silva. – Brasília, DF, 2023–

Orientador: Prof. Dr. Fábio Macêdo Mendes e Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. linguagem Logo. 2. Web Component. I. Prof. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de um componente web para auxílio na criação de um ambiente de ensino inspirado na linguagem Logo

CDU 02:141:005.6

Guilherme Augusto Nunes Silva e Thalisson Barreto de Melo Silva

Desenvolvimento de um componente web para auxílio na criação de um ambiente de ensino inspirado na linguagem Logo

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Prof. Dr. Fábio Macêdo Mendes e
Prof. Dr. Renato Coral Sampaio
Orientadores

Carla Rocha Aguiar
Convidado 1

Giovanni Almeida Santos
Convidado 2

Brasília, DF
2023

Resumo

Com o estímulo do aprendizado de programação, é perceptível que mais soluções interativas tendem a ser adequadas para engajar crianças, jovens e até adultos a desbravarem esta área. A linguagem *Logo*, importante ferramenta no ensino de programação, foi fundamental para expandir o pensamento de que o estudo nessa área pode ser mais inclusiva e engajadora. Por isso, ferramentas e plataformas que facilitam o aprendizado de software têm se popularizado, com soluções que permitem acoplar facilmente uma ferramenta já criada em outra plataforma, incentivando o reaproveitamento de código através do uso de bibliotecas.

A proposta deste trabalho é criar um componente web capaz de reproduzir de forma conceitual os aspectos da linguagem *Logo*, na criação de desenhos, estimulando o engajamento no estudo da programação através do uso de linguagens e plataformas mais populares atualmente. O aspecto principal é permitir que seja acoplável em diversas plataformas web, incluindo também no Jupyter Notebook, já que o aspecto motivador das ferramentas que engajam os alunos através da interatividade no desenvolvimento tem se popularizado através de novas linguagens de programação e bibliotecas criadas.

Como resultado, foi desenvolvido um componente web apresentando aplicações tanto em *websites* quanto em plataformas como o ambiente Jupyter Notebook através do que chamamos de *Jupyter Widgets*, a fim de trazer o formato de desenvolvimento para outra linguagem sem ser a nativa de criação do componente.

Palavras-chave: web component, aprendizado, programação, Jupyter Notebook, *Logo*.

Abstract

With programming learning stimulus, are noticeable that more interactive solutions have to be properly to engage kids, young people, and even adults to explore this world. The Logo language, an important tool for programming learning, was fundamental to expanding thinking so that this area of study can be more inclusive and engaging. Therefore, tools and platforms that help software development learning have become popular, generating more solutions that can be easily coupled with another platform, encouraging code reuse through code library use.

The purpose of this work is to create a web component able to reproduce conceptually many Logo language aspects, on draw creation, encouraging the programming study engagement using languages and platforms more popular actually, allowing it should be attached to different web applications, including Jupyter Notebook because the motivating the aspect of tools that engage students through interactivity in software development has become popular with new programming languages and libraries created.

As a result, it was developed a web component integrating web applications as well as platforms such as the Jupyter Notebook environment through Jupyter Widgets technology, to bring the software development format to another language without being the native language of the created component.

Key-words: web component, learning, programming, Jupyter Notebook, *Logo*.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

Brasscom	Associação Brasileira de Empresas de Tecnologia da Informação e Comunicação
TI	Tecnologia da informação
HTML	HyperText Markup Language
DOM	Document Object Modal
API	Application Programming Interface
CSS	Cascading Style Sheet
MIT	Massachusetts Institute of Technology
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ES6	ECMAScript 6

Sumário

1 Introdução

Segundo a Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (Brasscom) (PRACIANO, 2020), o Brasil tinha em 2019 um déficiti anual de 24 mil profissionais com perfil tecnológico, e esse número dobrou nos 6 primeiros meses de 2020 onde estima-se que até 2024 esse déficit alcance a marca de 420 mil. O mercado brasileiro demanda cada vez mais profissionais da área e, em contraponto, a capacidade de formação do nosso país não está acompanhando essa tendência a ponto de conseguir suprí-la.

Dentre os profissionais da Tecnologia da Informação (TI) um grupo que se destaca são os desenvolvedores, e boa parte da demanda por profissionais da TI vem dessa área. Segundo a Brasscom (PRACIANO, 2020), as maiores ofertas de empregos são lideradas por vagas de desenvolvedores web e de aplicativos. Para suprir essa alta demanda mais pessoas precisam aprender programação, porta de entrada para áreas da TI, nesse sentido diversas inovações já foram produzidas para despertar esse interesse nas pessoas.

Para ajudar na introdução de mais pessoas no aprendizado de programação, diversas ferramentas foram criadas com o propósito de tornar este processo mais intuitivo e atrativo. O aprendizado com o auxílio de jogos é algo bem forte na comunidade educacional, com destaque para a comunidade do Minecraft. O Minecraft Education Edition permite que os alunos realizem atividades específicas dentro do ambiente do próprio jogo, tendo atividades subdivididas em categorias especificas, desde o aprendizado de matemática até a conteúdos que exercitam lógicas de programação (MOJANG, 2015).

Uma das formas de incentivar e introduzir mais pessoas ao estudo da programação é agregar a utilização de recursos multimídia. Diversas aplicações e até mesmo linguagens foram desenvolvidas para auxiliar os estudantes a aprenderem como criar esse tipo de conteúdo a partir de códigos escritos. O módulo Turtle é um exemplo, criado originalmente como parte da linguagem *Logo* e utilizado inicialmente para instruir robôs a se movimentarem, foi adaptado posteriormente para auxiliar estudantes no aprendizado de lógica e algorítmos desenhando o caminho percorrido por um objeto que comunmente é uma tartaruga por meio de um código escrito. (FOUNDATION, 2019).

Existem várias implementações de ferramentas que adaptam o Turtle para linguagens mais atuais como Javascript e Python, porém existe uma grande barreira na utilização e no aproveitamento dessas soluções pelo fato de que costumam criar uma dependência direta com a plataforma a qual foi desenvolvida. Com isso, existe a dificuldade de adaptação dessa solução para outras plataformas que possibilitam renderizar formas e figuras através de sua interface gráfica, como o Jupyter Notebook.

1.1 Objetivos

O objetivo deste trabalho é criar um novo módulo Turtle capaz de ser executado em dois ambientes, em plataformas web permitindo o acesso direto do módulo Turtle via Javascript e como Jupyter Widget para a utilização via Python por meio do Jupyter Notebook.

1.1.1 Objetivos Específicos

- estudar as tecnologias necessárias para a construção de um componente que seja adaptável para ser tanto um componente *Web* simples sendo acessável via JavaScript quanto um *widget* para o Jupyter Notebook;
- desenvolver um componente *Web* com a implementaçãoda área de desenho e possibilidade de executar comandos de movimentação e mudança de estado do direcionador (no caso do sistema Turtle, representado por uma tartaruga);
- desenvolver uma aplicação *Web* simples para validar a integração do *webcomponent*.
- desenvolver um *widget* e associar o componente *Web* como sua interface, para integrar dentro do ambiente Jupyter Notebook.

1.2 Organização do trabalho

Com estas informações, este documento será organizado de forma que primeiramente seja apresentado alguns conceitos importantes para o entendimento da proposta e, por conseguinte, das ferramentas utilizadas para realizar o produto proposto, seguindo da seguinte forma:

- O Capitulo 1 fornece informações sobre contexto e os objetivos;
- O Capítulo 2 fornece a informações mais aprofundadas sobre a linguagem Logo, componentes web e suas aplicações integradas ao Jupter Notebook;
- O Capítulo 3 apresenta a proposta de solução juntamente com a metodologia de desenvolvimento do projeto;
- O Capítulo 4 apresenta os resultados desenvolvidos e obtidos com o projeto bem como sua utilização e limitações.
- O Capítulo 5 finaliza com as considerações finais deste trabalho.

2 Fundamentação Teórica

Os tópicos abordados nesse capítulo fornecem informações necessárias para o entendimento da proposta e por consequencia da solução que apresentamos nesse trabalho. Primeiramente é abordado a linguagem logo e sua importancia dentro do contexto educacional, em seguida os componentes web e sua utilização e por fim o Jupyter Notebook e sua relevancia dentro do contexto educacional.

2.1 Linguagem Logo

Como uma linguagem simples, de fácil aprendizado e de desenvolvimento mais interativo, o *Logo* sempre buscou promover a ideia de ser uma linguagem que estimula a criatividade e introduz, de forma prática, o aprendizado de aspectos comuns de programação estimulando de forma artística e matemática na realização de formas mais sofisticadas (LOGO, 2015).

Este capítulo tem como proposta apresentar um pouco mais sobre as raízes da linguagem *Logo* e suas motivações no contexto educacional da programação.

2.1.1 História

Em 1967, Seymour Papert, matemático e educador estadunidense nascido na África do Sul, trabalhou juntamente com a equipe de Bolt Beranek and Newman, liderados por Wally Feurzeig, pesquisador da área de inteligência artificial, onde desenvolveram a primeira versão da linguagem *Logo*. Com isso, a linguagem de programação *Logo*, derivada do LISP, foi desenvolvida como uma ferramenta de aprendizado (LOGO, 2015).

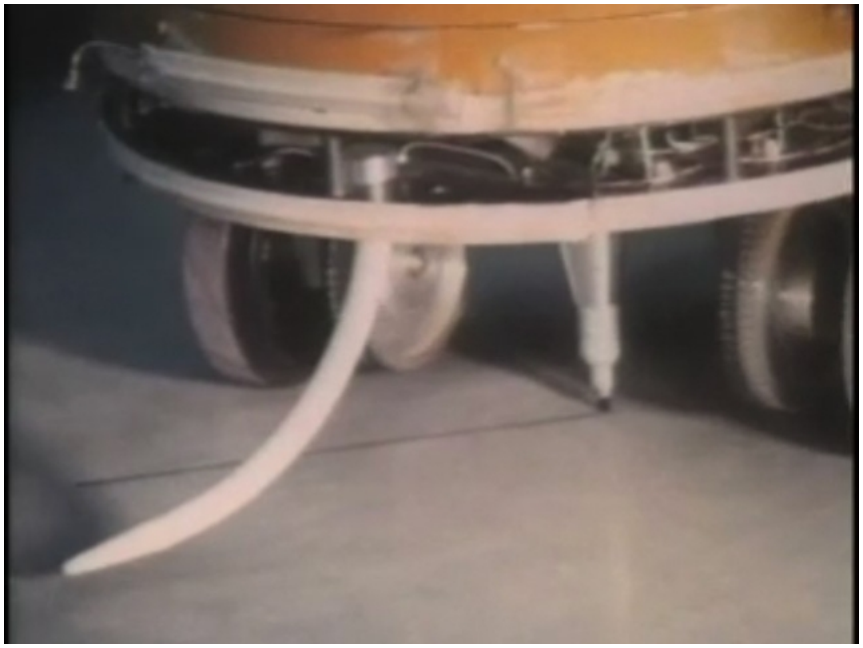
A *Turtle* era originalmente uma criatura robótica que ficava no chão e podia ser direcionada e movimentada através de comandos pelo computador, criando desenhos variados a partir do código implementado, como mostra na Figura 1, mas atualmente é bem representada através de diversas formas digitais com elementos que até mesmo não são relacionados com a tartaruga em si, mas que implementam as mesmas características lógicas que ela aplicava originalmente.

O mecanismo de desenho realizado pelo robô *Turtle* era gerado a partir de componentes que sustentavam um pincel que ficava sempre em contato com a superfície do chão, realizando os traços definidos através de código de forma clara e objetiva, seguindo os comandos de movimentação e orientação para realizar diversos desenhos específicos. Na Figura 2, é possível visualizar que o pincel, anexado na base inferior do robô com a

Figura 1 – Robô utilizado para a construção de desenhos realizados na linguagem *Logo*

ponta tocando ao chão, era fixo, sendo orientado e movimentado pelas rodas associadas à ele.

Figura 2 – Pincel acoplado ao robô Turtle para realizar os desenhos através de comandos



Após toda evolução dos robôs que realizavam as mesmas funcionalidades do *Turtle*, suas características foram migradas para a parte gráfica de computadores, sendo hoje usada para desenhar formas, desenhos e figuras digitais. Com isso, nos diversos contextos de aprendizado tornaram-na inspiração e pioneira para outras ferramentas, modelos e

vários outros sistemas.

Durante a década de 1970, enquanto o projeto *Logo* estava encubado no MIT e em alguns outros ambientes de pesquisa, pequenas atividades foram realizadas em algumas instituições de ensino de Massachusetts. Dan Watt, Cynthia Solomon e outros pesquisadores do MIT documentaram seu trabalho com um pequeno número de estudantes do ensino fundamental que utilizaram o *Logo*. Seus relatórios estão entre as dezenas de *Logo Memos* publicados pelo MIT durante esse período (LOGO, 2015).

A partir destes eventos iniciais que introduziram a aplicação da linguagem de programação *Logo* no meio educacional, foram promovidos diversos *workshops* com o objetivo de introduzir os participantes ao aprendizado inicial de programação, como por exemplo workshop com foco no exercício com elementos físicos, como abstrações da *Turtle*, através da construção de robôs com *Arduíno* (LOGO, 2015).

Com o adventos de novas tecnologias, a linguagem *Logo* e seus propósitos serviram de inspiração para a criação de ferramentas e ambientes que aprimoram o entendimento de conceitos específicos de programação em um contexto mais dinâmico e visual, originando diversas ferramentas desenvolvidas com esta filosofia.

2.1.2 Motivações

A proposta pedagógica da linguagem *Logo* possui raízes no construtivismo, formulado por Piaget, onde os alunos criam seu conhecimento através de experiências, interagindo com coisas e pessoas ao seu redor (LOGO, 2015). Tal filosofia embasa a proposta de tornar o aprendizado de programação mais didático e inclusivo.

O aprendizado da programação é dificultado pela falta de ferramentas que oferecem um feedback mais imediato, com uma interatividade maior com o desenvolvedor (VICTOR, 2012). Com esse problema em vista, o desenvolvimento da criatividade e imaginação são inibidas, não oferecendo uma liberdade maior no entendimento do assunto, pois o programador acaba se preocupando muito com as questões mais técnicas como a sintaxe do código, se sua lógica e semântica estão corretas, ao invés de ver o resultado gerado imediatamente.

Visto tal fato, a linguagem *Logo* aplicada em linguagens de programação mais populares pode funcionar como uma aplicação à esta prática, procurando, de forma mais clara e intuitiva, introduzir o ensino de programação através de uma interface visual, com *feedbacks* mais imediatos de cada ação realizada. A partir dessas características, trazer linguagens de programação populares em um contexto mais facilitado, visual e dinâmico consegue não só ajudar aspirantes à área de programação a começarem a aprender como também pode incentivar mais pessoas a se interessarem pela ideia, sendo um atrativo a maneira de começar a programação através de desenhos e figuras.

A partir de todos esses fatores positivos da metáfora *Logo*, muitas adaptações surgiram com a popularização de novas tecnologias. A biblioteca *Turtle*, por exemplo, desenvolvida em Python, traz aspectos da programação Python com as características visuais e de desenvolvimento do *Logo*, buscando estimular o aprendizado da linguagem através do desenvolvimento criativo pelas tentativas de desenvolvimento de uma figura mais sofisticada (FOUNDATION, 2019).

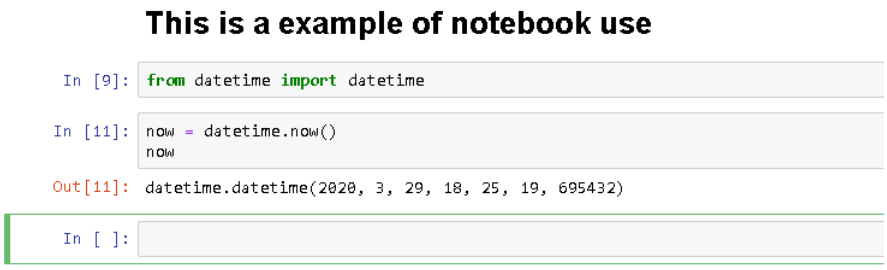
2.2 Jupyter Notebook

O Jupyter Notebook é integrado por diversos sistemas relacionados à diferentes linguagens de programação, com aplicações bem consolidadas, com foco na linguagem *Python*. A seguir iremos ver alguns conceitos específicos associados ao *notebooks* e sua influência no ambiente educativo.

Para entender um pouco sobre os notebooks e sua influência no contexto educacional, é importante entender alguns aspectos que o tornam uma tecnologia de grande impacto para o ensino de programação de forma mais acessível, prática e interativa, muito relacionando o por que usar, sua abrangência de conteúdos de estudo e sua influência para o aprimoramento do aprendizado de temas diversos.

O Jupyter *Notebook* é um ambiente onde se realiza exercícios e atividades específicas, permitindo a criação de materiais de estudo com anotações, exemplos de código e com um bom gerenciamento das dependências necessárias para a execução de um código específico, gerando uma maior interatividade com o usuário (KLUYVER et al., 2018a). Como visto na Figura 3, estes Jupyter Notebooks possuem uma estrutura com células de entrada de código ou texto, seguido de uma célula de saída do conteúdo apresentado na célula de entrada, caso gere alguma saída.

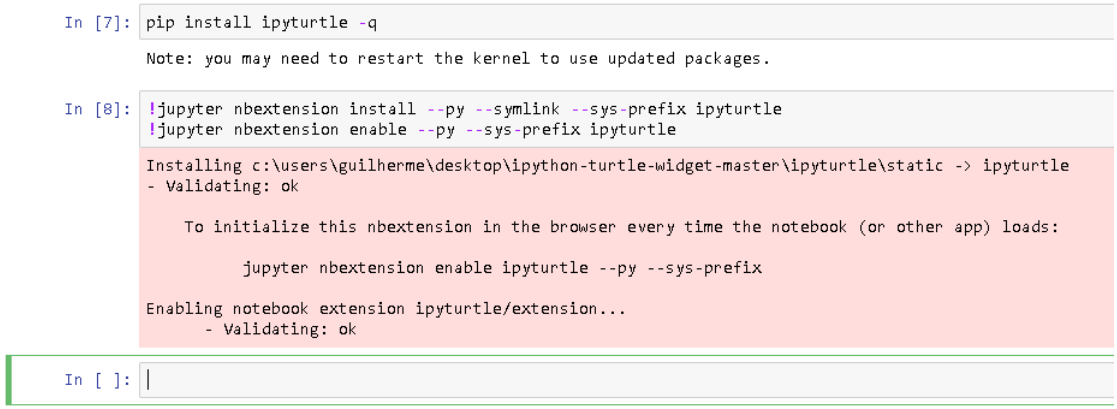
Figura 3 – Uso básico do ambiente do Jupyter Notebook



Além disso, é possível realizar configurações no ambiente do notebook através de comandos inseridos nas próprias células, além da inclusão de bibliotecas Python que podem auxiliar no desenvolvimento de acordo com a proposta de cada Jupyter Notebook. Como visto na Figura 4, o ambiente do Jupyter Notebook diferencia a execução de código Python para execução de comandos na máquina através da notação de exclamação, antes

de qualquer comando, centralizando todo o contexto do código (configuração de ambiente, código e documentação) em um documento específico.

Figura 4 – Instalação de pacotes e configuração do Jupyter Notebook



Com todos os fatores citados anteriormente, o desenvolvimento de produtos de software se torna mais prático e intuitivo, podendo ser utilizado facilmente tanto para desenvolvimento de aplicações simples como estudos e pequenos tutoriais ou até mesmo na aplicação mais complexas, como lógicas para análise na área de ciência de dados.

Semelhante à um caderno de anotações virtual, um *notebook* pode ser compartilhado em diversas plataformas, contribuindo para a difusão do conhecimento e da prática de programação em diversas áreas, tornando o aprendizado mais inclusivo. Sendo assim, a organização propõem documentos que auxiliam no ensino e no aprendizado para beneficiar todos os envolvidos, tanto para aprender sobre programação quanto para dominar outras áreas de conhecimento ou até mesmo praticar habilidades de *storytelling* durante a prática (BARBA et al., 2019b). Com todos esses fatores à seu favor, o uso de notebooks está com alto crescimento nesses últimos anos, principalmente na área da ciência de dados, facilitando também a disponibilização dos conteúdos para públicos de várias áreas.

2.2.1 Jupyter Notebook na educação

Segundo (BARBA et al., 2019a), o manual de ensino e o aprendizado disponibilizado pela organização Jupyter tem como proposta auxiliar o educador a ensinar diversos tipos de conteúdos, provendo diversas instruções que ajudam em preparar os materiais de forma prática, acessível e reutilizável, através de uso de ferramentas interativas que consequentemente proporcionam uma base para o próprio aluno conseguir estudar e criar seu próprio material de estudo com as ferramentas disponibilizadas.

Diversos estudos disponibilizados por (JUPYTER, 2019) reforçam que a utilização de Notebooks para o ensino de diversas disciplinas são efetivos se utilizados de forma correta. Com base nos casos de estudo disponibilizados, compreendemos que o ensino

através do uso de *notebooks* pode ser prático e eficaz, tanto para estudantes universitários quanto para estudantes do ensino fundamental e médio, ao passo de cada vez mais a tecnologia estar integrada com nossas atividades diárias.

Um dos estudos de caso de uso é relacionado a introdução de programação em *Python* para jovens com acesso limitado à computadores (KLUYVER et al., 2018b) através de um *workshop* de duas horas de duração, aplicando temas como arte e música através do uso de ferramentas. Com base na utilização de aplicações interativas como Notebooks, os resultados mais visuais relacionados aos conceitos trabalhados durante o *workshop* torna a transmissão do conhecimento mais prática e didática, trazendo um pouco mais de interação com *feedback* simultâneo no ensino tradicional da ciência da computação.

2.3 Componentes Web

Componentes Web são um conjunto de recursos que fornecem um modelo de componente padrão, permitindo a criação de elementos customizados reutilizáveis (CONRAD et al., 2022e). Um elemento desse tipo pode ser utilizado por vários produtos ou aplicações diferentes o que garante a esse recurso essa característica de interoperabilidade.

São uma maneira nativa de encapsular lógicas específicas de aplicações web como tags, de forma personalizada possuindo comportamentos e até estilos próprios. Apesar de não ser a única maneira de implementação, eles têm o diferencial de se preocupar em manter o código facilmente reutilizável, podendo ser acoplado em diversos ambientes sem a preocupação de conflito de código, diferente de soluções criadas de forma mais específica, como componentes reutilizáveis apenas dentro do contexto de um framework.

Temos vários exemplos de componentes *web* usados hoje, como por exemplo um componente para renderizar o mapa em um site, como o *good-map* (LEE, 2017), componente simplificado que traz uma interação semelhante a que temos nativamente na página do Google, podendo navegar e realizar ações simples dentro do mapa renderizado, garantindo uma boa usabilidade e encapsulando toda a lógica apenas nesse componente.

Para isso, os *WebComponents* utilizam três tecnologias principais: Elementos customizados, Shadow DOM e Templates HTML.

2.3.1 Elementos Customizados

Os Elementos customizados são definidos por um conjunto de APIs JavaScript, com comportamentos específicos e podendo ser utilizados de diferentes formas na aplicação. Como tags customizadas, sua utilização é bem facilitada, trazendo um pouco da caracterista de organização de código ao elemento criado (CONRAD et al., 2022a).

Existem várias páginas *web* que fazem uso desses elementos customizados para

incluirem no próprio contexto do que a página entrega, facilitando tanto para construirem elementos de forma mais organizada e descritiva como para também representar até mesmo que aquele componente criado tenha uma assinatura à qual plataforma ele pertence (caso seja reutilizado em outras plataformas *web*).

Páginas como o YouTube, plataforma de compartilhamento de vídeos, incluem diversas utilizações de tags HTML customizadas, tanto como os componentes que são usados para permitirem uma pré-visualização dos vídeos como até mesmo o próprio reprodutor disponibilizado pela plataforma, sempre englobando toda a lógica em uma tag simples que, nesse caso, possui uma assinatura de que é um componente que pertence à tal plataforma.

Algoritmo 1 Exemplo de elementos customizados na plataforma YouTube

```
1 <!- Elemento customizado para o reprodutor de video ->
2 <ytd-player id="ytd-player" context="
     WEB_PLAYER_CONTEXT_CONFIG_ID_KEVLAR_WATCH" class="style-scope ytd-
     watch-flexy" style="touch-action: pan-down;">
3
4 <!- Imagem de miniatura dos videos ->
5 <ytd-thumbnail width="9999" class="style-scope ytd-rich-grid-media" size
     ="large">
6
```

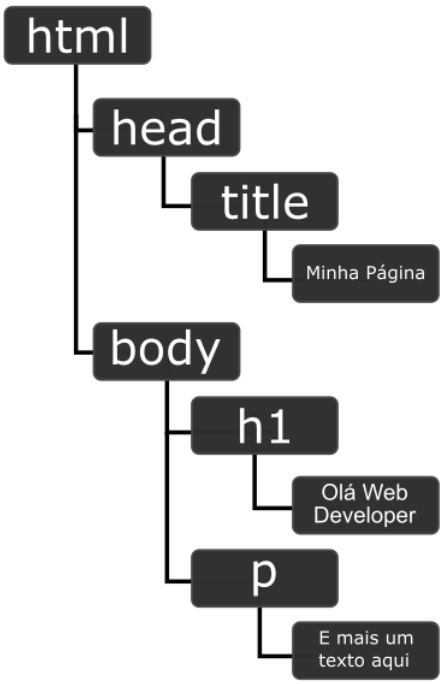
2.3.2 Shadow DOM

Para entendermos os conceitos que envolve o Shadow DOM, primeiro precisamos saber um conceito principal que nos ajuda a reforçar a própria ideia dessa tecnologia: a própria DOM.

DOM significa Document Object Model, em português Modelo de Objeto de Documento, sendo uma interface de programação para documentos da web. Basicamente, o DOM pode ser representado por várias árvores de nós que compõem a estrutura de uma página, permitindo a interação e manipulação das informações de uma página por ela. Se levarmos em consideração uma estrutura simples de uma página, como exemplificado na Figura 5, acessar a DOM permite com que possamos fazer várias modificações através da linguagem JavaScript, permitindo inserir lógicas mais complexas dentro da página (CONRAD et al., 2022b).

Porém, o Shadow DOM, é uma estrutura semelhante à uma árvore de nós HTML que é anexada a um elemento específico e que não interage diretamente com os nós do documento. Ou seja, é uma camada que pertence a um elemento que se encontra dentro do documento, encapsulando os nós para que qualquer definição de regras no CSS utilizando

Figura 5 – Estrutura de nós que representa a DOM



ids e classes seja aplicada ao contexto do elemento, e não do documento em si (CONRAD et al., 2022c).

2.3.3 Template HTML

Os Templates HTML são elementos que permitem encapsular um conteúdo que não é exibido ao inicializar uma página web. Com isso, permitem a reutilização deste código em diversas partes do sistema, garantindo, para elementos mais customizados, que sua estrutura fique mais consistente e bem reaproveitável (CONRAD et al., 2022d).

2.3.4 Jupyter Widgets

Os *Jupyter Widgets* são componentes Python que possuem eventos com representação no navegador. Um dos mais comuns que já estão atrelados a biblioteca *ipywidgets* são caixas de seleção simples, *sliders*, entre outros (VOLKL, 2019a). Arquiteturamente, eles possuem duas camadas principais: camada de interface do componente em JavaScript e a camada de abstração Python para a integração com o Jupyter Notebook. A comunicação entre essas duas camadas está especificado no Seção 3.2.1.1.

Na camada de interface do componente em JavaScript, entramos com os conceitos reforçados de Componentes Web, criando elementos customizados para agirem de força acoplada ao contexto do widget criado, além de agindo diretamente com a Shadom DOM daquele elemento específico a fim de aplicar lógicas (dinâmicas ou mais estáticas) para garantir o comportamento devido daquele widget de forma devida, permitindo que sua

inclusão no documento da plataforma web seja feita facilmente através da inclusão da tag HTML customizada relacionada à implementação da interface JavaScript.

Já, na camada de abstração Python que é integrada ao Jupyter Notebook, é feita a criação da lógica que irá servir como interface para acessar as funcionalidades implementadas pelo componente web em JavaScript, codificando os comandos através de classes Python em integração com o código JavaScript, onde as classes Python a serem usadas na interface do Jupyter Notebook devem ser registradas como widgets e trabalharão diretamente com a lógica da interface do componente dentro do ambiente Jupyter, em JavaScript, seguindo o modelo de arquitetura Model-View, através da biblioteca *backbone.js* (SYLVAIN et al., 2022).

Existem diversos *widgets* criados para diferentes propósitos, como, por exemplo, para aplicações mais voltadas em análises geoespaciais, como também com foco em visualização de dados de forma mais dinâmica e interativa. Logo, com esse material disponibilizado para agregar o valor didático em *notebooks*, o formato de apresentação auxilia à tornar o ensino mais interativo, confortável e inclusivo, abordando um *feedback* mais imediato nas atividades realizadas através de componentes visuais mais intuitivos.

3 Proposta

O início deste projeto se deu por meio de uma ideia do professor Fábio Macedo Mendes, um dos orientadores deste trabalho, que pensou na possibilidade de uma ferramenta para auxiliar o ensino de programação para ser usada em ambientes web mas também no Jupyter Notebook, ferramenta que ajuda bastante o ensino e aprendizado ao unir o código com texto em explicações que podem ser muito detalhadas. Dessa forma, foi sugerido uma implementação baseada em uma ferramenta já conhecida e utilizada no ensino de programação, ou seja, uma ferramenta já testada para esse propósito, o *módulo turtle* da linguagem *Logo*.

Com isso, este capítulo terá a finalidade de apresentar a metodologia que será utilizada no desenvolvimento da proposta, além de aprofundar um pouco mais na explicação das ferramentas que serão utilizadas para sua construção bem como a proposta definida e seus requisitos.

3.1 Metodologia

Através dos objetivos apresentados tem-se que o tipo ideal de pesquisa a ser realizada deverá ser a pesquisa exploratória qualitativa, por se tratar de um tema onde é realizado estudos através de levantamento bibliográfico e desenvolvimento experimental para garantir a viabilidade da proposta apresentada (GIL, 2002). Com isso, algumas metodologias e ferramentas serão utilizadas a fim de concluir o objetivo.

3.1.1 Kanban

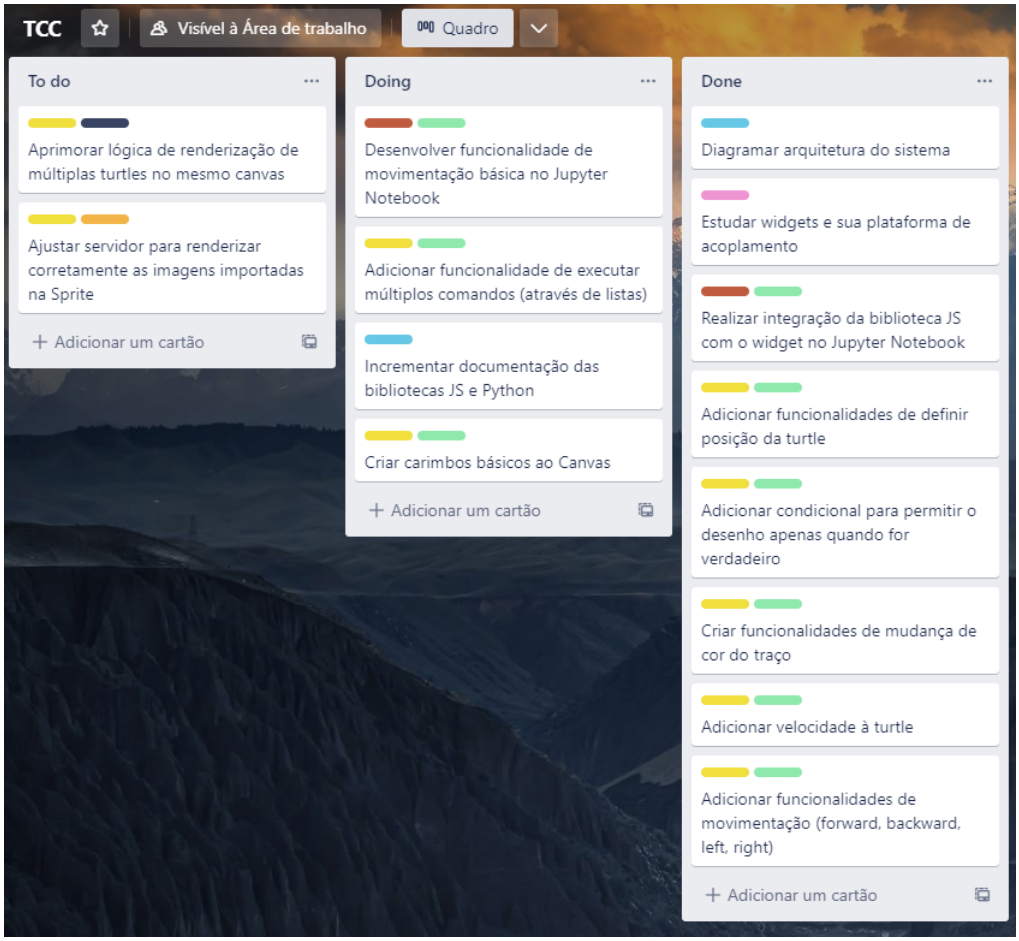
Para uma melhor organização das atividades que serão realizadas no desenvolvimento do componente *web* para serem aplicadas em ambientes expernos, como o Jupyter Notebook e uma aplicação *web* simples, o Kanban será utilizado durante o desenvolvimento da proposta por proporcionar um bom acompanhamento das tarefas definidas para a conclusão do componente, descrevendo o tempo designado para cada tarefa ser finalizada e as definidas para serem realizadas posteriormente.

O Kanban é um sistema ágil e visual que envolve a utilização de cartões que descrevem as atividades mapeadas do projeto, associando através de colunas específicas as atividades que estão sendo realizadas (RADIGAN, 2019). O quadro Kanban proposto terá a divisão de três colunas principais: **To Do**, **Doing** e **Done**, possibilitando o controle das atividades planejadas, das trabalhadas no momento e das já finalizadas, mantém sempre uma boa visibilidade para manter o trabalho em equipe mais produtivo. Ele pode ser re-

presentado de forma física ou virtual, de forma com que proporcione uma boa organização e um controle das tarefas de acordo com a prioridade definida por cada.

Na proposta de metodologia apresentada, a coluna **To Do** terá como ideia organizar todas as tarefas que deverão ser realizadas para o desenvolvimento do projeto, abordando cada cartão como uma atividade específica, podendo ser subdividida em várias tarefas. Já, a coluna **Doing**, representará os cartões que estão sendo desenvolvidos, garantindo um controle das atividades que foram priorizadas. Por fim, a coluna **Done** representa todas as tarefas já concluídas, espelhando o progresso do desenvolvimento.

Figura 6 – Exemplo de utilização do quadro Kanban no Trello



Para o desenvolvimento da proposta, optou-se pela utilização do board da ferramenta Trello, podendo ser abordado tanto para o desenvolvimento da parte escrita quanto na parte de código da proposta documentada. Com essa ferramenta a disposição, se torna mais fácil e prático a aplicação desta metodologia durante a etapa de desenvolvimento do *software*, listando todas as *issues* em forma de atividades a serem realizadas de forma organizada, como pode ser visto na Figura 6.

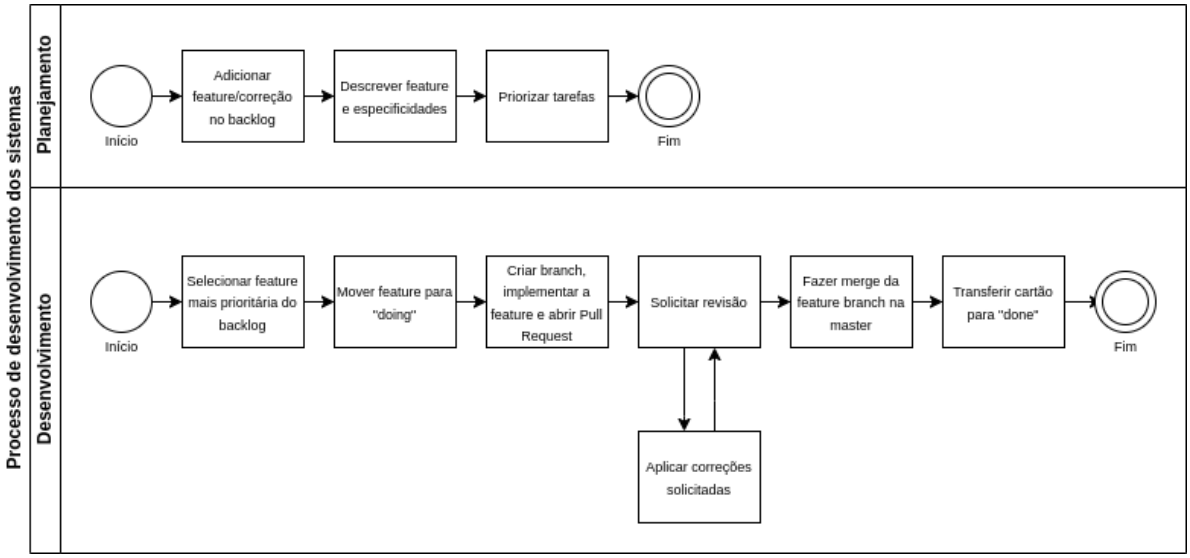
3.1.2 Processo

Seguindo a metodologia nosso processo de desenvolvimento foi realizado os passos:

- Levantamento dos requisitos e criação de cards para cada requisito.
- Priorizar e ordenar cards por ordem de prioridade no quadro Kanban.
- Selecionar uma feature do Kanban para desenvolvimento.
- Mover card para coluna de "fazendo"
- Criar uma branch para o card e desenvolver a feature
- Fazer merge da branch criada na master
- Mover cartão para coluna de feito

A Figura 7 abaixo ilustra o processo descrito.

Figura 7 – Processo de desenvolvimento



3.2 Ferramentas

3.2.1 Jupyter Notebook

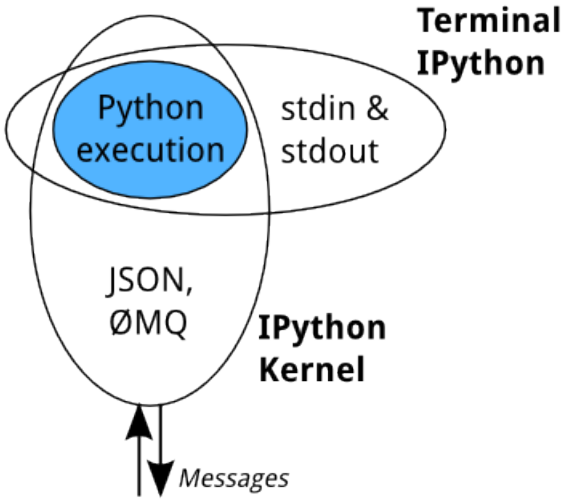
O Jupyter Notebook foi um dos ambientes utilizados para a integração do componente *web* proposto por este trabalho. Levando em conta os aspectos tratados na Seção 2.2, onde explica o funcionamento da arquitetura de um *Notebook*, é importante entender também como funciona as integrações de bibliotecas e afins, pois é fundamental para o entendimento geral do processo de realização da proposta abordada por este trabalho.

3.2.1.1 Arquitetura do sistema

Para explicar sobre a arquitetura dos *notebooks*, é importante entender sua relação com o *Kernel* do *IPython*, pois é a base para a criação e funcionamento dos *Notebooks*.

Como representado na Figura 8, o kernel do IPython é um processo que trabalha separadamente e que recebe as informações de código dos notebooks em formato JSON, através de soquetes *ZeroMQ*: biblioteca que permite o transporte em baixo nível de informações de forma adaptada à estrutura de filas de forma assíncrona, tornando o soquete transparente e consistente, permitindo conexões de vários terminais, mantendo uma consistência do *Kernel* do *IPython* com a camada do *Front-end* do *notebook*. Com isso, toda mudança no aspecto do conteúdo de determinados objetos do código desenvolvida no *Front-End* é adaptada para formato JSON e interpretada pelo Kernel do IPython para a sincronização das informações com os objetos instanciados no sistema.

Figura 8 – Diagrama arquitetural da camada de comunicação entre *Front-end* e o *Kernel* do *IPython*



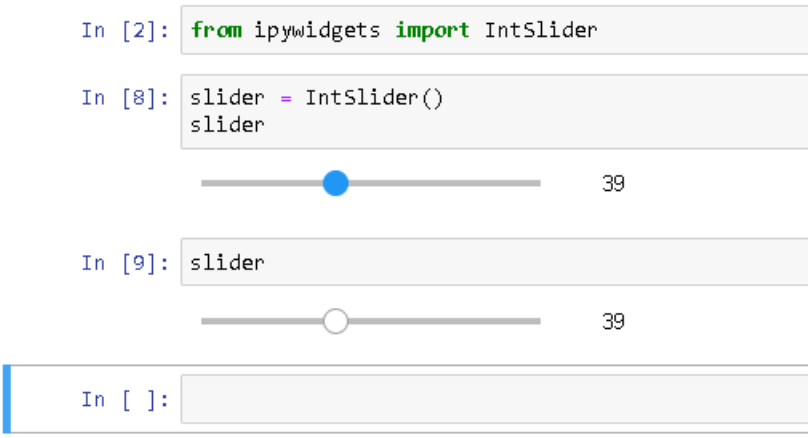
Logo, com essa característica, é possível com que as células de saída possam ser sincronizadas, caso a alteração de alguma informação específica seja feita, pois o *widget* age de forma que possam existir várias representações de um componente e que seus valores sejam alterados de forma síncrona.

3.3 Desenvolvimento de um ambiente inspirado na linguagem *Logo*

3.3.1 Componente web para associação com diferentes ambientes

Existem poucas bibliotecas e adaptações da linguagem Turtle que trabalham com a ideia de associar suas funcionalidades de forma abrangente, podendo ser adaptada tanto

Figura 9 – Exemplo de widget com sincronia da definição dos valores em duas chamadas da mesma instância



para simples websites quanto também para plataformas mais complexas como o Jupyter Notebook. O detalhe que acaba prejudicando na facilidade dessa reutilização é o fato de que, nas bibliotecas encontradas pela pesquisa feita para este trabalho, a lógica da linguagem Turtle se encontra separada em pastas mais por questão de lógica de negócio, não sendo generalizadas como componentes web para serem facilmente atreladas à outro tipo de sistema.

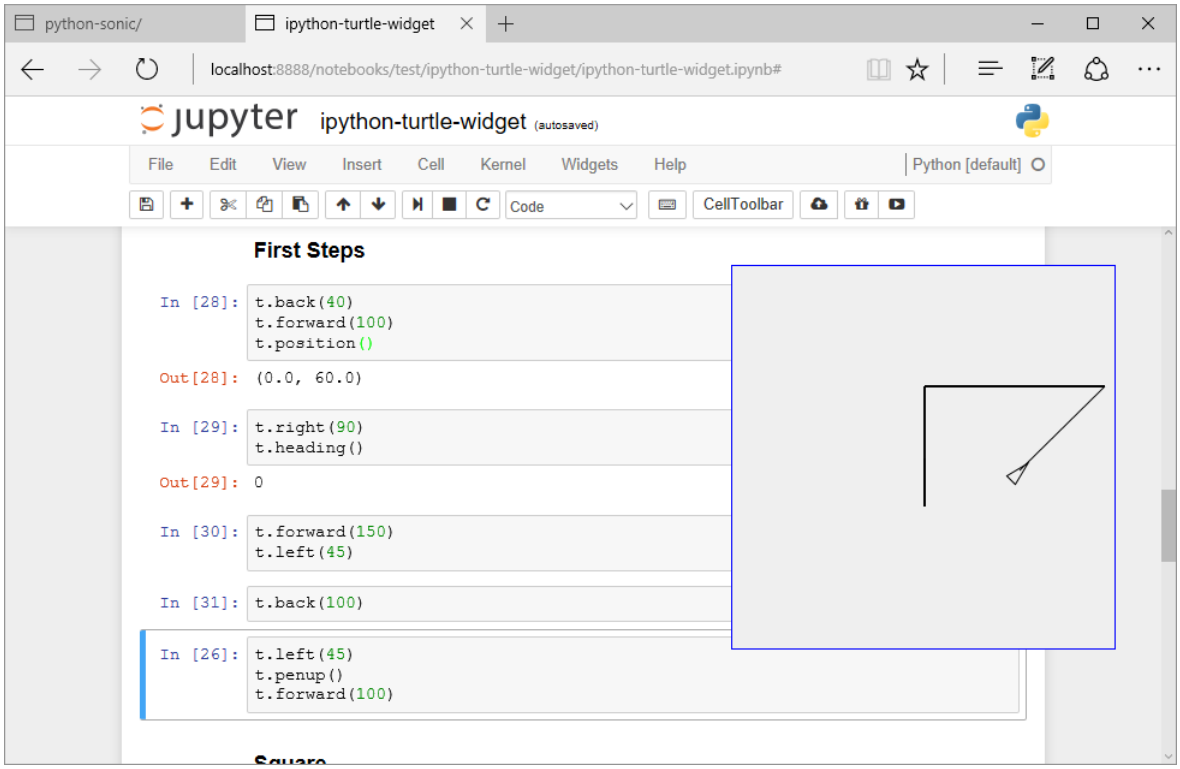
A biblioteca **ipyturtle** (VOLKL, 2019a) é um exemplo de um projeto que possui um módulo em Javascript que é diretamente implementado na arquitetura própria dos widgets para o Jupyter Notebook. Ao analisar o repositório e sua proposta, percebe-se que não houve uma preocupação em componentizar a lógica Turtle, sendo diretamente implementada com o módulo de widgets da biblioteca jupyter-widgets, limitando o reaproveitamento da lógica para outras plataformas.

Com o propósito de auxiliar na criação do *widget*, facilitando o desenvolvimento e mantendo o sistema facilmente acoplavel, é necessário criar um componente web que possa disponibilizar a lógica para ser acoplada em diversas interfaces, podendo integrar até mesmo em outras aplicações, que permitam a disposição do canvas para o desenho.

Levando isso em conta, a melhor forma de estruturar a camada de interface da aplicação que será utilizada de forma a ser reaproveitada para diferentes plataformas é criar em formato de **componente web**, pois dessa forma sua estrutura pode ser facilmente acoplada a uma interface web, como também reaproveitada como um *widget*. Logo, é necessário desenvolver em seu código base a maior parte das funções características da própria linguagem, deixando de forma mais generalista para sua adaptação tanto para as aplicações *web* quanto para os *widgets*.

Após a finalização do componente, é importante que, tanto para integrá-lo como um widget no Jupyter Notebook quanto para incluir dentro de um *website*, o compo-

Figura 10 – Exemplo de código da utilização da biblioteca ipyturtle dentro da interface do Jupyter Notebook (VOLKL, 2019b)



nente seja empacotado e publicado. Para chegar nesse ponto, é importante organizar o desenvolvimento de forma correta para que as funções que ativam comandos específicos na interface de desenvolvimento das imagens sejam acessadas a partir de adaptadores criados para integrar este componente *web*.

As funções foram divididas em três grupos centrais: funções de movimentação, funções de estado e funções auxiliares e de configuração, semelhante com a especificação realizada pela própria biblioteca Turtle (FOUNDATION, 2019).

- **Funções de movimentação**: são tanto funções relacionadas à movimentação do ponteiro quanto funções de ajuste no desenho, na cor do traço, na ativação ou desligamento do pincel para desenho, aumentando a variabilidade de desenhos que podem ser criados.

- **Funções de estado**: são funções específicas a respeito da visibilidade e aparência do ponteiro, como a mudança na disponibilidade do ponteiro, deixando apenas os traços visíveis, quanto na modificação e seu tamanho e forma.

- **Funções auxiliares e de configuração**: são funções de configuração da tela tanto para inicialização do ambiente para desenho quanto para sua reinicialização, criação de formas específicas para serem reutilizadas em vários desenhos, além de também

incluir funções de realização de eventos mais específicos à movimentação através de ações tanto do teclado quanto do *mouse*.

Segundo o guia para desenvolvedores da Fundação Mozilla (PIOVESAN et al., 2022), para a criação do componente web é necessário primeiro se criar a estrutura que queremos componentizar, essa estrutura é definida por meio de uma classe que vai conter todas as ações que o componente deve realizar, e logo após deve-se registrar essa classe como um elemento customizável, esse registro é feito por meio de uma função javascript proporcionada pela própria linguagem. Após feito o registro será incormporado um *Shadow DOM* ao elemento por meio de outra função também disponibilizada pela linguagem JavaScript, dessa forma o elemento pode ser utilizado em interfaces por meio do HTML e para facilitar esse uso deve-se por fim criar um template HTML para esse elemento que pode ser adicionado ao código como uma tag HTML comum, a criação dessa tag tabém é feita por meio de uma função da linguagem JavaScript que suporta todo o ciclo de criação desse tipo de componente.

Como esse projeto consiste em uma implementação de uma ferramenta já existente porém que possa ser executada em outras plataformas, os requisitos são baseados nas implementações originais da linguagem *Logo*, neste componente web vamos implementar as funções descritas nas Tabelas 1, 2 e 3 bem como funções extras de carimbo descritas na Tabela 4.

Tabela 1 – Funções *Turtle* de movimento propostas para implementação.

Função	**Ação**
forward	movimenta a turtle para frente
backward	movimenta a turtle para trás
setPosition	move a turtle para um ponto determinado.

Todas essas funções foram baseadas na implementação original do módulo turtle na linguagem *Logo*. Além dessas funções também faz parte do escopo do componente web a implementação de uma parte gráfica para visualização dessas ações por meio de um canvas onde um objeto se moverá seguindo os comandos utilizados, esse objeto terá dois estados, um estado parado e um estado em movimento, para cada estado será associado uma imagem em formato de sprite para que seja criado um efeito de movimento e deixe a animação do desenho mais interessante.O componente também precisa estar sempre pronto para executar as funções bem como precisa estar exeutando as animações a todo momento.

Tabela 2 – Funções de estado propostas para implementação.

Função	Ação
penUp	impede que as movimentações da turtle desenhe seu caminho na tela
penDown	habilita desenho pelo caminho de movimentação da turtle
setLineColor	muda cor da linha que será desenhada
setLineWidth	muda espessura da linha que será desenhada
speed	muda a velocidade da turtle
right	gira turtle para direita
left	gira turtle para a esquerda

Tabela 3 – Funções de auxiliares de configuração propostas para implementação.

Função	Ação
clear	limpa o caminho da turtle
turtleCommandsList	executa uma lista de comandos
getPosition	retorna posição atual da turtle.

Tabela 4 – Funções especiais de carimbo.

Função	Ação
circle	desenha um círculo
rectangle	desenha um retângulo

3.3.2 Widget para associação do componente web no Jupyter Notebook

Para a criação do widget, é preciso associar uma biblioteca fundamental para implmentar a adaptação correta do componente web com a plataforma Jupyter Notebook. O Backbone.js é uma biblioteca que fornece uma arquitetura MVC, auxiliando na sincronização em tempo real dos dados no *back end* com uma estrutura básica dela no *front end*.

Após esse processo de adaptação, é necessário desenvolver as estruturas básicas das funcionalidades propostas para serem executadas através do Jupyter Notebook por comandos Python, levando em conta todas as funcionalidades citadas no na Subseção 3.3.1. Neste passo, os métodos devem ser organizados dentro de cada pacote específico para que cada tipo de função seja especificada em um módulo específico, a fim de tornar o código mais manutenível e mais compreensivo, separando cada função por responsabilidades.

Durante o desenvolvimento do widget, a aplicação deverá ser empacotada de forma

frequente, sempre que houver uma versão estável com correções e adições de novas funcionalidades, para manter prático o processo de desenvolvimento e produção.

4 Resultados obtidos

Este capítulo trata dos resultados obtidos com o projeto bem como com os insumos desenvolvidos durante a sua realização.

4.1 Desenvolvimento da biblioteca em Javascript

A implementação da biblioteca turtle consiste em uma lista de funções que podem ser utilizadas para desenhar um caminho percorrido pelo objeto na tela que comumente é uma tartaruga. Para realizar a movimentação desse objeto são utilizadas funções simples como "ande para frente", "ande para trás", "gire 'x' graus a direita"e assim por diante permitindo desenhos que se limitam a imaginação do usuário.

Com isso, para a execução das ações do objeto *turtle* na tela, foram implementadas as funções descritas na Tabela 6.

Tabela 5 – Funções *Turtle* implementadas.

Função	**Ação**
forward	movimenta a turtle para frente
backward	movimenta a turtle para trás
setPosition	move a turtle para um ponto determinado
right	gira turtle para direita
left	gira turtle para a esquerda
penUp	impede que as movimentações da turtle desenhe seu caminho na tela
penDown	habilita desenho pelo caminho de movimentação da turtle
turtleCommandsList	recebe uma lista de comandos para serem executados
setLineColor	muda a cor da linha sendo desenhada
setLineWidth	define a espeçura da linha
speed	muda a velocidade que a turtle se move
getPosition	retorna a posição x e y atual da turtle
circle	desenha um círculo
rectangle	desenha um retângulo.

4.1.1 Componente Web

Como objetivo e consequência da criação de um elemento customizado encapsulando ele em uma tag específica, para utilizá-lo deve-se apenas adicionar a tag *<x-turtle>* no código HTML da página onde será integrada.

Algoritmo 2 Exemplo de inserção da tag HTML em uma página Web

```
1 <html>
2     <head></head>
3     <body>
4         <div>
5             <x-turtle width=600 height=800/>
6         </div>
7     </body>
8 </html>
9
```

O componente pode reber alguns parâmetros como altura, largura e a classe do canvas, e dessa forma o canvas é adicionado na tela e já pode ser usado. Porém, é possível também integrar facilmente utilizando apenas JavaScript através da DOM, criando um elemento com a tag customizada.

Algoritmo 3 Instancia do canvas e criação de uma tartaruga em JavaScript

```
1 const canvas = document.createElement('x-turtle');
2 const turtle = canvas.createTurtle();
```

4.1.2 Sincronização de elementos

O componente web executa os comandos de maneira assíncrona, sendo assim, a sincronização dos elementos com os comandos solicitados para execução são de suma impotancia para o entedimento do funcionamento da biblioteca. Isso acontece porque o *Loop* que executa a *turtle* precisa rodar assíncronamente pois de outra forma seria inviável já que o loop infinito consumiria todo o recurso da máquina. Dessa forma toda a execução de comandos fica assíncrona e precisa ser controlada para que tudo execute na ordem correta. O *Loop* é necessário para manter a *turtle* em constante execução bem como para desenhar na tela tanto a linha desenhada quanto a animação.

4.1.2.1 Execução dos comandos

O ponto principal de cada *turtle* construída é que a cada uma é associada uma lista de comandos que o elemento deverá executar. Mantendo essa capacidade juntamente com a adição de comandos à essa lista de forma assíncrona, mesmo que existam mais de uma,

as *turtles* sempre executação seu código de forma separada, sem manter essa dependência de execução de comandos de outra.

Na execução de comandos de forma separada (apenas um comando por vez), o objetivo é que a chamada feita pelo usuário tenha apenas o objetivo de adicionar um comando na lista de comandos do elemento. Inicializado pelo método *runTurtleActionsAndAnimation*, onde os *loopings* de atualização da *turtle* e a animação do Sprite são inicializados, o método de atualização do elemento é o gerenciador de execução de comandos que o objeto está enfileirando em sua lista, tendo sempre como objetivo executar o comando mais antigo e excluí-lo da lista de comandos.

4.1.2.2 Disposição espacial do elemento nos Canvas

Uma das características mais importantes da implementação desse componente é o comportamento dos Canvas em relação aos elementos associados: desenhos e animação das *sprites*.

O componente possui uma estrutura complexa, sendo ela um Canvas compartilhado, para que cada *turtle* consiga aplicar seus desenhos através dos comandos solicitados; e um Canvas particular que tem como responsabilidade desenhar o sprite e gerenciar suas animações.

O Canvas compartilhado é instanciado ao criar o elemento dentro do contexto da DOM da página (através do uso da tag *<x-turtle>*), sendo o elemento mais simples do componente e referenciado justamente como Canvas de *background* pelo fato de agregar o desenho de cada *turtle* instanciado.

Já, o Canvas particular, específico para gerenciar a animação das *sprites*, funciona de uma forma bem complexa quando se agrega mais especificamente o comportamento das animações em relação a orientação da *turtle* dentro do Canvas, ou seja, dependendo da angulação da *turtle*, o Canvas é apresentado de forma diferente.

Com a necessidade de manter essa orientação da *sprite* mais dinâmica dependente da angulação do elemento, este canvas passa por diversas **transladações** e **rotações** para garantir sempre que o sprite acompanhe o desenho mantendo-o no centro do Canvas particular ao executar comandos de movimentação, mudanças de angulação e até mesmo mudanças de posição do elemento no Canvas compartilhado.

Com essa implementação rebuscada é garantido que exista uma imagem final, respectiva ao desenho composto por todos os elementos *turtle* instanciados nesse Canvas. Além disso, garante que a movimentação do elemento ou animação do sprite esteja atrelada apenas ao elemento que está executando essas ações, mantendo essa lógica coesa apenas na *turtle* e desacoplando essa lógica de transposição da instancia em um Canvas específico para ela.

4.1.2.3 Limitações

Como existe essa grande característica de assincronicidade entre as *turtles* criadas e seus comandos executados, muito dos comandos propostos pela biblioteca python Turtle (referencia) têm como objetivo capturar informações atualizadas do elemento (por exemplo, a sua posição no Canvas e a sua angulação atual, sendo dois).

Pelo comportamento de listar os comandos toda vez que são chamados para serem executados posteriormente em ordem, dependendo da forma com que são usados a informação do elemento pode divergir com o que está sendo processado na tela. Por exemplo, se durante a execução de vários comandos de movimentação e mudança de posição do elemento, caso haja a necessidade de saber a posição da *tartaruga* em determinado momento para usar como base de calculo para a posição futura dela, o resultado desta posição atual diverge com a lógica empregada pelo código, que deseja saber a posição após ler e executar os comandos um de cada vez.

4.1.3 Suíte de testes

Para garantir o mínimo de confiabilidade e qualidade do código construído, apesar de que grande parte dos métodos de construção da animação e desenho são privados, através dos métodos públicos que agem como funções acessoras (descritas na subseção 4.1) foram implementados testes unitários com o framework *Jest*, em conjunto com bibliotecas suplementares para testes mais específicos em elementos como o Canvas e a própria DOM, que são dois pontos cruciais que foram bastante trabalhados no desenvolvimento deste componente.

Nesse contexto, os testes foram implementados fora do escopo do código, organizados em uma pasta chamada *tests*, organizando tanto os *mocks* quanto as *snapshots* dentro da pastas de testes. Apesar do teste estar desvinculado do código das funções, os *mocks* e *snapshots* ficam melhor organizados, além de organizar o código no geral em contextos específicos: testes e tudo relacionado na pasta *tests*, código fonte do componente na pasta *source*, entre outros exemplos.

Gerenciando o ambiente de testes para ajudar no desenvolvimento e melhoria da aplicação, foi possível obter uma cobertura de 92.5% de linhas de código, cobrindo grande parte do sistema, com exceção do script para iniciar o ambiente de desenvolvimento do sistema (que tem como objetivo executar um pequeno serviço para executar o html juntamente com o JavaScript na versão *ES6*, permitindo a importação e exportação de arquivos de forma mais simples.

4.2 Integração com ambiente Jupyter

Na implementação de viabilidade de uso do componente no ambiente do Jupyter Notebook, foi criado a implementação **ipyxturtle**.

De acordo com a estrutura de arquivos provida pela biblioteca ipywidgets, o código Python relativo às funcionalidades do componente web se localizam na pasta ipyxturtle, no arquivo widget.py. O padrão de escrita do código python está sendo mantida como *snake case*.

Outro ponto interessante que auxiliou grande parte do desenvolvimento é que para alterações pontuais no código do módulo JavaScript, através de sua integração com os *Watchers* providos pelo **webpacker**, não era necessário a instalação completa da biblioteca local. Logo, para manter mais produtivo o desenvolvimento desse módulo, foi necessário apenas executar o comando já provido pela biblioteca para executar os *watchers* para enxergarem qualquer alteração no código e atualizar diretamente com o pacote Python instalado.

As classes definidas e registradas pela biblioteca integram com o código do módulo Javascript, integrado diretamente ao componente web. Para o escopo do componente, foi necessário a criação de apenas duas classes, que representam de forma literal o que está sendo apresentado: Canvas representando a área de desenho, com suas limitações e propriedades e Turtle para o elemento ao qual será executado os comandos para desenho.

4.2.1 Módulo Python

4.2.1.1 Classe Canvas

Para a criação da área para implementação dos desenhos das tartarugas instanciadas, a classe Canvas tem como responsabilidade criar um elemento HTML do tipo Canvas, definindo sua área e permitindo a estilização do espaço de desenho.

Nesta classe é possível definir a altura e largura do elemento Canvas, assim como sua estilização e a escala do Sprite em relação ao tamanho do canvas, além de permitir a criação de tartarugas para contruir desenhos através de comandos.

Algoritmo 4 Exemplo de código demonstrando a importação de um canvas e criação de uma tartaruga

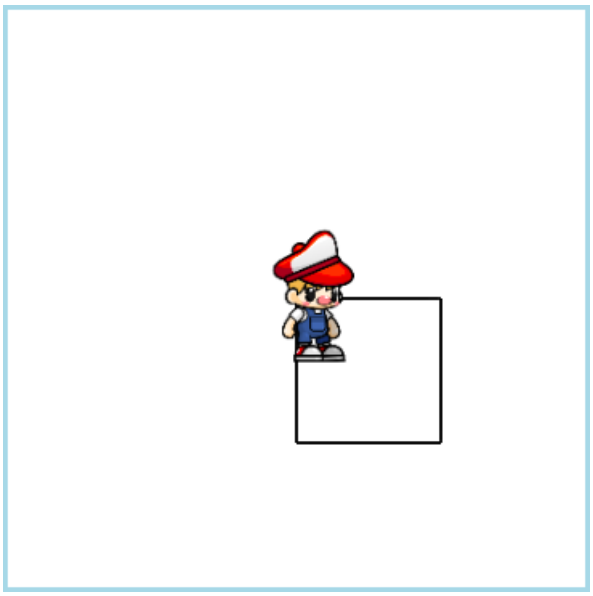
```
1 from ipyxturtle import Canvas
2
3 canvas = Canvas(width=500, height=500)
4 turtle = canvas.create_turtle()
```

4.2.1.2 Classe Turtle

Nesta classe, para integrar devidamente com o módulo Javascript, houve a necessidade de replicar o nome dos comandos para tornar mais similar tanto ao componente quanto à outras bibliotecas que fazem esse mesmo uso do módulo Turtle. Porém, todos os comandos possuem a mesma estrutura de implementação, apenas especificando como seu nome a função que será chamada e os parâmetros passados pela chamada de função. Além dos comandos principais de movimentação, cada Turtle tem uma referência de para qual Canvas o código será aplicado.

Logo, instanciando um Canvas e criando uma Turtle associado à ela, é possível executar todos os comandos disponíveis pelo *widget*. A classe Turtle fica encapsulada por padrão mas pode ser acessada ao ser instanciada pelo método `create_turtle`. A Figura 11 por exemplo foi gerada com o código do Algoritmo 4.

Figura 11 – Exemplo de desenho após execução do código do Algoritmo 4.2.1.2



Algoritmo 5 Exemplo de código completo para o desenho de um quadrado

```
1 from ipyxturtle import Canvas
2
3 canvas = Canvas()
4 turtle = canvas.create_turtle()
5
6 for _ in range(4):
7     turtle.forward(100)
8     turtle.right(90)
```

4.2.1.3 Função lista de comandos

Com objetivo de facilitar o uso da biblioteca e fornecer mais opções de uso foi implementado uma função de lista de comandos que recebe uma lista e vai executando comandos um a um. Essa lista é passada no formato descrito no Algoritmo 4.2.1.3.

Algoritmo 6 Exemplo de lista que pode ser enviada para método de lista de comados

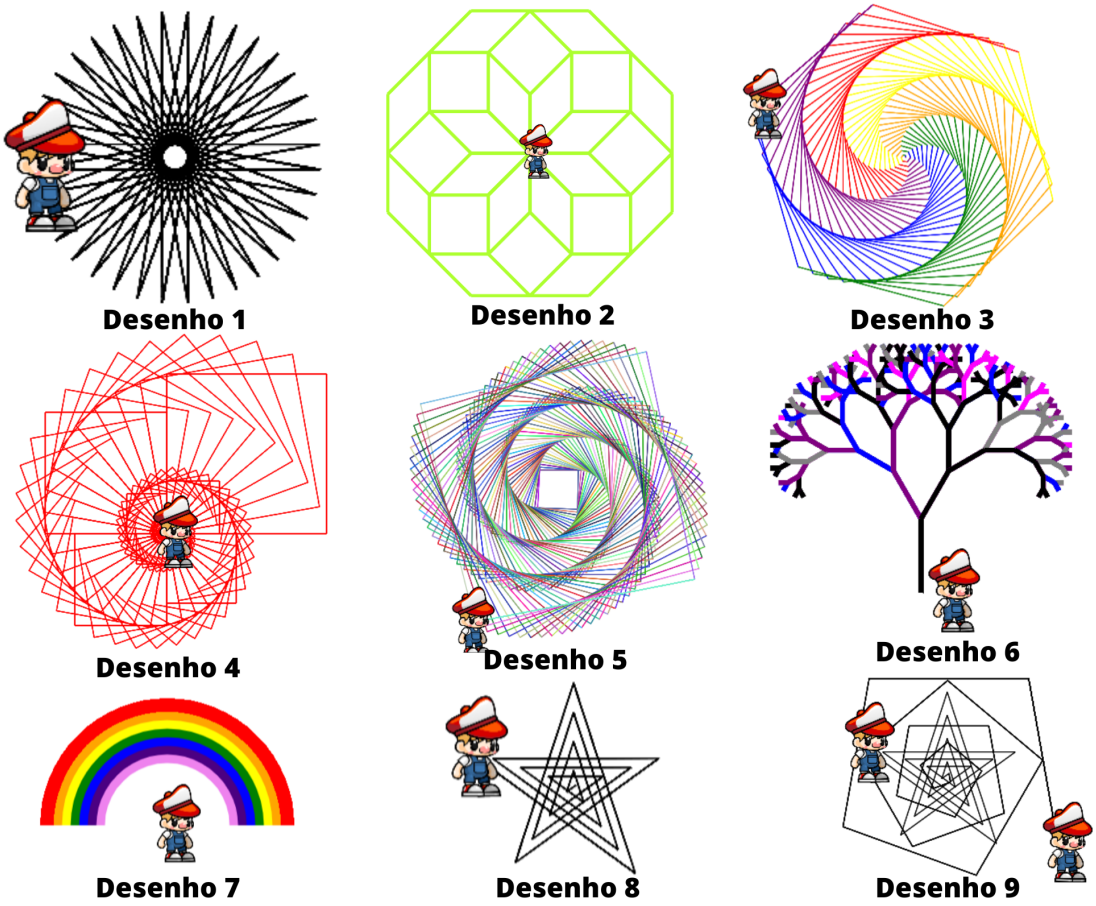
```
1 [{action: 'forwardAction', parameters: [100]}, {action: 'rectangleAction
                                    ', parameters: [50, 50]}, ...]
```

É importante notar que na lista de comandos são passados duas informações para cada comando, o nome do método que deve ser executado com um sulfixo *Action* e uma lista de parametros que pode ter nenhum ou quantos argumentos forem necessários para a função executada.

4.2.1.4 Exemplos e Validação

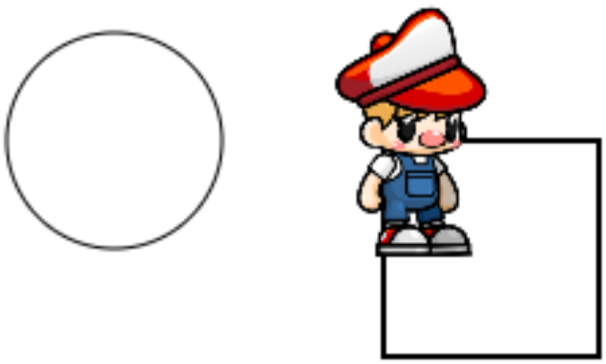
Na Figura 4.2.1.4 pode-se ver exemplos de desenhos que podem ser feitos usando a biblioteca xturtle. As figuras também servem como validação do projeto tanto da parte gráfica quanto das funções que deveriam ser implementadas.

Figura 12 – Exemplo de desenhos usando a nossa biblioteca turtle



Um último desenho mais simples foi feito a fim de cobrir o uso de todas as funções implementadas pela e validar a funcionalidade delas, esse desenho pode ser visto na Figura 4.2.1.4.

Figura 13 – Exemplo de desenhos usando lista de comandos



Para a Figura 13 o código usado foi o seguinte:

Algoritmo 7 Exemplo de lista do método de lista de comandos sendo usado

```
turtle = document.getElementById('canvas').createTurtle()
commands = [
  { action: 'circleAction',
    parameters: [40]
  },
  {
    action: 'penUpAction',
    parameters: []
  },
  { action: 'forwardAction',
    parameters: [100]
  },
  { action: 'penDownAction',
    parameters: []
  },
  {
    action: 'rectangleAction',
    parameters: [80, 80]
  }
]
turtle.turtleCommandsList(commands)
```

Para a Figura 12 os códigos foram os seguintes:

Algoritmo 8 Código do desenho 1 - Estrela de 36 pontas

```
turtle = document.getElementById('canvas').createTurtle()
for(let i = 0; i < 36; i++){
    turtle.forward(200)
    turtle.left(170)
}
```

Algoritmo 9 Código do desenho 2 - Octógono customizado verde

```
turtle = document.getElementById('canvas').createTurtle()
turtle.speed(40)
turtle.setLineColor("greenyellow")
turtle.setLineWidth(5)

for(i=0; i<8;i++){
    turtle.left(45)
    for(j=0; j<8;j++) {
        turtle.forward(100)
        turtle.right(45)
    }
}
```

Algoritmo 10 Código do desenho 3 - Recursão multicolorido em formato hexagonal

```
colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow']
turtle = document.getElementById('canvas').createTurtle()
turtle.speed(100)
turtle.setLineColor('black')
for(x = 0; x<360; x++) {
    turtle.setLineColor(colors[x%6])
    turtle.forward(x)
    turtle.left(59)
}
```

Algoritmo 11 Código do desenho 4 - Espiral de Quadrados

```
turtle = document.getElementById('canvas').createTurtle()
turtle.setLineColor("red")

function drawSquare(length) {
  for(let i = 0; i < 4; i++) {
    turtle.forward(length)
    turtle.left(90)
  }
}

function drawSquareSpiral(num, angle, length, scale) {
  for(let i = 0; i < num; i++) {
    drawSquare(length)
    turtle.left(angle)
    length = scale * length
  }
}

turtle.setPosition(-5, -53)
drawSquareSpiral(90, 10, 200, 0.97)
```

Algoritmo 12 Código do desenho 5 - Múltiplo espiral de quadrados de múltiplas cores

```
function between(min, max) {
  return Math.floor(
    Math.random() * (max - min) + min
  )
}

function valueToHex(c) {
  var hex = c.toString(16);
  return hex
}

function rgbToHex(r, g, b) {
  return(valueToHex(r) + valueToHex(g) + valueToHex(b));
}

turtle = document.getElementById('canvas').createTurtle()
turtle.speed(100)
for(x = 1;  x<=400; x++) {
 r = between(0,255)
 g = between(0,255)
 b = between(0,255)
 turtle.setLineColor(`\#\${rgbToHex(valueToHex(r),valueToHex(g),
                                    valueToHex(b))}`)
 turtle.forward(50 + x)
 turtle.right(91)
}
```

Algoritmo 13 Código do desenho 6 - Árvore binária colorida

```
turtle = document.getElementById('canvas').createTurtle()
turtle.left(90)
turtle.speed(20)
turtle.setLineColor('green')
turtle.setLineWidth(5)

function draw_fractal(blen) {
    sfcolor = ["black", "blue", "purple", "grey", "magenta"]
    const randIndex = Math.floor(Math.random() * sfcolor.length);
    turtle.setLineColor(sfcolor[randIndex])

    if(blen<10){
        return
    }
    else{
        turtle.forward(blen)
        turtle.left(30)
        draw_fractal(3*blen/4)
        turtle.right(60)
        draw_fractal(3*blen/4)
        turtle.left(30)
        turtle.backward(blen)
    }
 }
draw_fractal(80)
```

Algoritmo 14 Código do desenho 7 - Arco-íris

```
turtle = document.getElementById('canvas').createTurtle()
turtle.setLineWidth(15)

col = ['violet', 'indigo', 'blue',
       'green', 'yellow', 'orange', 'red']

for (let i = 0; i < 7; i++) {
    turtle.setLineColor(col[i])

    turtle.circle((10*(i + 8)), -180)
}
```

Algoritmo 15 Código do desenho 8 - Estrela expandida

```
turtle = document.getElementById('canvas').createTurtle()

for(let i = 0; i < 20; i ++) {
    turtle.forward(i * 10)
    turtle.right(144)
}
```

Algoritmo 16 Código do desenho 9 - Multíplas *turtles*

```
turtle = document.getElementById('canvas').createTurtle()
turtle2 =document.getElementById('canvas').createTurtle()

turtle.speed(50)
turtle2.speed(50)

for(let i = 0; i < 20; i++) {
    turtle2.forward(i * 10)
    turtle2.right(80)
    turtle.forward(i * 10)
    turtle.right(144)
}
```

Para utilizar o projeto, basta acoplar a biblioteca implementada no seu projeto web com javascript. Mais detalhes de implementação acesse o repositório do projeto no link <https://github.com/x-turtle> e para instruções de uso temos a documentação do projeto em <https://github.com/x-turtle/xturtle/wiki/Turtle-Functions>.

Também é possível acessar as bibliotecas desenvolvidas diretamente das plataformas de empacotamento utilizadas:

- Biblioteca JavaScript nativa: <https://www.npmjs.com/package/turtle-component>
- Biblioteca Python para executar o código nativo do componente web no ambiente Jupyter Notebook: <https://pypi.org/project/ipyxturtle/>

4.2.2 Módulo Javascript

Este módulo trabalha diretamente com o componente, fazendo a chamada das funções e apresentando na área do notebook onde o Canvas foi definido. Estruturado em arquivos de configuração e um arquivo para fazer a ponte entre o módulo Python e o componente web, este módulo dá a liberdade de organizar o código para que componentes externos consigam ser utilizados facilmente.

4.2.2.1 Model e View

Como este módulo é integrado com a biblioteca **Backbone.js**, a estrutura MVC é aplicada para que toda essa camada lógica interligue as estruturas/comandos no módulo Python para a lógica desenvolvida no módulo JavaScript. Dividida em duas classes principais, este módulo tem:

- a *Model* chamada **CanvasModel**, com o objetivo de gerenciar toda a definição estrutural do Canvas e interligar diretamente com o módulo Python através de seus

métodos getters e setters. Esta classe não depende de uma implementação complexa pois tudo é herdado da classe DOMWidgetModel e funciona como um middleware para o funcionamento do sistema.

- a *View* chamada **CanvasView**, com o objetivo de gerenciar toda a lógica de acesso tanto aos métodos de integração fornecidos pela *model* quanto pelo acesso direto ao Componente Web, permitindo a execução de seus comandos e gerenciando tanto o espaço em que a saída é construída quanto a execução dos comandos executados pelo módulo Python.

Essa integração de execução dos comandos e criação de novas tartarugas entre os módulos funcionava através de campos definidos no módulo Python que funcionavam como observadores, checando sempre qualquer alteração no campo dentro.

Logo, constituído de dois observadores, o sistema checava todas as alterações da variável **command**, relativa aos comandos solicitados para a tartaruga; como também da variável **last_turtle**, mantendo sempre uma referência da última tartaruga criada para identificar quando uma nova tartaruga é instanciada via código Python.

4.3 Comparativo com a biblioteca inspirada

Nosso projeto teve como inspiração a biblioteca Turtle (FOUNDATION,), desenvolvida na linguagem Python. Essa biblioteca é bem completa e documentada, e selecionamos as funcionalidades mais essenciais dela para serem implementadas em nossa própria biblioteca, de modo que seja possível utiliza-la com facilidade.

A biblioteca Turtle possui um total de 90 funções, das quais implementamos 15, representando cerca de 17% das funções disponíveis. Embora seja um número relativamente baixo, muitas das funções que escolhemos implementar têm funcionalidades mais abrangentes que as funções Python mais simples. Por exemplo, a função *setPosition* pode ser usada para substituir as funções *setx* e *sety* da biblioteca Python.

5 Conclusão

A criação de um componente web tem grande importância para permitir que diversos ambientes de desenvolvimento, sejam websites ou sistemas diversos que usam JavaScript possam integrar um sistema de aprendizado de programação mais intuitivo e visual que permite que o aprendiz possa enxergar de forma mais imediata aquilo que ele está programando.

Com essa caracteristica de se tornar um componente em sistemas com possibilidade de execução de códigos JavaScript, é possível, de forma simples, integrar essa ferramenta instalando-a em qualquer serviço e possibilitando a sua integração através de qualquer lógica que aplique uma interface acima dos comandos permitidos pelo componente.

Reforçando a ideia de acoplar em diversos sistemas e aproveitando o contexto de notebooks serem atualmente populares para o ensino de qualquer assunto, a ferramenta *ipyxturtle* pôde servir como prova de conceito de que o componente web pode ser integrado em uma plataforma diferente, como o Jupyter Notebook, apenas dependendo de uma camada de interface que consiga executar os comandos fornecidos pela biblioteca, garantindo também que o código do componente exista sem influenciar diretamente outras lógicas dentro do sistema como um todo.

Importante ressaltar é que esse sistema não se restringe apenas a ambientes que trabalham com a linguagem Python e JavaScript. O ideal é que apartir de qualquer linguagem que esteja diretamente relacionada ao uso de elementos HTML possa utilizar essa ferramenta, permitindo o aprendizado em diversas linguagens de forma imediata e didática.

Por isso, é importante manter esse sistema sempre intuitivo para que novas plataformas consigam integrá-lo, sendo necessário que a organização do sistema como um todo seja bem atualizada, tanto com refatorações que agregam na qualidade do código quanto no cuidado com as documentações, explicando, de forma devida, como funciona e como se integra, tentando sempre tornar todo esse processo mais fácil para que mais pessoas possam contribuir com a melhora desse sistema. Logo, com a proposta de manter tanto o repositório em padrões de software livre quanto a própria organização **x-turtle** sempre trazendo novas exemplificações de serviços que possam agregar essa biblioteca, é possível manter sempre o sistema mais didático, permitindo que outros desenvolvedores também contribuam com essa evolução conjunta do componente e os serviços que agregam ele, criando um ciclo de desenvolvimento e manutenção do sistema principal.

5.1 Trabalhos futuros

Além da possibilidade de melhoria constante no código por meio de atualizações de bibliotecas por exemplo, é importante ressaltar a limitação de assincronicidade (citada na Seção 4.1.2.3) que deve ser repensada para que os usuários possam usufuir de uma experiência completa com todas as funções turtle implementadas.

Além disso, mais diretamente relacionado ao sistema de notebooks proporcionado pela Jupyter, o ideal é que a atual ferramenta que integra o componente Web aos notebooks do Jupyter possa ser facilmente utilizada em plataformas online que usam desse mesmo artefato, como o Google Colab, Amazon Sagemaker, entre outras plataformas que fazem uso notebooks no contexto de aprendizado.

Por parte de novas funcionalidades que são interessantes para agregar no componente é possível ressaltar a capacidade do desenvolvedor colorir dos desenhos através do preenchimento de cor em uma forma definida, não apenas alterando a cor de seus traços, mas também preenchendo uma área completa com uma cor específica.

Portanto, permitindo essa agregação do componente em vários sistemas, mantendo o código-fonte e o repositório bem organizado e manutenível, é ideal também que ele possa ser utilizado como material de aprendizado para pessoas com interesse na área de desenvolvimento de sistemas que vão ter um primeiro contato com programação, através de aplicação de lógicas de programação e matemática na construção de imagens garantindo um feedback mais instantâneo durante a implementação do código.

Referências

BARBA, L. A. et al. *Teaching and Learning with Jupyter - Chapter 1 Introduction*. 2019. Disponível em: <https://jupyter4edu.github.io/jupyter-edu-book/index.html>. Citado na página 25.

BARBA, L. A. et al. *Teaching and Learning with Jupyter - Chapter 2.2 But first, what is Jupyter Notebook?* 2019. Disponível em: <https://jupyter4edu.github.io/jupyter-edu-book/index.html>. Citado na página 25.

CONRAD, S. et al. *Custom Elements*. 2022. <https://developer.mozilla.org/en-US/docs/Web/Web_Components#custom_elements>. Citado na página 26.

CONRAD, S. et al. *Document Object Model - DOM*. 2022. <https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model>. Citado na página 27.

CONRAD, S. et al. *Shadow DOM*. 2022. <https://developer.mozilla.org/en-US/docs/Web/Web_Components#shadow_dom>. Citado na página 28.

CONRAD, S. et al. *Template HTML*. 2022. <https://developer.mozilla.org/en-US/docs/Web/Web_Components#html_templates>. Citado na página 28.

CONRAD, S. et al. *Web Components*. 2022. <https://developer.mozilla.org/en-US/docs/Web/Web_Components>. Citado na página 26.

FOUNDATION, P. S. *Turtle Graphics - Módulo Turtle*. <https://github.com/python/cpython/blob/3.11/Doc/library/turtle.rst>. Citado na página 54.

FOUNDATION, P. S. *Turtle - Turtle Graphics*. 2019. Disponível em: <https://docs.python.org/3.3/library/turtle.html?highlight=turtle>. Citado 3 vezes nas páginas 19, 24 e 36.

GIL, A. *Como elaborar projetos de pesquisa*. Atlas, 2002. ISBN 9788522431694. Disponível em: <https://books.google.com.br/books?id=X4uvAAAACAAJ>. Citado na página 31.

JUPYTER, P. *Jupyter*. 2019. Disponível em: <https://jupyter.org/index.html>. Citado na página 25.

KLUYVER, T. et al. *The Jupyter Notebooks - Introduction*. 2018. Disponível em: <https://jupyter-notebook.readthedocs.io/en/stable/>. Citado na página 24.

KLUYVER, T. et al. *The Jupyter Notebooks - Usage case studies*. 2018. Disponível em: <https://jupyter4edu.github.io/jupyter-edu-book/case-studies.html#interactivity-in-computer-science-high-school-and-middle-school>. Citado na página 26.

LEE, K. *Good map - A simple custom element wrapper for Google Maps JavaScript API*. 2017. <https://github.com/keanulee/good-map>. Citado na página 26.

LOGO, F. *What is Logo? - History*. 2015. Disponível em: <https://el.media.mit.edu/logo-foundation/what_is_logo/history.html>. Citado 2 vezes nas páginas 21 e 23.

MOJANG. *Minecraft Education Edition - A game-based learning platform that promotes creativity, collaboration, and problem-solving in an immersive digital environment*. 2015. Disponível em: <https://education.minecraft.net/>. Citado na página 19.

PIOVESAN, G. et al. *Web Components*. 2022. <https://developer.mozilla.org/pt-BR/docs/Web/Web_Components>. Citado na página 37.

PRACIANO, D. *Mercado de TI tem grande demanda e déficit de novos profissionais*. 2020. <https://brasscom.org.br/mercado-de-ti-tem-grande-demanda-e-deficit-de-novos-profissionais/>. Citado na página 19.

RADIGAN, D. *Kanban - A brief introduction | Atlassian*. 2019. <https://www.atlassian.com/agile/kanban>. Citado na página 31.

SYLVAIN, C. et al. *Building a Custom Widget*. 2022. <https://ipywidgets.readthedocs.io/en/8.0.2/examples/WidgetCustom.html>. Citado na página 29.

VICTOR, B. *Bret Victor - Invent on principle*. 2012. Disponível em: <https://www.youtube.com/watch?v=8QiPFmIMxFc>. Citado na página 23.

VOLKL, G. *ipython-turtle-widget - Creating Turtle Graphics in IPython/Jupyter*. 2019. Disponível em: <https://github.com/gkvoelkl/ipython-turtle-widget>. Citado 2 vezes nas páginas 28 e 35.

VOLKL, G. *ipython-turtle-widget - Creating Turtle Graphics in IPython/Jupyter - Example Image*. 2019. Disponível em: <https://github.com/gkvoelkl/ipython-turtle-widget/blob/master/pic/screen.png>. Citado 2 vezes nas páginas 9 e 36.