Universidade Anhembi Morumbi

Curso de Engenharia de Computação

Projeto Final de Curso de Engenharia

Título:

Jogo educacional em Python para Raspberry Pi: Desenvolvimento de um emulador de jogo em Raspberry Pi e Python com Tkinter, e de tutoriais de como foi feito para ser usado como projeto educacional para aprendizagem de eletrônica e programação, com esquemas e códigos disponibilizados de forma gratuita, para ser adotado por escolas brasileiras

Autores do Projeto Final:

Adriana Laura Cerdeira

Diego Luiz Godoy Portalupi

Leonardo Bueno Santos

Hudson Gustavo Ferreira

Mikaelle Cristina Iosi

Autores do Pré-projeto:

Adriana Laura Cerdeira

Andre de Almeida Alves Batista

Gabriel Campos de Paula

Janderson Barbosa Braz

Thiago Henrique da Cruz

Orientador:

Husani Kamau Poli dos Santos

São Paulo – SP

Março/2024

RESUMO:

O projeto visa o desenvolvimento de um emulador de uma adaptação do jogo “Onde no Mundo está a Carmend San Diego” em Raspberry Pi com seis botões, usando Python, Tkinter e SQLite, além da preparação e disponibilização de tutoriais de como tudo foi feito para que professores e pais possam adotar o projeto para ensinar eletrônica e programação para crianças. Este projeto foi pensado de forma modular e com o menor custo possível. Ele é suportado pelo estudo feito durante a disciplina de Pré-projeto por Cerdeira, et al. apontando a importância do uso de laboratórios makers, laboratórios makers, programação, Raspberry Pi e Arduino, e jogos em escolas.

Palavras chaves: desenvolvimento de projeto em Raspberry Pi; desenvolvimento de projetos maker; Raspberry Pi na educação; ensino de programação para desenvolver pensamento lógico; laboratórios makers em escolas públicas; implementação de laboratórios makers em escolas.

1 – INTRODUÇÃO

Isenção: Os autores deste trabalho e repositório não são responsáveis por qualquer dano causado ao baixar arquivos deste projeto.

1.1 – CONTEXTUALIZAÇÃO

Os benefícios do ensino de eletrônica e programação, a cultura maker, já vem sendo debatidos desde os anos 60 com a teoria construcionista de Seymour Papert e a criação da linguagem Logo. Desde então se fizeram inúmeras pesquisas e artigos a respeito de seus benefícios. Mas sua implementação ainda é escassa, e uma das razões levantadas é a falta de recursos, sejam eles humanos, assim como recursos materiais.

Então, um dos grandes desafios de hoje é trazer essa cultura maker para as escolas ajudando os professores, que por vez podem não ser treinados, ou não possuírem a verba necessária. Nesse sentido, desenvolvemos um projeto de aprendizado de tecnologias usando o desenvolvimento de um jogo pensado para o Raspberry Pi em Python para incentivar o ensino de uma maneira barata, ou até, sem custo adicional. Podendo ser realizada com equipamentos já existentes na casa ou escola, e criando um apelo emocional para sua adoção junto a pais e professores. Usando-se de um jogo clássico, que pode ser adotado por professores de diversas áreas.

Nossa questão inicial foi se era possível fazer isso? Se trazia benefícios para as crianças? Como seria estruturado esse projeto? Qual seria a plataforma escolhida, Raspberry Pi ou Arduino? Qual seria a linguagem de programação escolhida? Qual seria o jogo escolhido? Como deveríamos disponibilizá-lo? Isso tudo respondemos no trabalho de pré-projeto no semestre passado e também abordaremos aqui.

As hipóteses testadas no semestre passado eram de que sim, este tipo de ensino traz muitos benefícios para as crianças. De que poderíamos com certeza fazer um projeto do tipo em Raspberry Pi, e talvez pudesse ser feito em Arduino, o que por sua vez possibilitaria fazê-lo em uma plataforma de prototipagem online como o Tinkercad ou Wokwi. Devido às plataformas, tínhamos a ideia de que entre as melhores linguagens a se escolher deveriam estar o C, Python ou até uma linguagem mais lúdica como o Scratch. Do jogo, tínhamos a hipótese de que o melhor jogo para ser usado como inspiração deveria ser o jogo “Onde no mundo está Carmen SanDiego?” por ter sido um dos jogos educacionais de maior sucesso que

já houve. E por último imaginamos que a melhor maneira de disponibilizar toda a documentação e tutoriais seria com uma mistura de GitHub e YouTube.

No semestre passado pudemos corroborar todas essas hipóteses, como será mostrado mais adiante e chegamos à arquitetura final para a criação do jogo em Python usando o framework Tkinter e o SQLite, e que a melhor plataforma para o emulador seria o Raspberry Pi 3 para frente, com seis botões. Além disso escolhemos o YouTube e o GitHub para hospedar o nosso código e tutoriais, além do resto da documentação.

Este semestre, o Projeto Final foi dedicado ao levantamento dos requisitos de projeto, desenvolvimento do jogo e a criação dos diagramas, dos tutoriais e da documentação. Isto embasado em toda a pesquisa feita no Pré-projeto final, que vamos expor aqui. Além disso, disponibilizaremos aqui o código do jogo e o link para os vídeos tutoriais do YouTube e documentação no GitHub. Também disponibilizaremos os esquemas elétricos e fotos do jogo em funcionamento e protoboard conectada e montada.

1.2 – OBJETIVOS

O objetivo final deste trabalho é o desenvolvimento do jogo e dos materiais educacionais de suporte, tutoriais, documentação, diagramas, etc.

Isso vem suportado pelo objetivo do trabalho anterior, de Pré-projeto, onde definimos a arquitetura a ser usadas e o jogo.

Na etapa inicial de pré-projeto e pesquisa nos propusemos a responder as perguntas levantadas para tomar a decisão destes quatro pontos principais, plataforma, linguagem, jogo e método de disponibilização. E na etapa seguinte, a de projeto final, ocorrendo agora, nos dedicamos ao desenvolvimento do projeto, com este já bem delineado, construindo o projeto eletrônico e de programação, seus diagramas, documentação, além da gravação dos tutoriais e disponibilização do material o GitHub e YouTube.

A ideia foi usar a criação de um emulador, como os jogos de fliperama, ou consoles Nintendo, Gameboy antigos, de algum jogo clássico para criar diferentes etapas de aprendizagem. Desde a montagem do emulador, com uma placa Raspberry Pi 3 ou superior, a programação do jogo em si, sua instalação no emulador e por final o uso do jogo.

Portanto aqui, primeiramente, precisamos responder as perguntas necessárias para que se pudesse decidir qual a estrutura de um projeto destes.

Para tanto, durante o pré-projeto, levantamos vários dados, e tomamos diversas decisões a partir destes dados. Entre eles temos:

- Qual a utilidade e recepção de projetos do maker nas escolas?
- Qual a utilidade e recepção das plataformas Arduino e Raspberry Pi nas escolas?
- A linguagem de programação mais adequada tendo em vista a plataforma usada, o fato de que o jogo não pode rodar somente no emulador, mas também em um computador qualquer e que deve ser de fácil aprendizagem.
- A arquitetura a ser usada (placas, displays, etc.). A Raspberry Pi tem mais possibilidades, mas a Arduino é muito mais barata. Seria possível criar o jogo em Arduino? Seria possível criar o projeto inteiro em uma plataforma de prototipagem online como Tinkercad ou Wokwi?
- Qual o jogo a ser escolhido?

Já na segunda etapa, tivemos o objetivo de desenvolver o projeto eletrônico e de programação delineado a partir destes estudos. Nesta etapa teremos a criação do código, dos diagramas e dos tutoriais a serem disponibilizados para os professores.

Para isso. Também temos que levantar alguns dados. Este não requer uma pesquisa no sentido mais clássico, e sim, parar e entender o jogo para poder responder as perguntas abaixo. Estas perguntas, em um projeto no mercado são a parte de briefing e planejamento do projeto de software e são respondidas através de entrevistas com o cliente. Neste caso, não temos um cliente, mas temos o jogo original. Para tanto, jogamos o jogo original e pensamos em como responder a estas perguntas com o objetivo de adaptar o jogo original para um público de programadores iniciantes e brasileiros. Seguem as perguntas:

- Quais os requisitos funcionais e não funcionais de projeto do software?
- Como deve ser modelada a base de dados?
- Como devem ser modeladas as classes?
- Como deve ser o diagrama de atividades?
- Com deve ser feito o esquema eletrônico?
- Qual a melhor licença a ser usada?

1.3 – HIPÓTESES LEVANTADAS

Ao iniciar nossa pesquisa partimos das seguintes hipóteses:

- O ensino de eletrônica e programação hoje em dia é essencial e deve ser tratado como outra disciplina do ensino básico como matemática, física, história, etc.
- O ensino de eletrônica e programação ajuda as crianças a desenvolverem raciocínio lógico e criatividade.
- O uso de jogos na aprendizagem ajuda na fixação de conteúdo.
- Os jogos de videogame precisam ser adotados pelos professores como uma ferramenta de auxílio no ensino, ao invés de vistos como os inimigos do estudo.
- O uso de placas como Raspberry Pi e Arduino para a criação da parte de eletrônica pode baratear o custo de projetos de ensino de eletrônica e programação, possibilitando sua maior adoção por escolas, principalmente as públicas.
- O uso de linguagens como o Python, que são mais fáceis, pode facilitar a adoção de projetos makers.
- Placas alternativas como Banana Pi e Orange Pi podem ajudar ainda mais a baratear os custos dos projetos makers.
- Laboratórios makers ainda não são amplamente adotados pelas escolas brasileiras, especialmente as públicas, devido ao seu custo e falta de qualificação dos professores.
- Disponibilizar um conteúdo que possibilite a aprendizagem de eletrônica e programação, não só por parte dos alunos, mas dos pais e professores também, ajudaria na disseminação da cultura maker nas escolas brasileiras.
- A criação de conteúdo grátis para o ensino de programação e eletrônica para crianças e adolescentes é muito importante.

1.4 – JUSTIFICATIVA

A criação de conteúdo para o ensino de programação e eletrônica é importante, pois, como já dito, a ideia de programação Arduíno, Raspberry Pi, jogos sendo usados no ensino não é nova. Uma breve pesquisa do Google encontrou artigos científicos sobre os benefícios da introdução de aulas de tecnologia na escola de décadas atrás. Um membro da própria equipe teve sua primeira introdução na escola na década de 90, com a linguagem Logo. E hoje em dia algumas escolas particulares brasileiras já tem alguma parte de programação. Mas essa é a questão aqui, escolas particulares e poucas.

Ao buscar como aprender programação Arduíno e Raspberry Pi nos deparamos com que por mais Open Source que o Arduino se proclame, não há como começar sem pagar um curso fechado, um livro com X projetos, ou montar um Frankstein de tutoriais desconexos abertos.

Isso limita a introdução desse tipo de material em escolas, especialmente as públicas. Limita um pai, uma mãe de decidir que seu filho deveria aprender isto, pois tem um custo, pois eles próprios não entendem como juntar esse material todo em um curso. Um professor que não entenda ele próprio de programação, e eletrônica, como um professor de geografia por exemplo, também teria muita dificuldade em tirar bom proveito, ou introduzir o conteúdo disponível hoje.

Por isso a nossa ideia é desenvolver um projeto, que possa ser pego de qualquer etapa e adaptado à um curso, com uma preocupação de manter uma possiblidade de baixo custo, ou sem custo nenhum. Que ele seja bem explicado, com tutoriais e fornecimento de material, para que mesmo um professor que não seja da área possa aprender com o material, e em torno ensinar seus alunos

Como dito acima, em si a ideia não é original, já há muita discussão sobre a implementação da aprendizagem de programação e eletrônica em escolas, e há bastante material a ser encontrado, em inglês. Em português, e especialmente voltado para a realidade dos estudantes e professores brasileiros, que muitas vezes não sabem inglês e tem pouca verba, há muito pouco material disponível gratuito.

Então é interessante pensar em criar um jogo em português. Instruções de como criar o projeto em português. E permitir às crianças se sentirem realizadas ao conseguir ter algo concreto que construíram para mostrar e usar, e desse modo encorajá-las a continuar com a prática. Rompendo a barreira do idioma e do custo.

E é esse o problema que nos propomos a resolver com este projeto. A criação desse projeto educacional, desse curso aberto de aprendizagem, deste conteúdo a ser consumido.

1.5 – METODOLOGIA DE PESQUISA

Para levantar os dados pesquisados nos utilizamos de quatro técnicas principais.

A revisão da literatura com pesquisas em bases de dados de artigos acadêmicos com as palavras chaves: Arduino na educação, Utilidade de uso de Arduino na educação, Programação na educação, Raspberry Pi na educação, Ensino de programação para desenvolver o pensamento lógico, utilidades de jogos na educação.

Entre os lugares pesquisados estão o Repositório Digital da Ufersa, Repositório Institucional da Universidade Tecnológica Federal do Paraná (RIUT), Biblioteca Digital de Trabalhos Acadêmicos (BDTA), Periódicos científicos UFRGS, Portal de Eventos do Instituto Federal do Rio Grande do Sul, Google livros, Repositório de Produção Científica e Intelectual da Unicamp, Fundação Dialnet, Biblioteca Digital da Sociedade Brasileira de Computação, Repositório Digital Lume, Researchgate - Discover scientific kowledge, Semantic Scholar - Ferramenta de busca de literatura cientifica, Springer Link , Revista do Smithsonian, Renote - Revista de Novas Tecnologias na Educação, Faz Educação & Tecnologia, Google Acadêmico, BDTD - Biblioteca Digital Brasileira de Teses e Dissertações, Periódicos CAPES.

Achamos 36 artigos que nos pareciam relevantes que foram distribuídos entre os membros do grupo para ler, extrair pontos importantes e relevantes à nossa pesquisa, e discussão do grupo.

Aos artigos acadêmicos, juntou-se a leitura de alguns artigos de jornais e revistas acerca da atual situação do ensino maker nas escolas e da situação de recursos das escolas, além de informações dos sites governamentais acerca das atuais políticas de governo em relação a adoção da cultura maker nas escolas.

Esta parte da pesquisa se concentrou principalmente em levantar os benefícios do ensino de eletrônica e programação, os benefícios do uso de jogos na educação e de exemplos de aplicação de tais projetos, especialmente com o uso de Arduino e Raspberry Pi, com interesse nas respostas obtidas das crianças à aplicação desses projetos. Achamos várias informações interessantes que suportavam nossa hipótese da necessidade de criação de conteúdo para estes projetos.

O segundo método usado foi o levantamento de características (datasheets) e custos dos itens necessários como placas, botões, displays, protoboard, cabos, para saber quais variações de projetos poderíamos ter e seus custos.

Fizemos pesquisas sobre as plataformas de prototipagem existentes tanto para Arduino, quanto para Raspberry Pi, com ênfase em buscar plataformas gratuitas.

Pesquisamos a documentação das linguagens e frameworks, além de buscar reviews e tutoriais sobre quais as melhores linguagens e bibliotecas a serem adotados para o ensino de crianças e adolescentes iniciantes na programação.

E por último, após esses levantamentos e a aquisição das peças, fizemos vários testes para ver se era possível criar o jogo escolhido nas placas Arduino, nas plataformas de prototipagem, e nas placas Raspberry Pi.

Esses testes incluíram testes de display, para visualização das informações do jogo e possível envio de instruções, e testes de memória, tanto da própria placa como adicionando memórias externas para ajudar. Esses testes determinaram que infelizmente a plataforma Arduino não será adequada para um jogo do tipo escolhido, o que por sua vez descartou a possibilidade de uso de simuladores gratuitos online como o Tinkercad ou Wokwi como detalharemos mais a frente.

Uma vez definida a arquitetura a ser usada no desenvolvimento do jogo, jogamos o jogo inicial para entender seu funcionamento e definir como por adaptá-lo ao nosso público fazendo o levantamento dos requisitos funcionais e não funcionais do projeto para deste modo poder definir a modelagem da base de dados e os diagramas de atividades, classes e esquema eletrônico.

Por fim, apoiado em todos estes levantamentos, desenvolvemos o jogo, sua documentação e os tutoriais.

1.6 – ESTRUTURA DE CAPÍTULOS DO ARTIGO

No capítulo 2 apresentaremos a fundamentação teórica do trabalho. Nossas hipóteses iniciais e descobertas através da revisão da literatura disponível. No capítulo 3 apresentaremos as plataformas Arduino e Raspberry Pi, assim como os as descobertas feitas a partir do levantamento de dados disponíveis na web acerca dos custos das peças e das plataformas de prototipagem online disponíveis, e os testes de código conduzidos. No capítulo 4 apresentaremos as informações levantadas sobre as linguagens de programação. No capítulo 5

apresentaremos os levantamentos sobre os jogos. No capítulo 6 apresentaremos um resumo dos resultados obtidos, e a estrutura do projeto final desenvolvida a partir deste estudo, e por último, no capítulo 7 apresentaremos a conclusão de todo nosso estudo e resposta do problema de pesquisa levantado na introdução. Ao final anexaremos as tabelas comparativas que criamos e o código do jogo desenvolvido.

1.7 – CONTRIBUIÇÕES TEÓRICAS

Nossa pesquisa irá contribuir não só com um conteúdo educacional que poderá ser adotado por professores brasileiros de maneira modular, mas também pode ser usado de base teórica para a defesa da geração de conteúdos similares.

Isto porque não estamos somente pesquisando e apresentando neste artigo o porquê a cultura maker é importante, mas as vantagens e desvantagens das plataformas disponíveis e o porquê da escolha das plataformas finais. Com suas características e valores.

Ou seja, pessoas com o objetivo de criar conteúdos similares podem usar o nosso estudo para embasar as argumentações das escolhas feitas em seus próprios projetos.

E os professores podem usar o projeto final elaborado, na integra, ou de maneira adaptada às suas necessidades, para ensinar seus alunos e criar neles as competências necessárias para a vida do século XXI, com suas tecnologias e expectativas.

1.8 CONTRIBUIÇÕES PRÁTICAS

Os resultados serão úteis para educadores do ensino fundamental e médio que poderão usar o projeto final na integra ou em partes para ensinar seus alunos as competências necessárias para a vida do século XXI, com suas tecnologias e expectativas.

Além disso, pais que estejam interessados em ajudar seus filhos a aprender eletrônica e programação poderão usar o projeto final para fazê-lo, visto que este é justamente um conteúdo educativo, usando-se de um projeto de criação e programação de um emulador de um jogo educacional para ensinar a eletrônica e programação.

Ou seja, qualquer pessoa interessada em ensinar ou aprender eletrônica ou programação, ou ambos, poderá pegar o nosso projeto para aprender ou ensinar.

E não só o projeto final será útil neste sentido, mas a pesquisa feita na defesa e elaboração do projeto será útil para que outros que queiram criar conteúdos similares possam embasar seus projetos.

2 – FUNDAMENTAÇÃO TEÓRICA

2.1 – O QUE É A CULTURA MAKER E SUA IMPORTÂNCIA NO ENSINO

A cultura maker é uma extensão da cultura “faça você mesmo” e pode ser descrita como uma filosofia onde as pessoas criam artefatos recriados e montados com o auxílio de software e objetos físicos. Ou seja, em sua essência é uma cultura de construção, e não é nova. Agora lhe foi dado um nome em inglês, e parece novidade, mas na realidade já está rodando por aí de uma forma ou outra há bastante tempo.

Quando pensamos em ensino, e em especial o ensino de programação e eletrônica, a cultura maker se relaciona principalmente ao ensino de robótica. E aqui, muitos pensam, que o ensino de robótica quer dizer somente criar robôs no sentido mais clássico de uma representação do ser humano com braços e pernas que se movam, mas não. Em robótica se aprende principalmente programação e eletrônica, e isso inclui a criação de jogos, de emuladores.

A Happy Code[1], por exemplo, uma escola de robótica para crianças, tem em sua apresentação que uma de suas atividades é a criação de jogos para Smartphones. Sim, porque quando falamos de cultura maker, de ensino de robótica, de ensino de programação, estamos sempre falando do mesmo; ensinar às nossas crianças a criar e desenvolverem projetos usando as tecnologias usadas no dia a dia das pessoas hoje. Criar dispositivos para a Internet das Coisas, criar jogos e aplicativos de smartphones, criar plataformas web. Entenderem como funciona o mundo no qual elas vivem.

E por que isso é importante?

Não precisamos ir longe para responder esta pergunta, segundo a Base Nacional Comum Curricular (BNCC)[2], um documento que objetiva nortear o ensino da educação básica temos as dez competências gerais da educação básica, e entre elas encontramos: “*Valorizar e utilizar os conhecimentos historicamente construídos sobre o mundo físico, social, cultural e digital para entender e explicar a realidade"* e *"Compreender, utilizar e criar tecnologias digitais de informação e comunicação"*.

Ou seja, temos um documento governamental que tem como pauta que a criança consiga entender o mundo de hoje. Consiga criar nele. E isso nos diz que laboratórios makers não são somente interessantes para escolas, mas essenciais.

Além disso, a maioria dos artigos pesquisados enfatizavam como em todos os experimentos de aplicação de projetos de eletrônica e programação os alunos demonstraram um grande progresso no desenvolvimento de habilidades como raciocínio lógico, resolução de problemas, criatividade, entre outras.

Em específico, temos um exemplo muito bom em um estudo de implementação de projetos com Arduino aplicada ao ensino de Biologia, conduzido por Asli Görgülü Ari e Gülsüm Meço[3] em uma escola de Istanbul em 2021. Este estudo demonstrou que o desenvolvimento dos projetos da pesquisa pelas crianças aumentou a capacidade das crianças de relacionar causa e efeito. Além disso, entrevistas com as crianças, para estabelecer as vantagens e desvantagens que os alunos perceberam na aplicação do projeto, demonstraram que as atividades os ajudaram a entender a disciplina de uma forma melhor, que se divertiram com as atividades e passaram a gostar mais das aulas de ciência.

Ao decorrer de nossa pesquisa o grupo leu muitos artigos, e o que nos levou a escolher esse como um artigo que resumia e corroborava bem a ideia é justamente o fato de que o projeto de programação e eletrônica não estava sendo aplicado em um curso específico de robótica, ou na matemática ou física, onde costuma acontecer quando não há um curso específico. Mas para auxiliar a disciplina de biologia. Algo que tradicionalmente não é ligado à programação e eletrônica em escolas, demonstrando bem a nossa hipótese de que a aplicação deste tipo de projeto ajuda a educação em geral, e não só um setor específico.

Mas se este tipo de aprendizagem é tão essencial? Se os levantamentos de revisão bibliográfica apontaram que a ideia de que o ensino de programação ajuda a desenvolver o raciocínio lógico, a criatividade e a capacidade de resolução de problemas é algo que vem desde 1960, quando Seymour Papert criou a linguagem Logo, justamente para ser usada em escolas, porque não é mais uma disciplina do currículo obrigatório como matemática ou português?

Essa é justamente uma das questões que este trabalho buscou responder, e foi respondida pela grande maioria dos artigos estudados, mas brilhantemente resumida pela dissertação de Bruna Braga de Paula quando ela disse:

> *“Além da falta de informações, estudos e propostas pedagógicas que possam ajudar os professores a trabalharem com atividades makers, falta também adaptações de atividades com recursos flexíveis, por exemplo, uso de materiais recicláveis.”(BRAGA, 2022)*[3]

E é justamente com o intuito de sanar este problema de falta de recursos flexíveis que estamos fazendo este estudo para poder estruturar o nosso conteúdo de uma maneira que ele fique justamente flexível, com módulos para que o professor possa escolher se quer adotar o projeto inteiro ou só um pedaço. Com tutoriais, para que os professores que não tenham o treinamento, possam usar o próprio projeto para aprender, e depois repassar essa aprendizagem para os alunos. De maneira reciclável, tomando o cuidado de que todo o projeto seja feito pensado em usar ferramentas de prototipagem como protoboards, cabos de conexão, shields, que permitam que nada precise ser soldado, e que todas as peças possam ser reusadas por diversos alunos.

Também estão disponibilizados nos recursos no GitHub os diagramas de modelagem da base de dados, do fluxo das principais funções, de classes, e outras ferramentas de briefing de engenharia de software para que o professor possa inclusive decidir não usar nosso código. Usar somente a ideia do projeto, apresentá-lo para as crianças e ver quais soluções elas idealizam a partir deste briefing.

Ou seja, um projeto realmente modular, para que realmente se tenha um grande poder de decisão de quão fácil ou difícil se quer fazer o projeto, usando de inspiração ou de tutorial completo. Quão alto ou baixo o custo, com diversas possibilidades de construção do jogo. Em que nível de programação e eletrônica. Quer se envolver a eletrônica? Quer se usar só a programação? Ou quer somente usar o jogo para ensinar a matéria?

E novamente foi a Bruna Braga de Paula que resumiu bem a necessidade de que se tenham tantas escolhas, em entrevista dada sobre a sua dissertação de mestrado para Daniel Sanes, da Academia Rhyzos, quando disse:

> *“Claro, mostrar aos alunos que, em vez de uma régua e tesoura, poderiam utilizar uma cortadora laser, por exemplo, é interessante, mas a falta desse instrumento não impossibilita o desenvolvimento do projeto.” (BRAGA, 2022)*[4]

E essa falta de instrumento precisa ser levada a sério porque já é sabido que as escolas públicas brasileiras têm um grande problema de falta de recursos. Só a título de exemplo, de algo que já é senso comum entre o brasileiro, em primeiro de setembro de 2023, o jornal Estadão publicou o seguinte trecho em um artigo justamente sobre a falta de recursos nas escolas estaduais de São Paulo:

> *"Apesar da intenção da secretaria da educação de São Paulo de usar apenas materiais digitais, só 1.953 (39%) das 5,3 mil escolas estaduais têm wi-fi, computadores e outros equipamentos para atender a maioria das turmas."* (CAFARDO,2023)[5]

Por isso mesmo, buscamos levantar as possibilidades mais baratas de criação do projeto, além de ter uma possibilidade que não teria custo adicional nenhum, já que nossa escolas públicas sofrem com a falta de recursos mínimos, quanto mais adicionais.

2.2 – QUAL A UTILIDADE DOS JOGOS NO AMBIENTE EDUCACIONAL

Já discutimos a grande utilidade da aprendizagem de eletrônica e programação no ensino, mas uma parte do projeto foi justamente disponibilizar o jogo em si, para que os alunos possam jogar, mesmo que se decida não fazer a parte de criação. E qual valor isso teria para a educação?

Foi bem interessante descobrir pela revisão da literatura que isto teria um valor muito grande.

Em artigo para a revista "Saberes Docentes em Ação", Caroline diz que:

> *"Os jogos e as brincadeiras têm um papel muito importante na educação infantil e para a vida de uma criança, pois ao brincar a criança espontaneamente adquire uma aprendizagem mais prazerosa, é um momento de comunicação consigo mesma buscando através de sua realidade a sua imaginação." (CAROLINE, 2021)*[6]

Mostrando que o jogo ajuda a criança a aprender de uma maneira mais prazerosa, menos onerosa para ela, e que, portanto, terá melhores resultados do que com aquela repetição maçante do ensino tradicional. No mesmo artigo foi levantado o fato de que os jogos, as

brincadeiras estimulam a criatividade das crianças e as ajudam a fixar conteúdo. Além disso, neste mesmo artigo Caroline faz uma colocação muito relevante para o nosso estudo:

> *"A interação do professor com a criança na sala de aula também é muito importante. Professores acomodados, que se preocupam em terminar logo o dia para irem embora, que se deixam levar pelo método tradicional, que não deixam as crianças brincarem, que acham tudo uma bobagem, acabam prejudicando o desenvolvimento do processo da aprendizagem da criança." (CAROLINE, 2021)*[7]

Ou seja, não é simplesmente o caso de o professor entregar um jogo e deixar a criança brincar enquanto ele faz alguma outra coisa. É importante que ele esteja interagindo na brincadeira com a criança. Por isso a preocupação com o jogo em si. Não só que ele permita uma lógica que a criança consiga acompanhar, mas que ele desperte no professor um sentimento de nostalgia, de algo seu que ele quer compartilhar com seus alunos. Um jogo que ele próprio tenha jogado.

E é nesse ponto que o jogo "Onde está Carmen SanDiego?" tem se apontado como uma das melhores opções. Além de ter sido um jogo educacional de muito sucesso, muitas pessoas acima dos 25 anos lembram de ter jogado. Seja oficialmente enquanto o jogo era vendido, ou em algum emulador. Mais ainda, se lembram de como o jogo as ajudou a estudar geografia.

O sucesso do jogo foi tanto, que ele gerou diversas versões, um jogo de auditório e uma série de desenho animado que foi refeita em 2019 pela Netflix com uma versão interativa.

É difícil você encontrar uma pessoa entre os trinta e cinquenta anos que ao ouvir falar do jogo não abra um sorriso e comente carinhosamente de como jogou ou viu a série quando era criança e adolescente.

Além disso ele é um jogo que atingiu diversas idades, o que é raro quando se fala de crianças e adolescentes. Normalmente você acerta uma faixa exata, devido às grandes mudanças de interesse que ocorrem nestas idades. Mas este foi um jogo que era jogado por crianças de 10 anos e adolescentes de 18. Que ainda é jogado por adultos em emuladores online.

Outro atrativo do jogo é a ideia das pistas. Foi nosso intuito, poder organizar o projeto e a apresentação do material de maneira que o professor possa pegar as informações do jogo e alterá-las para o seu currículo.

Por exemplo, no jogo original, o usuário deveria perseguir a ladra, Carmen SanDiego, ou um de seus comparsas, mundo afora e para isto recebia pistas que muitas vezes continham informações sobre um determinado local.

Desejamos que no nosso jogo, um professor de história, por exemplo, possa cadastrar as cidades de interesse dele e as pistas na base de dados com relação a acontecimentos históricos que aconteceram na cidade, ajudando assim seus alunos a fixarem o conteúdo da disciplina ao jogar.

A revisão da literatura ajudou a fundamentar essa ideia de que o jogo poderia ajudar a criança a estudar como algo já conhecido dos educadores e profissionais da área. E foi interessante descobrir que estes estudos não se limitam a brincadeiras de crianças pequenas, mas a jogos digitais, como demonstraram Savi e Ulbricht em seu artigo sobre Jogos digitais educacionais na revista Renote:

> *"Jogos educacionais bem projetados podem ser criados e utilizados para unir práticas educativas com recursos multimídia em ambientes lúdicos a fim de estimular e enriquecer as atividades de ensino e aprendizagem." (SAVI, 2008)*

Portanto, o jogo em si pode ser uma ferramenta de auxílio ao professor, não somente o projeto de criação do jogo. Além disso, ao final do projeto, a criança acaba com algo que ela vai usar, vai jogar, e não simplesmente guardar em uma gaveta, o que pode deixa-la mais animada ainda com a ideia de continuar a construir coisas com as quais possa interagir.

3 – AS PLATAFORMAS DE PROTOTIPAGEM USADAS NA CULTURA MAKER

Para o projeto, inicialmente cogitamos duas possibilidades para o emulador do jogo; ser feito em Arduino Uno ou Mega ou Raspberry Pi.

A Arduino é uma empresa que desenvolve microcontroladores de código aberto que tem como missão possibilitar que qualquer um melhore a sua vida através de eletrônicos e tecnologia digital acessível.[9] Suas placas têm a características de serem fáceis de usar e de código aberto, e, portanto, podem ser copiadas e vendidas por outras empresas, fazendo com que não seja necessária a importação, pois se encontram versões nacionais da placa. Isto faz com que suas

placas sejam muito baratas. A placa Arduino Uno Rev3 original comprada pelo grupo para testes custou em torno de R$ 290,00, com frete e importação. Já sua versão nacional da Casa da Robótica custou R$ 82,79 com frete.

Além de seu custo baixo, outra vantagem do Arduino era o fato de que há plataformas online de prototipagem grátis, como o Tinkercad e Wokwi, que permitem que você faça todo o projeto de eletrônica e simule a execução do código de forma completamente gratuita, sem precisar realmente comprar as placas. Por isso o nosso grande interesse em testar a possibilidade de criar o jogo na placa apesar de ser um micrcontrolador e não um microprocessador.

A placa Raspberry Pi por sua vez, foi criada pela Raspberry Pi Foundation[10] com o intuito de incentivar a aprendizagem de programação criando um computador completo em uma única placa, que fosse do tamanho de um cartão de crédito, e que custasse o mesmo que um livro didático. E foi assim mesmo que sua primeira versão foi lançada. Hoje já temos diversas versões, com mais ou menos memória. Hoje, uma Raspberry Pi comprada no Brasil pode variar entre R$ 500,00 e R$ 1000,00, segundo nossas pesquisas.

Porém ela é um projeto proprietário. Isso não impediu algumas empresas de fazer engenharia reversa e criarem as suas próprias placas alternativas como a Orange Pi e Banana Pi, que podem ser encontradas com valores menores, a partir de R$ 290,00.

A grande diferença entre uma placa Arduino e uma Raspberry Pi não é só o preço, mas sua memória e processamento. A placa Arduino é um microcontrolador, idealizado para controlar periféricos com um programa simples e leve. Uma placa Arduino Uno tem uma memória de armazenamento de 32KB e uma memória RAM de 2KB. Já as placas Raspberry Pi hoje no mercado têm memórias RAM que variam de 1GB a 16GB, e seu armazenamento vai depender do tamanho do cartão SD escolhido pelo usuário. Uma placa Arduino é monoprogramada, podendo executar uma ação por vez, já uma placa Raspberry Pi tem um sistema operacional multitarefas.

É inegável que uma placa Raspberry Pi seria muito mais indicada para a criação do emulador por permitir uma performance melhor para o jogo. Não precisamos nem de testes para isso, está simples apresentação das diferenças já nos diz isto.

Porém as placas Raspberry Pi têm um defeito muito grande: não há uma plataforma de simulador da eletrônica e código grátis disponível. O máximo que conseguimos são máquinas virtuais que nos permitem baixar o Raspberry Pi OS.

E um de nossos pilares principais neste projeto era o custo final do projeto, e por isso queríamos ver se havia a possibilidade de “subverter” a função de um microcontrolador para o nosso caso. Nesse sentido, foi importante para o grupo testar o código e a visualização do jogo em placas Arduino. Especialmente pois encontramos exemplos online de pessoas que haviam usado o Arduino para recriar o jogo Mario, da Nintendo em Arduino. Pensamos que talvez fosse possível para o nosso caso também.

Foram conduzidos testes nas placas Arduino Uno e Arduino Mega. Foram conduzidos testes de visualização de display, logo chegando à conclusão que se necessitaria de um display mínimo de 2,8”. Isto descartou a possibilidade do uso da plataforma Tinkercad, visto que o maior display que eles possuem é o de 16 colunas por 2 linhas, e testes neste display mostraram não ser possível entender as instruções do jogo com tão poucas linhas.

Por sua vez, testes de memória descartaram de vez a plataforma Arduino, pois o Arduino só permite rodar programas de sua memória flash, e não de um cartão SD card. O que faz com que o programa precisasse ser muito leve e isso não foi possível. Mesmo com testes onde foram colocadas todas as informações de strings guardadas em um cartão SD card, indo buscar a informação sempre que se necessitava um texto, para deixar o programa mais leve. Isto, além de deixar o programa muito lento, devido a tantas buscas, também não resultou em um programa leve o suficiente mesmo com a implantação parcial do jogo.

Isto levou a conclusão que a placa a ser usada deverá ser uma placa Raspberry Pi. A placa comprada para o projeto é uma placa Raspberry Pi 4 com 8GB de memória, porém a construção do projeto levou em conta a necessidade de manter o código o mais leve possível para que ele possa rodar em placas com menos memória.

Foi escolhida a placa Raspberry Pi por ser uma plataforma respaldada com artigos científicos, porém, isto não impede que, ao adotar o projeto, os professores usem uma de suas alternativas como a Orange Pi, que apresenta um excelente custo-benefício. Ela possui uma placa com uma versão mais potente, por um valor menor do que a placa Raspberry Pi com menor memória. Além disso, já possui memória interna, sem a necessidade do SD Card. Ela não foi escolhida pelo projeto devido à escassez de artigos acadêmicos respaldando seu uso. Foram encontrados somente 2 artigos acadêmicos sobre seu uso em educação, um em português e um

em inglês. Enquanto da Raspberry Pi foram encontrados diversos artigos demonstrando que ela tem uma grande aceitação por parte de alunos e educadores.

Temos em Santos (2018)[11] que *"O diferencial desse dispositivo é ser um computador completo do tamanho de um cartão de crédito que pode ser armazenado em qualquer lugar evitando assim, possíveis roubos."* Fazendo do Raspberry Pi uma escolha excelente para uso em escolas públicas, onde esta é uma preocupação, como mencionou Cafardo no artigo do Estadão onde disse que *"(...)por falta de segurança – a escola já foi assaltada vária vezes e teve até fiações roubadas(...)"*

Além disso ela é de muito fácil uso, e possui as portas GPIO, permitindo, assim como o Arduino, controlar botões, displays, sensores e outros periféricos.

4 – AS LINGUAGENS DE PROGRAMAÇÃO E SUA DIFICULDADE E FACILIDADE

Inicialmente havia sido levantada a hipótese de que o jogo e o emulador deveria ser programado em C, Python ou Scratch.

O C foi cogitado, apesar de sua dificuldade, por ser a linguagem principal usada para programar diretamente em Arduino. Especialmente pelo fato de que o programa em Arduino, se fosse possível, precisaria ser leve, e, portanto, não precisaria de bibliotecas extras para fazer a tradução de Scratch ou Python para C. Porém, uma vez que foi estabelecido que o Arduino não se adequava ao projeto, foi descartada por ser uma linguagem de difícil aprendizado.

O Scratch também foi apontado por diversas pesquisas online, e por vários dos artigos estudados, como sendo uma linguagem amigável para as crianças por ser uma linguagem visual, que se usa de blocos que você arrasta para criar a sua lógica. Porém, pesquisas da documentação e fóruns levantaram o fato de que não é possível conectar com uma base de dados SQL com o Scratch. São necessários APIs externas em outras linguagens, o que já traria um nível de complicação que faz com que seja contraproducente.

E o projeto se usa de uma base de dados local SQLite para as informações de jogo.

Por fim, após várias consultas à fóruns de programação e a leitura da própria literatura acadêmica, foi decidido que o Python seria a linguagem utilizada, com a biblioteca Tkinter

para a criação do front-end, por ser uma linguagem fácil de aprender, e para a qual o Raspberry Pi já vem otimizado.

Além disso se usou um pouco de SQL para a busca das informações na base de dados.

5 – AS POSSIBILIDADES DE JOGOS

Entre os jogos educacionais populares dos anos 80 e 90, os mais mencionados são *"Oregon Trail"* (Trilha de Oregon) e *"Where in the World is Carmen SanDiego?"* (Onde no mundo está a Carmen SanDiego).

No artigo da BuzzFeed, Barrios(2023)[12] resume bem falando:

> *"'Aprender é divertido!'...nunca disse uma criança. (...) Ainda assim, mesmo considerando que o edu-lazer era um cavalo de Tróia de lições de matemática ou história disfarçadas de gráficos em blocos dos anos 90, ninguém pode negar que passou horas na Trilha Oregon ou tentando encontrar a Carmen SanDiego" (BARRIOS, 2023)*

Não foram os únicos jogos mencionados, mas o grupo chegou à conclusão, que entre os jogos comumente mencionados eram os mais emblemáticos e que podiam ser adotados por outras disciplinas. Isso porque os outros jogos tinham a tendência a serem voltados ou para a matemática ou gramática, limitando seu uso.

O "*Oregon Trail*"[13], era uma série de jogos educacionais onde o jogador toma o lugar de um líder de carroças levando um grupo de colonos desde o Missouri até o Oregon, e com isso ensinava a história americana. Foi um jogo muito popular nos Estados Unidos, mas é bem específico para o país. Além disso, devido ao tema do jogo, a colonização do Oeste Americano, ele foi classificado como racista e culturalmente insensível.

Já o jogo "*Where in the World is Carmen SanDiego?*"[14] era um jogo que foi lançado em 1985, onde a premissa era que o jogador deveria encontrar e capturar a Carmen SanDiego, uma ladra internacional, dentro de um certo tempo. Para isso, ele deveria perseguir a Carmen, ou um de seus comparsas, mundo afora.

Ele foi primeiramente distribuído com o Almanaque Mundial e Livro de Fatos Mundial, uma publicação americana com fatos mundiais.

O jogo teve muito sucesso no mundo inteiro e diversas edições. Além disso gerou um show de auditório e um desenho animado na década de 90, e um reboot em uma série da Netflix em 2019, com uma edição especial e limitada do jogo para ser jogada no Google Earth.

O interessante do jogo é que as pistas sempre continham curiosidades sobre cidades e países, o que fazia com que a criança jogasse com um Atlas e uma enciclopédia do lado. Encorajando-a a buscar informações sobre as cidades ou países para onde o jogador poderia "viajar" para capturar a Carmen. Além, das próprias curiosidades mostradas quando um jogador chegava a um lugar.

Ou seja, era um jogo bem educacional, pois além de ensinar coisas sobre os locais, incentivava a criança a pesquisar.

Ele ainda pode ser jogado hoje em dia em emuladores online, além de ser muito fácil de encontrar vídeos de pessoas que jogaram o jogo em algum emulador.

As pesquisas online também levantaram dois sites em português que oferecem jogos online que se propões a serem educativos. O https://www.escolagames.com.br/jogos-educativos e o https://www.ludoeducativo.com.br/pt/.

Isso já demonstra o interesse das crianças em jogar, então é mais interessante ainda que ao aprenderem a programar, elas terminem com um produto final que vão querer usar, e não só com algo para colocar na gaveta.

Os sites eram bem interessantes para crianças menores, porém um pouco ainda com aquela ideia de que você está fazendo lição de casa. Para passar as etapas, você devia responder a uma provinha. Ajuda a estudar, mas vai ser lição. E como disse Barrios(2023) "*'Aprender é divertido!'...nunca disse uma criança."*

Além disso, havia muita propaganda em ambos os sites, e nem sempre voltada para crianças. O que pode inibir pais e professores de adotarem estes sites, apesar de serem projetos apoiados pela Fapesp e CNPq. Mas isto já indica como os próprios educadores estão começando a ver os videogames como aliados ao invés de inimigos da educação.

Então percebemos que a ideia de usar jogos na educação não é nova, e ter um projeto que o professor pode adaptar para si é bem atual e interessante. E que como já é bem conhecida, a necessidade de prover isto para as crianças, a criação de conteúdo ofertado de forma gratuita para os professores poderem adotar, continua sendo de imensa importância.

6 – RESUMO DA PESQUISA E ESCOLHA DA ESTRUTURA FINAL DO PROJETO

6.1 – RESUMO E ESTRUTURA DO JOGO

A pesquisa levantou que o ensino de programação e eletrônica, além de ser muito importante para o desenvolvimento das crianças, e para seu entendimento do mundo em que vivem, é muito bem recebido por elas. É considerado uma atividade divertida que as empolga.

A pesquisa também identificou que o uso de jogos na educação ajuda não só a fixar o conteúdo, mas a desenvolver o relacionamento entre professores e alunos. O que fundamentou a ideia da escolha de um jogo nostálgico, para que os professores fiquem incentivados em compartilhar uma experiência da sua infância com seus alunos. E também justificou a ideia de usar um jogo educacional, que depois de criado poderá ser jogado pelas crianças e adolescentes, ajudando-as a estudar.

A revisão da literatura também mostrou que a plataforma Raspberry Pi é muito bem recebida nas escolas por ser de baixo custo, e fácil uso. E que juntamente com a linguagem Python, seria uma boa escolha para o projeto.

Os testes práticos infelizmente descartaram a possibilidade de fazer o projeto em Arduino ou Tinkercad.

Ao pesquisar jogos dos anos 80 e 90, a possível infância de grande parte dos educadores, foi levantado que a melhor possibilidade de jogo a ser adaptado seria o "*Where in the Wolrd is Carmen SanDiego?".*

E por fim, o Python foi levantado como sendo a linguagem mais adequada para a criação do jogo, tanto por facilidade de aprendizado, como por otimização da placa Raspberry Pi.

Sendo assim, a estrutura final do jogo escolhida ficou da seguinte forma.

Vamos começar com o jogo. Após pesquisas por jogos educacionais que tiveram sucesso com as crianças e adolescentes descobrimos que o jogo *"Where in the World is Carmen SanDiego?"* (Onde no mundo está a Carmen SanDiego?) lançado em 1985, e muito popular nos anos 90, ainda é considerado um dos melhores jogos educacionais e o caso de maior sucesso de um jogo educacional.

Portanto o jogo criado para este projeto foi inspirado neste jogo, porém simplificado para facilitar a criação da lógica por um iniciante. Além disso alteramos o nome para, “Cadê a Zefa, gente?”. Não será a Interpol a perseguir a Zefa, notória ladra internacional de origem brasileira que atua mundo afora, mas a polícia federal que a está perseguindo desde que ela roubou a taça Jules Rimet com seu pai (um quinto integrante do grupo que a PF nunca divulgou e que escapou), em 1983 na tenra idade de 10 anos. Desde então ela tem aplicado roubos cada vez mais ousados pelo Brasil e o mundo e a nossa dedicada PF nunca deixou de persegui-la décadas afora. Será o nosso detetive intrépido a pessoa que finalmente irá apreender está elusiva ladra?

Além disso, alteraremos o layout do jogo, sem seguir a tradicional tela do videofone.

Isso para evitar problemas de direitos autorais, apesar de que este jogo não foi feito para ter fins lucrativos. Muito pelo contrário, o código foi disponibilizado baixo a licença GNU GPL (Licença Geral Pública)[15], de forma completamente gratuita. Foi escolhida a licença GNU GPL por ela justamente especificar que qualquer código derivado de um código com uma licença GPL deve ser também fornecido completamente aberto baixo a mesma licença. Ou seja, ele poderá ser alterado e redistribuído, desde que continue com seu código completamente aberto, e que o hardware no qual ele vá não seja alterado para impedir alterações no código.

O jogo foi criado em Python, com 3 versões.

Uma versão que pode ser jogada na linha de comando, simplesmente de texto, com inputs textuais do usuário.

Uma versão para desktop multiplataforma com uma interface gráfica desenvolvida com a biblioteca Tkinter. Essa biblioteca foi escolhida por ser simples, de fácil uso, e já vir com a instalação do Python, não necessitando nova instalação.

E a terceira versão é o emulador físico do jogo que foi criado para Raspberry Pi da versão 3B para cima, podendo ser implementado com uma de suas alternativas como Banana Pi, ou Orange Pi. Ela possui seis botões sendo 3 de escolha de opções, 1 de fechar o jogo, 1 para voltar no menu anterior, e 1 para um novo jogo. Ele usa um display de 7” com entrada HDMI.

Incialmente foi cogitado usar um display menor, que não permite o uso de grandes gráficos, mas que é mais barato, TFT de 2.8”. Porém ao desenvolver o programa para este display, percebemos que as bibliotecas necessárias para usar o display e a programação feita, além de

ser complexa, foi difícil de ser realizada por uma escassez muito grande de documentação das bibliotecas.

Como o projeto é idealizado para introdução a programação, com alunos iniciantes, ficou decidido que esta dificuldade pode frustrar e desencorajar a criança ou adolescente a seguir o projeto, e portanto decidimos não fazer está versão do emulador, apesar de seu custo menor.

A estrutura do código foi pensada para o maior reaproveitamento possível sendo separada no modelo MVC (Model/View/Controller ou Modelo/Visualização/Controlador). As classes do Modelo são as mesmas para as três versões, tendo alterações grandes das classes de Controlador e Visualização entre as versões de interface gráfica e de linha de comando, e poucas alterações entre as duas versões diferentes de interface gráfica, sendo que a versão do emulador constrói em cima da interface de desktop. Para tanto ela utiliza-se do artifício da função invoke para invocar as chamadas aos botões da interface através dos botões do GPIO. Deste modo há apenas adição de alguns métodos na classe de controlador de Fluxo de Jogo do emulador, sendo que está é filha da classe de Fluxo de Jogo da interface de Desktop. E há umas pequenas alterações nas classes de Visualização das telas.

Foram disponibilizados os diagramas de base de dados, classes e funções principais, e as imagens das telas do jogo para que os professores consigam entender bem como foi pensada a lógica do jogo, e até decidir se querem usar o desenvolvimento do próprio código como desafio para seus alunos, usando o nosso como gabarito.

Além disso foi disponibilizada uma base de dados SQLite já pronta para uso ou alteração. A base SQLite foi escolhida por ser local e não necessitar de conexão com a internet.

Foram criados vídeos tutoriais que estão hospedados no YouTube de como criar o projeto inteiro, com links para cursos gratuitos de boa qualidade encontrados pelo grupo sobre as linguagens e plataformas base usadas no projeto.

Esse trabalho todo de criação do projeto (diagramas, código, tutoriais) foi disponibilizado no repositório do GitHub abaixo:

https://github.com/computerbuddybr/Cade_A_Zefa

Sendo que os vídeos tutoriais foram disponibilizados no canal do YouTube de um dos integrantes do curso, em uma Playlist no link:

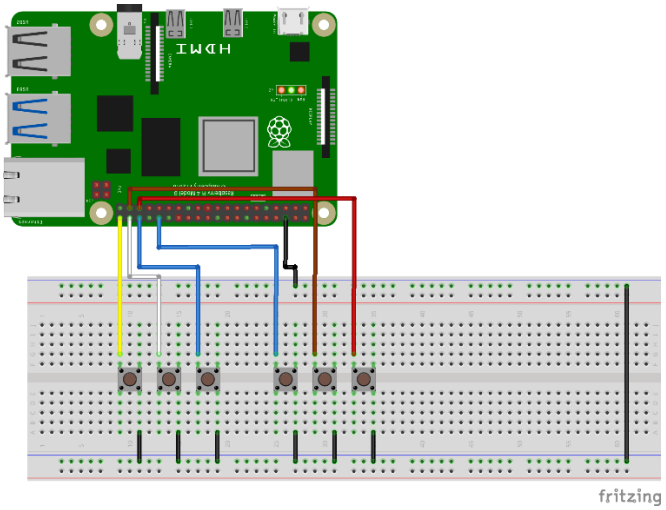
https://www.youtube.com/playlist?list=PLhx-V5qg9T6RO1tbo4svEUJkJL4f0R207

6.2 – ESQUEMAS ELETRÔNICOS DAS LIGAÇÕES DOS BOTÕES NA PLACA RASPBERRY PI

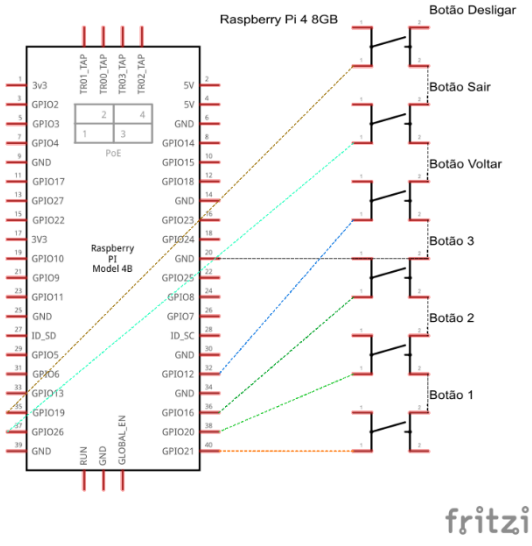
Total 6 botões:

- Botão 1
- Botão 2
- Botão 3
- Botão v (voltar)
- Botão s (sair)
- Botão d (desligar)

Os botões foram projetados para serem exatamente os mesmos das opções de testes na linha de comando e nas opções do Jogo para desktop. Deste modo, foi possível reaproveitar o código de lógica da criação de jogo (backend), parte da lógica do fluxo do jogo (backend), alterando somente parte das classes de visualização (front-end) e de fluxo de jogo.

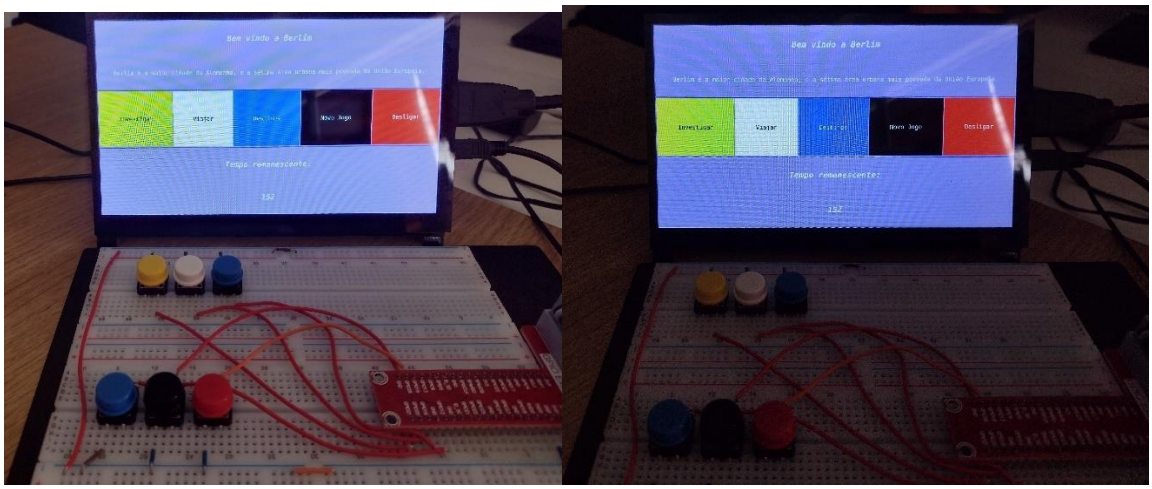


Ligação dos botões na protoboard e na Raspberry Pi 4

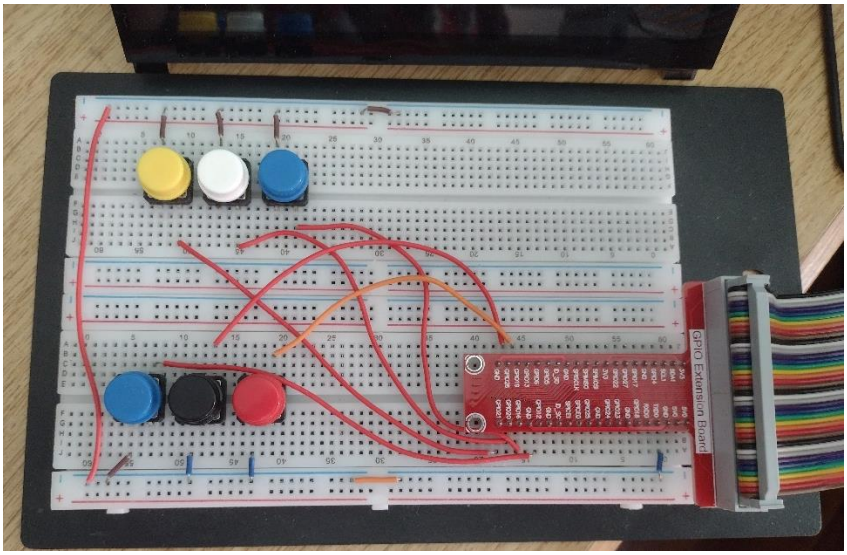


Esquema de ligação dos botões à placa Raspberry Pi 4

6.4 – IMAGENS DOS BOTÕES, RASPBERRY PI E DISPLAY 7” MONTADOS



Protoboard, placa adaptadora Raspberry Pi, 6 botões e Display HDMI 7” com Jogo funcionando



Protoboard, placa adaptadora Raspberry Pi, 6 botões

Raspberry Pi 4 8GB em case dissipador de calor

6.5 – IMAGENS DAS TELAS DO JOGO NO DISPLAY 7” E COMPUTADOR

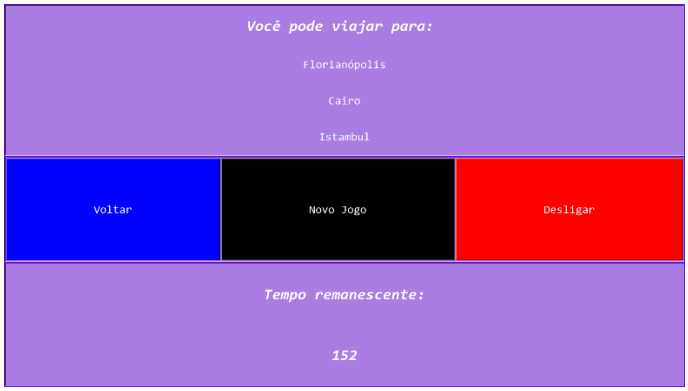
Tela Inicial

Menu principal do Jogo

Tela Investigar

Tela Pista

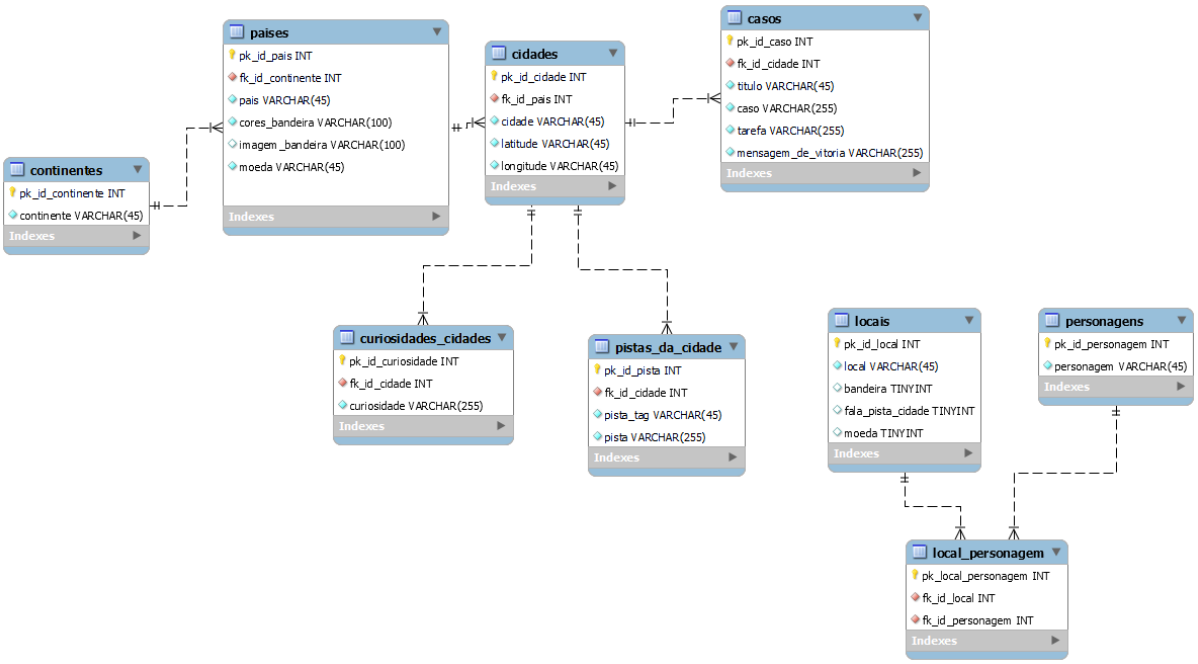
Tela Viajar



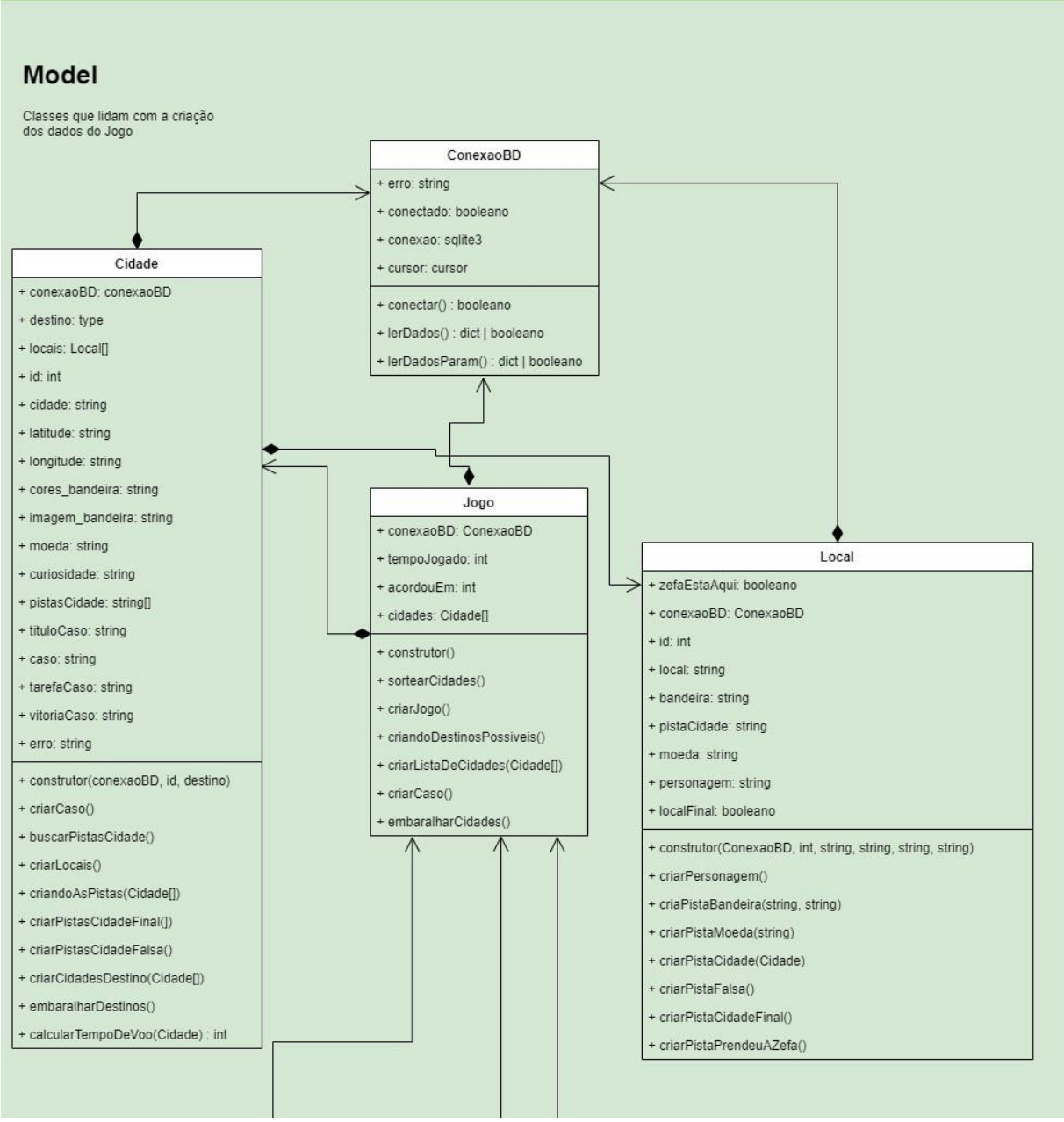
Tela Destinos

Tela Final

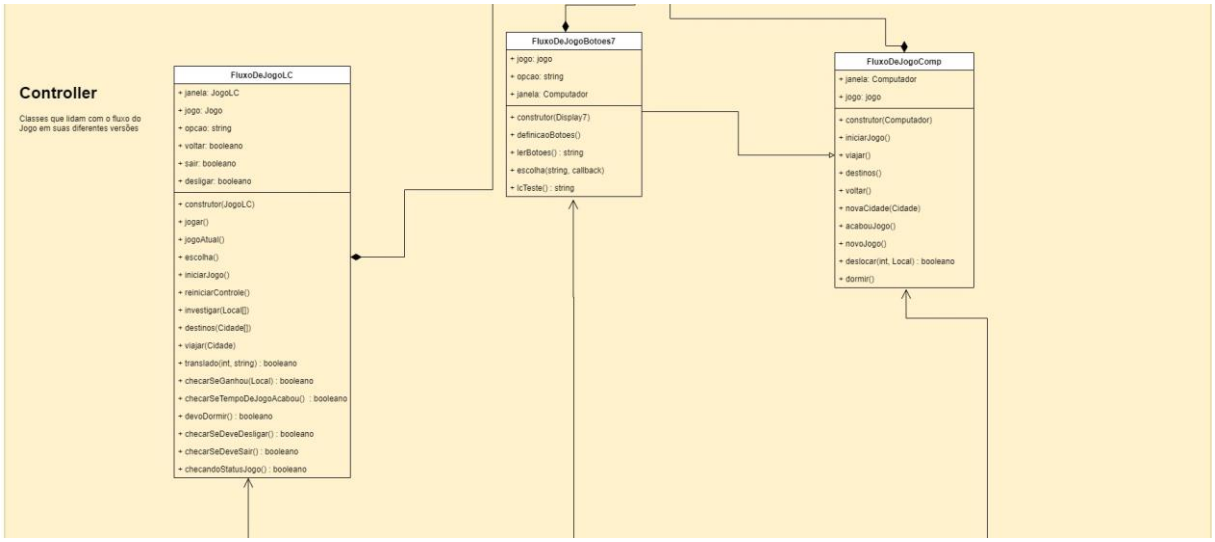
6.6 – MODELO DE BASE DE DADOS RELACIONAL


continentes
pk_id_continente INT
continente VARCHAR(45)
Indexes
paises
pk_id_pais INT
fk_id_continente INT
pais VARCHAR(45)
cores_bandeira VARCHAR(100)
imagem _bandeira VARCHAR(100)
moeda VARCHAR(45)
Indexes
cidades
pk_id_cidade INT
fk_id_pais INT
cidade VARCHAR(45)
latitude VARCHAR(45)
longitude VARCHAR(45)
Indexes
casos
pk_id_caso INT
fk_id_cidade INT
titulo VARCHAR(45)
caso VARCHAR(255)
tarefa VARCHAR(255)
mensagem_de_vitoria VARCHAR(255)
Indexes
curiosidades_cidades
pk_id_curiosidade INT
fk_id_cidade INT
curiosidade VARCHAR(255)
Indexes
pistas_da_cidade
pk_id_pista INT
fk_id_cidade INT
pista_tag VARCHAR(45)
pista VARCHAR(255)
Indexes
locais
pk_id_local INT
local VARCHAR(45)
bandeira TINYINT
fala_pista_cidade TINYINT
moeda TINYINT
Indexes
personagens
pk_id_personagem INT
personagem VARCHAR(45)
Indexes
local_personagem
pk_local_personagem INT
fk_id_local INT
fk_id_personagem INT
Indexes

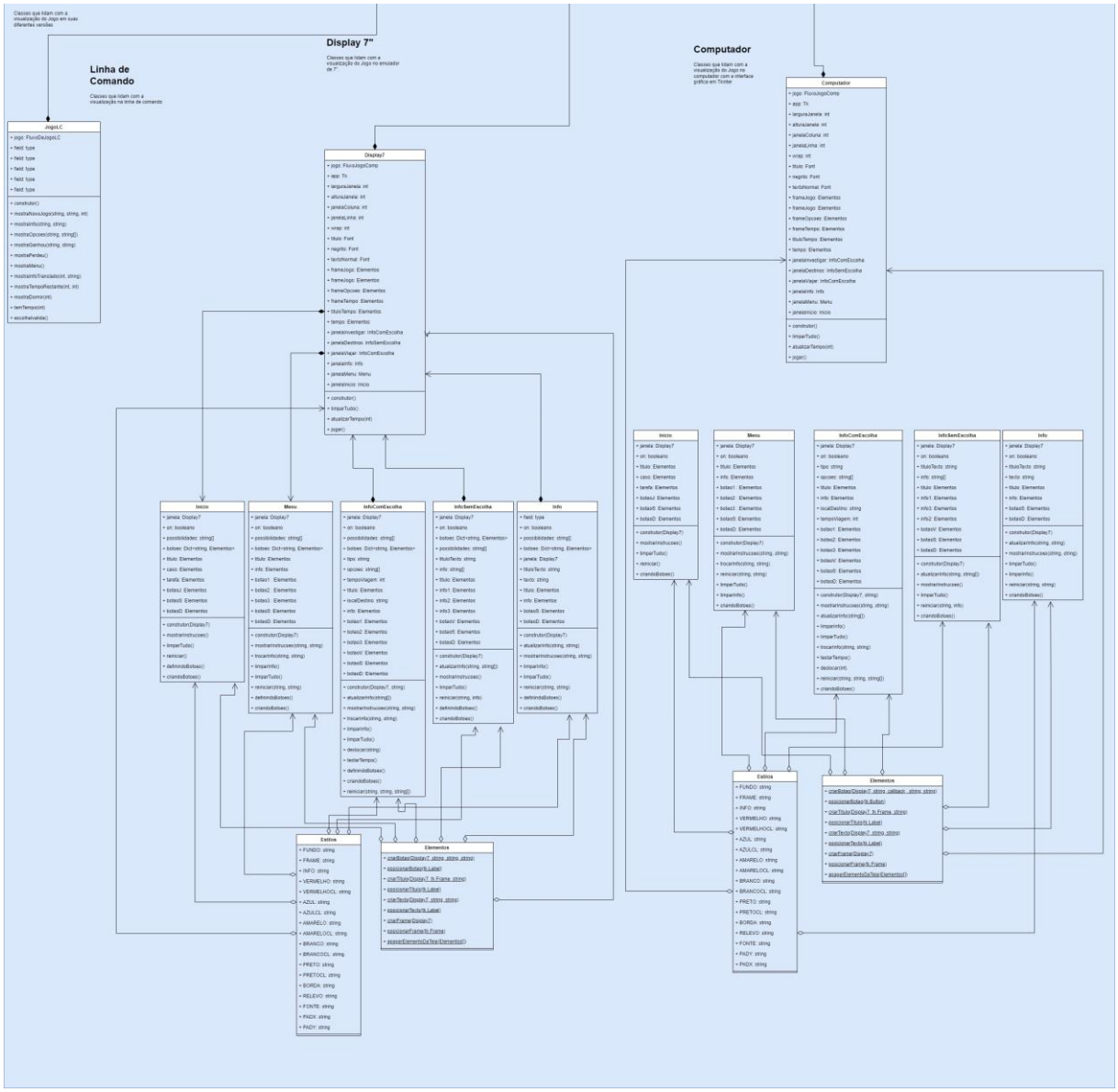
6.7 – DIAGRAMA DE CLASSES



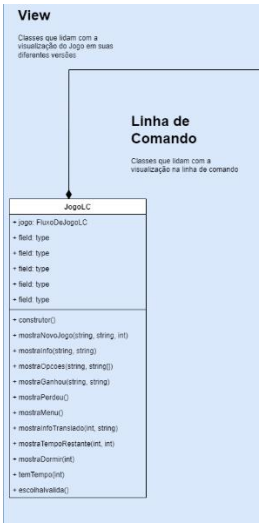
Corte Diagrama de Classes- Classes de Modelo – Controle dos dados do Jogo



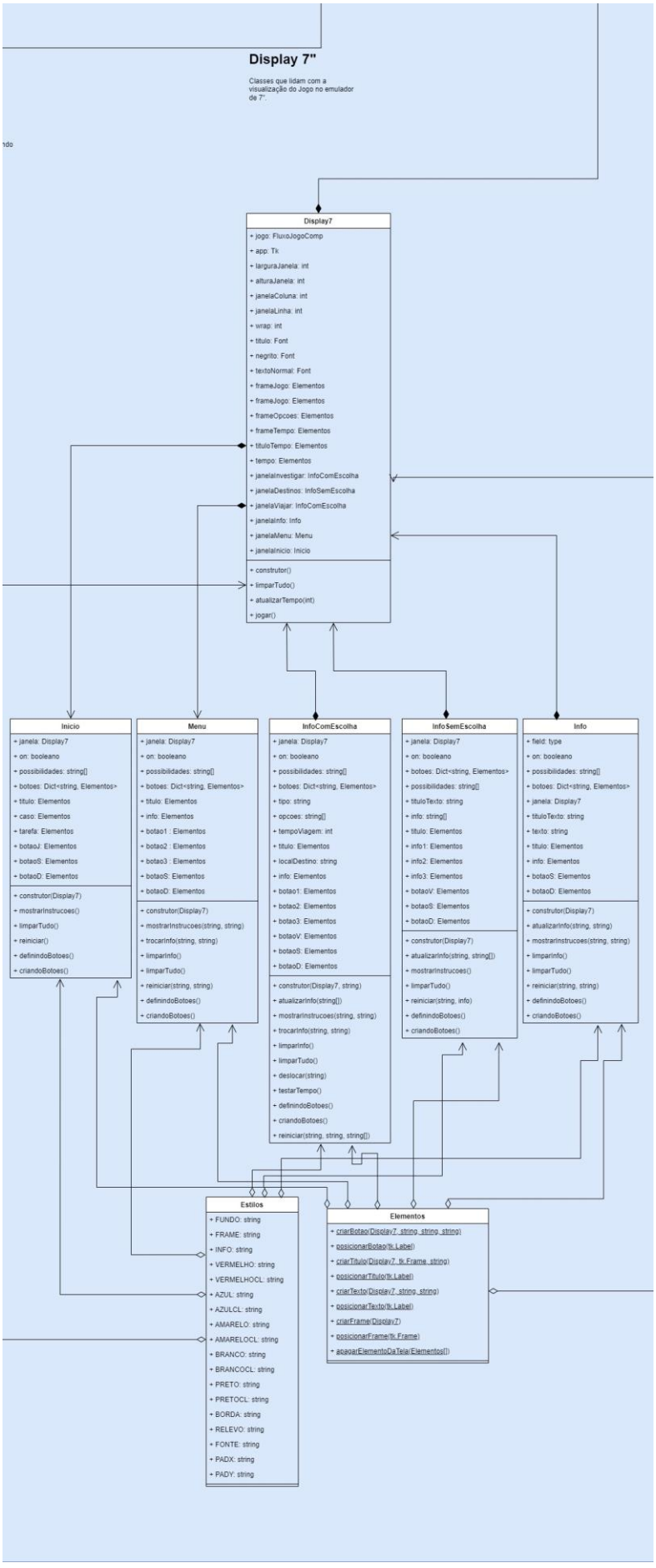
Corte Diagrama de Classes- Classes de Controle – Controle dos diferentes fluxos do Jogo



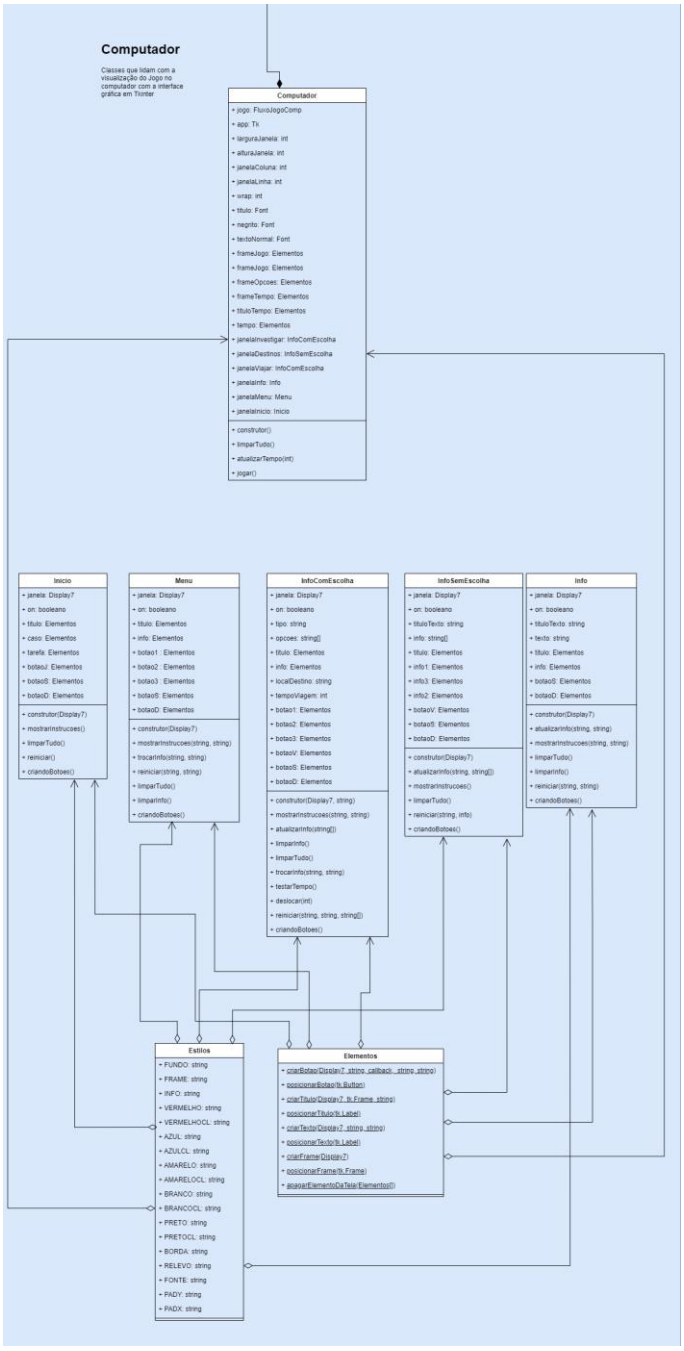
Corte Diagrama de Classes- Classes de Visual – Controle das diferentes visualizações Jogo



Corte Diagrama de Classes- Classes de Visual – Controle da visualização na Linha de Comando


Display 7"
Classes que lidam com a visualização do Jogo no emulador de 7".
Display7
+ jogo: FluxoJogoComp
+ app: Tk
+ larguraJanela: int
+ alturaJanela: int
+ janelaColuna: int
+ janelaLinha: int
+ wrap: int
+ titulo: Font
+ negrito: Font
+ textoNormal: Font
+ frameJogo: Elementos
+ frameJogo: Elementos
+ frameOpcoes: Elementos
+ frameTempo: Elementos
+ tituloTempo: Elementos
+ tempo: Elementos
+ janelaInvestigar: InfoComEscolha
+ janelaDestinos: InfoSemEscolha
+ janelaViajar: InfoComEscolha
+ janelaInfo: Info
+ janelaMenu: Menu
+ janelaInicio: Inicio
+ construtor()
+ limparTudo()
+ atualizarTempo(int)
+ jogar()
Inicio
+ janela: Display7
+ on: booleano
+ possibilidades: string[]
+ botoes: Dict<string, Elementos>
+ titulo: Elementos
+ caso: Elementos
+ tarefa: Elementos
+ botaoJ: Elementos
+ botaoS: Elementos
+ botaoD: Elementos
+ construtor(Display7)
+ mostrarInstrucoes()
+ limparTudo()
+ reiniciar()
+ definindoBotoes()
+ criandoBotoes()
Menu
+ janela: Display7
+ on: booleano
+ possibilidades: string[]
+ botoes: Dict<string, Elementos>
+ titulo: Elementos
+ info: Elementos
+ botao1 : Elementos
+ botao2 : Elementos
+ botao3 : Elementos
+ botaoS: Elementos
+ botaoD: Elementos
+ construtor(Display7)
+ mostrarInstrucoes(string, string)
+ trocarInfo(string, string)
+ limparInfo()
+ limparTudo()
+ reiniciar(string, string)
+ definindoBotoes()
+ criandoBotoes()
InfoComEscolha
+ janela: Display7
+ on: booleano
+ possibilidades: string[]
+ botoes: Dict<string, Elementos>
+ tipo: string
+ opcoes: string[]
+ tempoViagem: int
+ titulo: Elementos
+ localDestino: string
+ info: Elementos
+ botao1: Elementos
+ botao2: Elementos
+ botao3: Elementos
+ botaoV: Elementos
+ botaoS: Elementos
+ botaoD: Elementos
+ construtor(Display7, string)
+ atualizarInfo(string[])
+ mostrarInstrucoes(string, string)
+ trocarInfo(string, string)
+ limparInfo()
+ limparTudo()
+ deslocar(string)
+ testarTempo()
+ definindoBotoes()
+ criandoBotoes()
+ reiniciar(string, string, string[])
InfoSemEscolha
+ janela: Display7
+ on: booleano
+ botoes: Dict<string, Elementos>
+ possibilidades: string[]
+ tituloTexto: string
+ info: string[]
+ titulo: Elementos
+ info1: Elementos
+ info2: Elementos
+ info3: Elementos
+ botaoV: Elementos
+ botaoS: Elementos
+ botaoD: Elementos
+ construtor(Display7)
+ atualizarInfo(string, string[])
+ mostrarInstrucoes()
+ limparTudo()
+ reiniciar(string, info)
+ definindoBotoes()
+ criandoBotoes()
Info
+ field: type
+ on: booleano
+ possibilidades: string[]
+ botoes: Dict<string, Elementos>
+ janela: Display7
+ tituloTexto: string
+ texto: string
+ titulo: Elementos
+ info: Elementos
+ botaoS: Elementos
+ botaoD: Elementos
+ construtor(Display7)
+ atualizarInfo(string, string)
+ mostrarInstrucoes(string, string)
+ limparInfo()
+ limparTudo()
+ reiniciar(string, string)
+ definindoBotoes()
+ criandoBotoes()
Estilos
+ FUNDO: string
+ FRAME: string
+ INFO: string
+ VERMELHO: string
+ VERMELHOCL: string
+ AZUL: string
+ AZULCL: string
+ AMARELO: string
+ AMARELOCL: string
+ BRANCO: string
+ BRANCOCL: string
+ PRETO: string
+ PRETOCL: string
+ BORDA: string
+ RELEVO: string
+ FONTE: string
+ PADX: string
+ PADY: string
Elementos
+ criarBotao(Display7, string, string, string)
+ posicionarBotao(tk.Label)
+ criarTitulo(Display7, tk.Frame, string)
+ posicionarTitulo(tk.Label)
+ criarTexto(Display7, string, string)
+ posicionarTexto(tk.Label)
+ criarFrame(Display7)
+ posicionarFrame(tk.Frame)
+ apagarElementoDaTela(Elementos[])

Corte Diagrama de Classes- Classes de Visual – Controle da visualização no Display de 7”



Corte Diagrama de Classes- Classes de Visual – Controle da visualização no Desktop

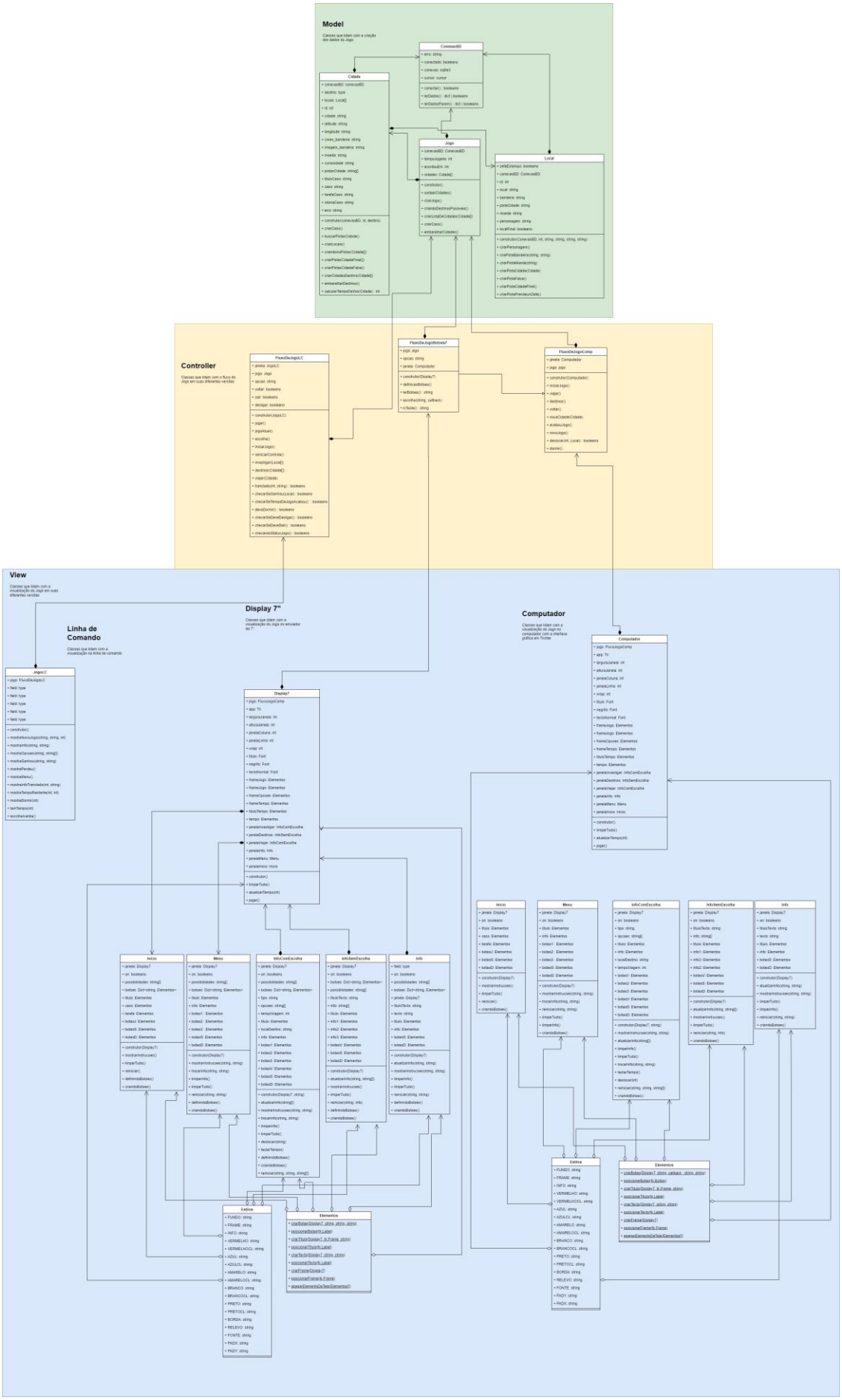

Model
Classes que lidam com a criação dos dados do Jogo
ConexaoBD
Cidade
Jogo
Local
Controller
Classes que lidam com o fluxo do Jogo em suas diferentes versões
FluxoDoJogoLC
FluxoDoJogoBotoes7
FluxoDeJogosComp
View
Classes que lidam com a visualização do Jogo em suas diferentes versões
Linha de Comando
Classes que lidam com a visualização na linha de comando
JogoLC
Display 7"
Classes que lidam com a visualização do Jogo no emulador de 7"
Computador
Classes que lidam com a visualização do Jogo no computador com a interface gráfica em Tkinter
Computador
Inicio
Menu
InfoComEscolha
InfoSemEscolha
Info
Extras
Elementos

Diagrama de Classes do Sistema Inteiro

6.8 – DIAGRAMA DE ATIVIDADES

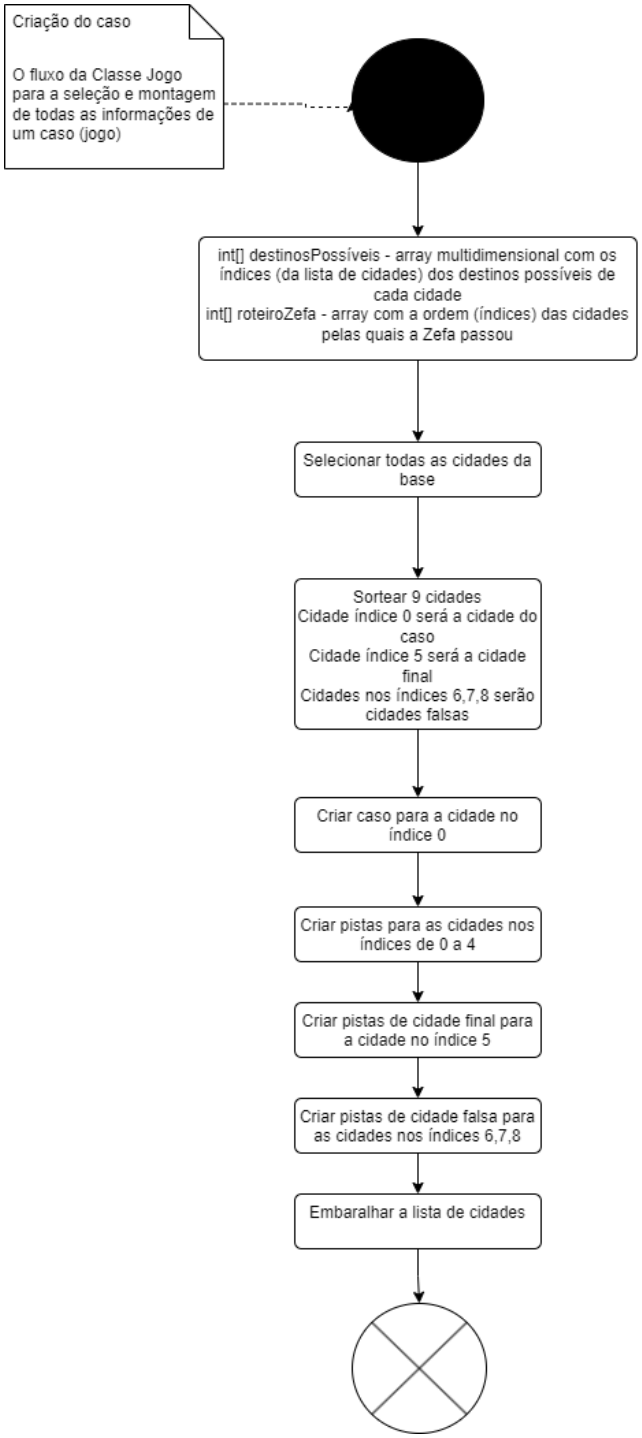


Diagrama de atividades da Classe Jogo e a criação do Caso

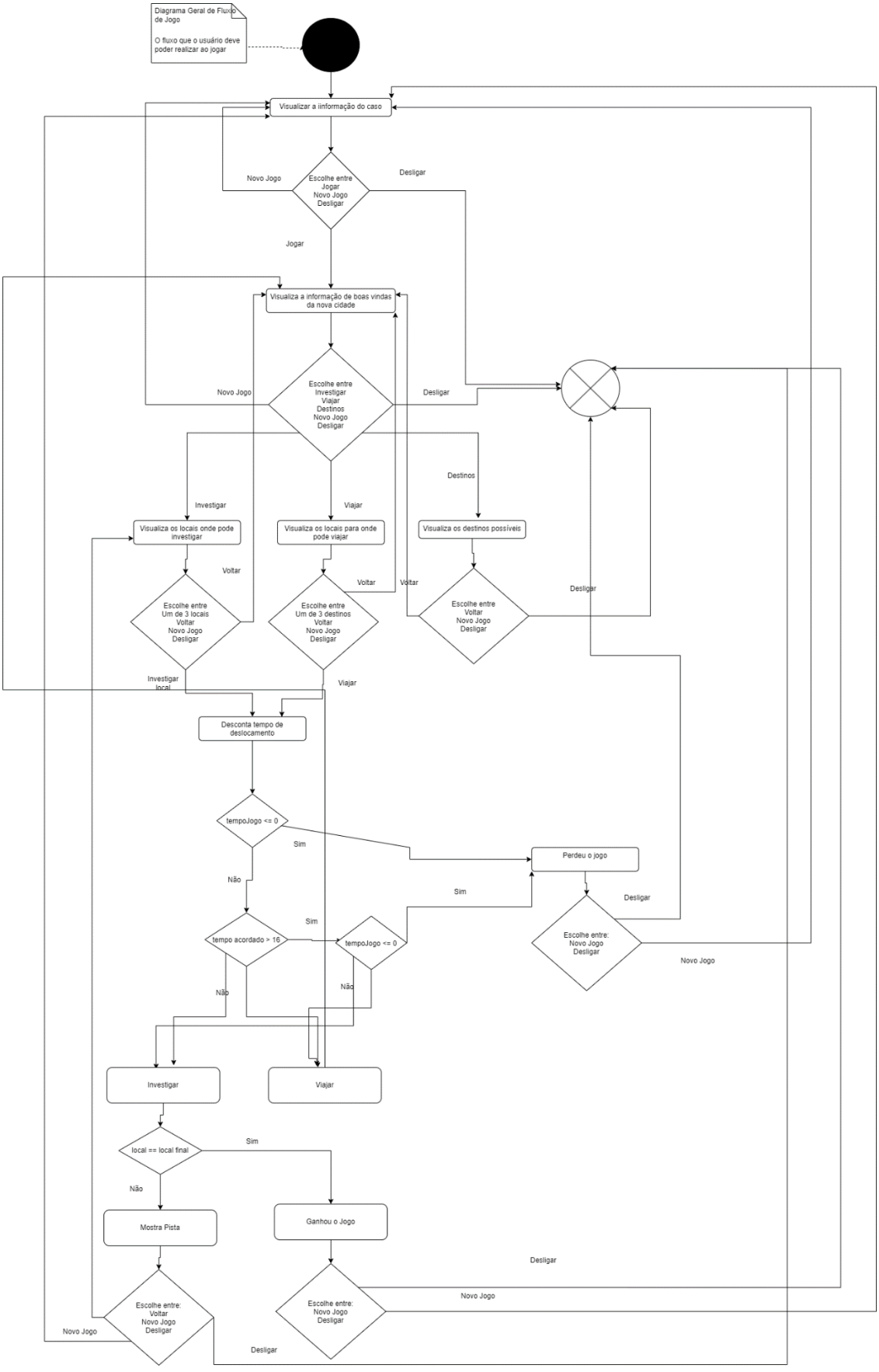


Diagrama de atividades do Fluxo de Jogo

6.9 – REQUISITOS FUNCIONAIS

- RF 01 – O sistema deve criar um caso para o jogo com nove cidades, sendo uma a cidade inicial, uma a final, e três cidades de pistas falsas de forma aleatória.
- RF 02 - O sistema deve permitir ao usuário investigar, visualizar os destinos e viajar.
- RF 03 - O sistema deve permitir ao usuário escolher uma opção de 3.
- RF 04 - O sistema deve calcular o tempo de viagem para cada escolha.
- RF 05 - O sistema deve informar ao usuário se ele capturou a Zefa ou não.
- RF 06 - O sistema deve deduzir o número de horas gastos em deslocamentos e dormir do número de horas totais disponíveis.
- RF 07 - O sistema deve permitir ao usuário se deslocar para os locais de investigação em cada cidade.
- RF 08 - O sistema deve permitir ao usuário viajar para outros países.
- RF 09 - O sistema deve permitir ao usuário visualizar os destinos possíveis.
- RF 10 - O sistema deve permitir ao usuário capturar a Zefa.
- RF 11 - O sistema deve permitir ao usuário voltar ao menu anterior.
- RF 12 – O sistema deve permitir ao usuário gerar um novo jogo.

6.10 – REQUISITOS NÃO FUNCIONAIS

- RNF 01 - O sistema deve ser capaz de rodar em um Raspberry Pi 3B+.
- RNF 02 - O sistema deve ser criado em Python.
- RNF 03 - O sistema deve ter no máximo seis botões.
- RFN 04 – O sistema deve ser criado em três versões de possibilidade de entradas e saídas: Linha de Comando, Display 7” com seis botões, Desktop com interface gráfica
- RFN 05 - O sistema deve permitir escolhas simplesmente com os seis botões.

7 – CONCLUSÃO

Neste projeto nos propusemos a buscar as respostas para a nossa ideia de construção de um projeto open source, em português, de aprendizagem de eletrônica e programação, usando um emulador de um jogo educacional com Arduino ou Raspberry Pi para ser usado em escolas.

Tínhamos as seguintes perguntas principais a responder:

1. Seria benéfico para o ensino de crianças e adolescentes aplicar um projeto do tipo em escolas?
2. Seria benéfico para o ensino de crianças e adolescentes usar jogos educacionais nas escolas?
3. As crianças se interessariam por um projeto do tipo?
4. É possível criar um projeto de baixo ou nenhum custo?
5. E por fim decidir a estrutura que este projeto, que será criado no semestre que vem na disciplina de Projeto de final de curso, tomaria.

Com a revisão da literatura das inúmeras pesquisas feitas mundo afora, tendo aqui visto exemplos de pesquisa no Brasil, Turquia, Espanha entre outros, respondemos as três primeiras perguntas com um grande sim.

Sim, o ensino de projetos makers, do ensino construtivista, é de grande benefício para o desenvolvimento de raciocínio lógico das crianças. Isso é uma teoria que já vem sendo testada desde os anos 60 com grande sucesso, provando que crianças expostas a tais projetos acabam indo melhor na escola em geral.

Sim, o uso de jogos ajuda a incentivar a criatividade da criança, o que por sua vez ajuda ela a se expressar.

Sim, o uso de jogos educacionais, que trabalham com fatos do currículo, ajuda a fixar conteúdo.

E por fim, o uso de jogos ajuda a criança a desenvolver seu relacionamento com o professor, e criar nela um sentimento de confiança por este, que é importante para o relacionamento deles.

A quarta pergunta também foi respondida com um mais ou menos. É possível criar um projeto de baixo custo ou nenhum custo? Os testes revelaram que a plataforma Arduino não poderia ser usada, e, portanto, o projeto inteiro, desde sua eletrônica, não poderia ser criado em um simulador grátis online como o Tinkercad. Porém a parte de programação do jogo para desktop ainda pode ser feita sem custo adicional.

Já o projeto físico ficou com um custo variável. O projeto pode variar de R$ 600,00 a R$ 1100,00 dependendo se se usa uma Raspberry Pi original, e qual modelo, ou uma Orange Pi. Com a Raspberry Pi 3, seis botões e um display Touch de 7” fica em torno de R$ 700,00. Porém está pode ficar mais difícil de conseguir por ser mais antiga. Já uma versão com uma

placa melhor, uma Raspberry Pi 4, de 4GB, fica em torno de R$ 1100,00. Usando-se uma placa Orange Pi, o custo pode ser baixado para R$ 600,00.

Ou seja, infelizmente, há um custo mínimo para a versão que inclui a eletrônica, porém, como tomamos o cuidado de que nada fosse soldado, não o é necessário comprar um por aluno, visto que as peças podem ser reutilizadas para vários projetos, o que deixa o custo total da escola mais baixo, se não da unidade.

Além disso, a própria Raspberry Pi, pode posteriormente ser usada como um computador menos potente pela escola ou aluno. Inclusive podendo desenvolver o programa, e seguir os tutoriais diretamente na Raspberry Pi

Decidimos pela linguagem de programação Python para o jogo e o controle dos botões e displays, além de uma base de dados SQLite.

Todos os códigos foram disponibilizados, com documentação e muito comentados, em repositório no GitHub, além de vídeos tutoriais disponíveis no YouTube, com links no repositório do GitHub.

Com isso esperamos ter contribuído para a nossa comunidade e o ensino brasileiro.

REFERÊNCIAS

1 - Escola ensina robótica e criação de games e apps para crianças a partir de 6 anos. Blog Happy Code. Acessado em: 06/10/2023 Disponível em: https://happy.com.br/blog/escola-ensina-robotica-e-criacao-de-games-e-apps-para-criancas-partir-de-6-anos/

2 - Base nacional comum curricular. Ministério da Educação. Acessado em: 06/10/2023 Disponível em: http://basenacionalcomum.mec.gov.br/abase/#introducao

3 - Aslı Görgülü Arı and Gülsüm Meço . A New Application in Biology Education: Development and Implementation of Arduino-Supported STEM Activities. MDPI. 7/6/2021. Disponível em: https://www.mdpi.com/2079-7737/10/6/506

4 - De Paula, Bruna Braga. Dissertação de Mestrado: Cultura maker na educação: uma abordagem integrada ao ensino 14/02/2022/ Disponível em:

https://repositorio.unifesp.br/bitstream/handle/11600/63764/Disserta%c3%a7%c3%a3o%20Bruna%20Braga_Vers%c3%a3o%20Final_MPIT_17.03.2022_Turnitin_22.04.pdf?sequence=5&isAllowed=y

5 – Sanes, Daniel. Pesquisadora defende acessibilidade da cultura maker. Academia Rhyzos. 15/09/2022. Disponível em: https://rhyzos.com/acessibilidade-ensino-cultura-maker/

6 – Cafardo, Leonardo. Maioria das escolas estaduais de SP não tem computador e conexão suficientes: 'Wi-fi não pega'. Estadão. 01/09/2023. Disponível em: https://www.estadao.com.br/educacao/maioria-das-escolas-estaduais-de-sp-nao-tem-computador-e-conexao-suficientes-wifi-nao-pega/

7 - Candido Caroline, T.R. A importância dos jogos e brincadeiras na educação infantil. Saberes Docentes em Ação, vº 05, nº1. 01/11/2021. Disponível em: https://maceio.al.gov.br/uploads/documentos/1-A-IMPORTANCIA-DOS-JOGOS-E-BRINCADEIRAS-NA-EDUCACAO-INFANTIL-1.pdf

8 - Savi, R., Ulbricht, V. R. Jogos Digitais Educacionais: Benefícios e Desafios. Revista Novas Tecnologias na Educação, Porto Alegre, v. 6, n. 1, 2008. DOI: 10.22456/1679-1916.14405. Disponível em: https://seer.ufrgs.br/index.php/renote/article/view/14405.

9 – Seção Sobre da página da Arduino. Acessado em: 06/10/2023. Disponível em: https://www.arduino.cc/en/about

10 – Seção sobre da página da Raspberry Pi Foundation. Acessado em: 06/10/2023. Disponível em: https://www.raspberrypi.org/about/

11 - Santos, W. K.T. Exemplos de utilização do Raspberry Pi para Auxílio no ensino fundamental. Universidade Federal Rural do Semiárido. Curso de Ciência da Computação. 2018. Disponível em: https://repositorio.ufersa.edu.br/handle/prefix/5651?locale=pt_BR

12 – Barrios, William. 23 Educational Games From The '90s That Actually Go Hard. Buzz Feed. 09/03/2023. Disponível em: https://www.buzzfeed.com/williambarrios/90s-learning-computer-games

13 – The Oregon Trail (series) Wikipedia. Acessado em: 05/10/2023. Disponível em: https://en.wikipedia.org/wiki/The_Oregon_Trail_(series)

14 - Where in the World Is Carmen Sandiego? (1985 video game). Wikipedia. Acessado em: 06/10/2023. Disponível em:

https://en.wikipedia.org/wiki/Where_in_the_World_Is_Carmen_Sandiego%3F_(1985_video_game)

15 – Licença GNU GPL. Acessado em: 06/10/2023. Disponível em: https://www.gnu.org/licenses/gpl-3.0.en.html

16 – Kummarikuntla, Teja. Choosing the Right Python GUI Framework: A Complete Guide. Tooljet. 26/10/2023. Disponível em: https://blog.tooljet.com/python-gui-framework/#:~:text=Some%20popular%20options%20include%20Tkinter,appealing%20GUI%20applications%20using%20Python.

17 – Cerdeira et. al. Jogo em Raspberry Pi para o ensino fundamental e médio: Desenvolvimento do projeto educacional de eletrônica e programação. Universidade Anhembi Morumbi. 10/2023

18 – Site retrogames, emulador de jogos antigos. Acessado em: Outubro 2023. Disponível em: https://www.retrogames.cz/play_1305-DOS.php

TABELAS COMPARATIVA DE PREÇOS ENCONTRADOS

Raspberry Pi 4 4GB	**Banana Pi**	Orange Pi
https://www.robocore.net/placa-raspberry-pi/raspberry-pi-3-model-b-plus	https://produto.mercadolivre.com.br/MLB-3782248444-placa-banana-pi-m1-bpi-m1-_JM#position=2&search_layout=grid&type=item&tracking_id=a818485c-b884-4599-887c-7ed6ab955dca	https://produto.mercadolivre.com.br/MLB-3857304598-orange-pi-zero-3-4gb-ram-wi-fi-bluetooth-allwinner-h618-_JM#polycard_client=recommendations_home_navigation-recommendations&reco_backend=machinalis-homes-pdp-boos&reco_client=home_navigation-recommendations&reco_item_pos=2&reco_backend_type=function&reco_id=4e7fb744-ef76-48a5-8256-ef5cce561ee1
R$ 699,00	R$ 290,00	

		R$ 298,00
Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	CPU: A20 ARM Cortex-A7 Dual-Core (option:allwinner A40i) GPU: ARM Mali 400 MP2	Rockchip RK3566 quad-core 64-bit processor with 22nm advanced process, up to 1.8GHz
4GB LPDDR4-3200 SDRAM	1 GB DDR3	2 GB

Displays	**2,8" TFT Shield**	Display 7" SHCHV HDMI
Informação	Aliexpress	Ali Express
Valor	50,11	R$ 197,45

Cartões de memória micro SD	**32GB**	**64GB**	**128GB**
Informação	Amazon	Amazon	Amazon
Valor	26,99	39,5	59,75

Custo Projeto Final		
	Mais barato	**Mais Caro**
Item	**Valor**	**Valor**
OrangePi	290	
Display 7"	197,45	197,45
Cartão SD Card 32	26,99	

Raspberry Pi		699
Cartão SD Card 128		59,75
Kit de cabos, botões, resistores, leds e protoboard	85	85
Total	599,42	1041,20

CÓDIGO DO JOGO

Pacote Controller:

Classe FluxoDeJogoBotoes7:

```
from Controller.FluxoDeJogoComp import FluxoDeJogoComp
# import RPi.GPIO as pi
class FluxoDeJogoBotoes7(FluxoDeJogoComp):
    """
    Esta classe que controla o fluxo do jogo
    """

    def __init__(self, janela):
        """
        Inicia o jogo
        """
        self.definicaoBotoes() # Preciso definir os botões antes de rodar o super, pois o construtor da classe mãe roda enquanto o botão desligar não tiver sido apertado
        # self.janela = janela
        # self.jogo = Jogo()
        super().__init__(janela)

    def definicaoBotoes(self):
        # pi.setmode(pi.BOARD)  # Configurando o tipo de leitura dos Pinos. Neste caso usando os números do GPIO
        # pi.setwarnings(False) # Desabilita avisos
        # Definindo em que pino está cada um - repare como as minhas chaves são exatamente o mesmo que eu espero na Linha de Comando. Desse modo só vou precisar alterar o método de entrada. Mas não o valor da entrada em si
        self.botoes = {
            "1": 40,
            "2": 38,
            "3": 36,
            "v": 32,
            "s": 37,
            "d": 35,

        }
```

```
        # Configurando os pinos como entrada com Pull-up
        # for botao in self.botoes.values():
        #     pi.setup(botao, pi.IN, pi.PUD_UP)




    def lerBotoes(self):
        """
        Lê um botão
        :param botao: o número de pino do botão
        :return:
        """
        while True:
            # Fazendo um loop pelos botões. Uso o número do pino para ler e retorno a chave
            for botao, gpio in self.botoes.items():
                if pi.input(gpio) == 0:
                    print(f"Apertei botao {botao}")
                    return botao

    def invocandoBotao(self, botao):
        """
        Esta função vai incovar o botão desejado
        """
        botao.invoke() # vai chamar a ação do botão
    def escolha(self, possibilidades, jbotoes):
        """
        Checa se a escolha feita está dentro das possibilidades, se não continua esperando
resposta
        :param: possibilidades: array com as possibilidades de botões
        :param: jbotoes: o dicionário de botões da janela
        :return:
        """
        print("Possibilidades")
        print(possibilidades)
        valido = False
        while valido != True:
            # Está é a única linha de código diferente nesta função. O método pelo qual leio a
minha escolha
            # self.opcao = self.lerBotoes()
            self.opcao = self.lcTeste()
            print(f"Escolha {self.opcao}")
            valido = self.opcao in possibilidades
            if valido == False:
                print("Escolha inválida, favor escolher outra opção")
            else:
                self.invocandoBotao(jbotoes[self.opcao])
```

```
    def lcTeste(self):
        """
        Função para testes da linha de comando
        :return:
        """
        print("Digite a sua escolha:")
        return input()
```

Classe FluxoDeJogoComp:

```
from Model.Jogo import Jogo
class FluxoDeJogoComp:
    """
    Esta classe que controla o fluxo do jogo
    """

    def __init__(self, janela):
        """
        Inicia o jogo
        """
        self.janela = janela
        self.jogo = Jogo()



    def iniciarJogo(self):
        # Apaga a variável jogo anterior dando o valor None, ou Nulo para
        #self.jogo = 0
        self.jogo = Jogo()
        # Imprime a informação de um jogo novo
        self.janela.janelaInicio.reiniciar()




    def investigar(self):
        self.janela.limparTudo()
        opcoes = []
        for opcao in self.jogo.cidadeAtual.locais:
            opcoes.append(opcao.local)
        self.janela.janelaInvestigar.reiniciar(f"Clique em uma das opções abaixo para começar a
investigar na cidade de {self.jogo.cidadeAtual.cidade}", "Não se esqueça, deslocamentos
dentro da cidade levam 1 hora!", opcoes)

        print("Investigar")
    def viajar(self):
        print("Viajar")
        self.janela.limparTudo()
        opcoes = []
        tempoDeVoo = ""
```

```
        for opcao in self.jogo.cidadeAtual.cidadesDeDestino:
            tempoDeVoo += f"\n{opcao.cidade}:
{self.jogo.cidadeAtual.calcularTempoDeVoo(opcao)} horas de voo."

            opcoes.append(opcao.cidade)
        self.janela.janelaViajar.reiniciar("Para onde quer viajar? ", tempoDeVoo, opcoes)
        print("Destinos")
    def destinos(self):
        self.janela.limparTudo()
        opcoes = []
        for opcao in self.jogo.cidadeAtual.cidadesDeDestino:
            opcoes.append(opcao.cidade)
        self.janela.janelaDestinos.reiniciar("Você pode viajar para: ", opcoes)
        print("Destinos")

    def voltar(self):
        print("Voltar")
        self.janela.limparTudo()
        self.janela.janelaMenu.reiniciar(self.jogo.cidadeAtual.cidade, "Qual seu próximo passo?
Clique em um dos botões abaixo para escolher.")

    def novaCidade(self, cidade):
        self.jogo.cidadeAtual = cidade
        print("Nova cidade")
        self.janela.limparTudo()
        self.janela.janelaMenu.reiniciar(self.jogo.cidadeAtual.cidade,
self.jogo.cidadeAtual.curiosidade)

    def acabouJogo(self):
        print("Acabou o jogo")
        self.janela.limparTudo()
        self.janela.janelaInfo.reiniciar("Que pena! Acabou seu tempo!", "Mas não desanime,
você ainda terá outras oportunidades de capturar a Zefa!")
    def ganhou(self):
        print("Ganhou")
        self.janela.limparTudo()
        self.janela.janelaInfo.reiniciar("Eba! Você capturou a Zefa com sucesso!",
self.jogo.cidadeInicial.vitoriaCaso)
    def novoJogo(self):
        self.janela.limparTudo()
        self.iniciarJogo()
        self.janela.atualizarTempo("")
        print("Novo jogo")


    def deslocar(self, tempoViagem, local):
        """
        Se desloca e checa se o jogo pode continuar
        :param tempoViagem: tempo gasto no deslocamento
        :return: False se acabou o tempo de jogo True se pode continuar
```

```
        """

        self.jogo.tempoJogado -= tempoViagem
        self.infoTempo = f"Tempo depois do deslocamento para {local}"
        self.janela.atualizarTempo(f"{self.infoTempo}:")
        if self.jogo.tempoJogado <= 0:
            return False
        return True
    def dormir(self):
        """
        Verifica se tem que dormir. Se sim, desconta o tempo e checa se acabou o tempo
        :return: 0 - Não precisou dormir
        :return: 1 - Precisou dormir e acabou o tempo (deve mostrar dormir e depois a
mensagem de que acabou o tempo
        :return: 2 -Precisou dormir e não acabou o tempo deve mostrar a mensagem de dormir
e voltar ao fluxo de jogo
        """

        if self.jogo.acordouEm - self.jogo.tempoJogado >= 16:
            self.infoTempo += " e de dormir"
            self.jogo.tempoJogado -= 8
            self.jogo.acordouEm = self.jogo.tempoJogado
            self.janela.atualizarTempo(f"{self.infoTempo}:")
            if self.jogo.tempoJogado <= 0:
                return 1
            return 2
        return 0
```

Classe FluxoDeJogoBotoes7:

```
from Model.Jogo import Jogo


class FluxoDeJogoLC:
    """
    Esta classe que controla o fluxo do jogo
    """

    def __init__(self, janela):
        """
        Inicia o jogo e fica rodando, com novos jogos, enquanto não desligar
        """
        # Recebe o objeto view com funções que vão  servir para mostrar minhas informações.
Neste caso, todos os meus objetos view vão implementar estas funções
        self.janela = janela
```

```
    def jogar(self):
        """
        Começa o jogo.
        :return:
        """

        # Variável de controle para saber se devo desligar o programa
        self.desligar = False
        # Variáveis de controle do jogo
        self.reiniciarControle()
        # Loop que mantem o programa rodando até o usuário escolher desligar
        while self.desligar == False:
            self.jogoAtual()

    def jogoAtual(self):
        """
        Controla o jogo atual
        :return:
        """
        # Inicia o jogo
        self.iniciarJogo()
        # Laço que manterá o jogo acontecendo até que alguém escolha desligar ou sair.
        # Desligar para de rodar o programa
        # Sair, reinicia um jogo novo
        while self.sair != True:
            # Checa se o jogo deve continuar, se não para a função com o return, para retornar
ao loop inicial
            if self.checandoStatusJogo():
                return

            # Imprime o menu
            self.janela.mostraMenu()
            self.escolha()
            # Testa a opcão escolhida para escolher a ação a ser tomada
            if self.opcao == "1":
                self.investigar(self.jogo.cidadeAtual.locais)
            elif self.opcao == "2":
                self.viajar(self.jogo.cidadeAtual.cidadesDeDestino)
            elif self.opcao == "3":
                self.destinos(self.jogo.cidadeAtual.cidadesDeDestino)
            elif self.opcao == "v":
                print("Voltar")
                self.voltar = True
                continue
            elif self.opcao == "s":
                print("Sair")
                self.sair = True


    def escolha(self):
```

```
        """
        Checa se a escolha feita está dentro das possibilidades, se não continua esperando
resposta
        :return:
        """
        possibilidades = ["1", "2", "3", "s", "d", "v"]
        valido = False
        while valido != True:
            self.opcao = input()
            valido = self.opcao in possibilidades
            if valido == False:
                self.janela.escolhaIvalida()
    def iniciarJogo(self):
        # Apaga a variável jogo anterior dando o valor None, ou Nulo para
        #self.jogo = 0
        self.jogo = Jogo()
        # Inicia as variáveis de controle
        self.reiniciarControle()
        # Imprime a informação de um jogo novo
        self.janela.mostraNovoJogo(self.jogo.cidadeInicial.caso,
self.jogo.cidadeInicial.tarefaCaso, self.jogo.tempoJogado)
        self.janela.mostraInfo(f"Boas vindas a {self.jogo.cidadeAtual.cidade}",
self.jogo.cidadeAtual.curiosidade)



    def reiniciarControle(self):
        """
        Reinicia as variáveis de controle depois de um loop de jogo
        :return:
        """
        # Propriedade de controle
        self.sair = False
        # Propriedade de controle
        self.voltar = False
        # Propriedade que vai receber aopção selecionada
        self.opcao = 0

    def investigar(self, locais):
        """
        Função que roda quando o botão investigar é escolhido
        :param locais:
        :return:
        """
        if self.checandoStatusJogo():
            return

        opcoes = []
        for local in locais:
            opcoes.append(local.local)
```

```
        self.janela.mostraOpcoes("Escolha o local que deseja investigar digitando a opção de
destino ou v para voltar ou d para desligar.", opcoes)
        self.escolha()
        if self.checandoStatusJogo():
            return
        if self.opcao == "v":
            return
        if self.translado(1, locais[int(self.opcao) - 1].local):
            return

        self.janela.mostraInfo(locais[int(self.opcao) - 1].local, locais[int(self.opcao) - 1].pista)
        # Checa se ganhou o jogo e toma as ações necessárias
        self.checarSeGanhou(locais[int(self.opcao) - 1])


    def destinos(self, destinos):
        """
        Mostra os destinos para onde poderia viajar
        :param destinos:
        :return:
        """
        if self.checandoStatusJogo():
            return
        opcoes = []
        for destino in destinos:
            opcoes.append(destino.cidade)
        self.janela.mostraOpcoes("Estes são os destinos para os quais você pode viajar:", opcoes)

    def viajar(self, destinos):
        """
        Mostra os destinos para onde poderia viajar
        :param destinos: as cidades para onde o jogador pode viajar
        :return:
        """
        if self.checandoStatusJogo():
            return
        opcoes = []
        for destino in destinos:
            opcoes.append(destino.cidade)
        self.janela.mostraOpcoes("Para onde você quer viajar?\nSe precisar, digite v para
voltar.", opcoes)
        self.escolha()
        if self.checandoStatusJogo():
            return
        if self.opcao == "v":
            return
        # Faz o translado, e faz todas as checagens de tempo. Caso tenha estourado o tempo de
jogo retorna
        # Calcula o tempo de voo
```

```
        tempoVoo = self.jogo.cidadeAtual.calcularTempoDeVoo(destinos[int(self.opcao) - 1])
        if self.translado(tempoVoo, destinos[int(self.opcao) - 1].cidade):
            return
        self.jogo.cidadeAtual = destinos[int(self.opcao) - 1]
        self.janela.mostraInfo(f"Boas vindas a {self.jogo.cidadeAtual.cidade}",
self.jogo.cidadeAtual.curiosidade)

    def translado(self, tempo, destino):
        """
        Desconta tempo de translado
        :param tempo: tempo de qualquer viagem
        :param destino: para onde está se deslocando
        :return: Boolean: True caso tenha ultrapassado o tempo de jogo | False caso ainda
tenha tempo de jogo
        """
        self.janela.mostraInfoTranslado(tempo, destino)
        self.jogo.tempoJogado -= tempo
        self.janela.mostraTempoRestante(tempo, self.jogo.tempoJogado)
        if self.checarSeTempoDeJogoAcabou():
            return True
        if self.devoDormir():
            return True
        return False

    def checarSeGanhou(self, local):
        """
        Verifica se a pessoa chegou ao local onde a zefa está e ganhou o jogo e toma as ações
necessárias
        :param local: local de investigação
        :return:
        """
        if local.zefaEstaAqui:
            self.sair = True
            self.janela.mostraGanhou(self.jogo.cidadeInicial.vitoriaCaso, self.jogo.tempoJogado)
            return True
        return False
    def checarSeTempoDeJogoAcabou(self):
        """
        Checa se já acabou o tempo de jogo
        :return: Booleno True se acabou o tempo | False se ainda tem tempo
        """
        if self.jogo.tempoJogado <= 0:
            self.janela.mostraPerdeu()
            # self.imprimirTitulo("Que pena. Acabou o tempo que você tinha para achar a Zefa!
Mas não desista! Na próxima você acha ela!")
            self.sair = True
            return True
        else:
            return False
```

```
    def devoDormir(self):
        """
        Checa se o jogador deve dormir, se sim desconta as horas e verifica se ainda tem tempo de jogo ou não.
        :return: Boolean True se não tem mais tempo de Jogo | False se tem
        """
        if (self.jogo.acordouEm - self.jogo.tempoJogado) >= 16:

            self.janela.mostraDormir(self.jogo.tempoJogado)
            self.jogo.tempoJogado -= 8
            self.jogo.acordouEm = self.jogo.tempoJogado
            if self.checarSeTempoDeJogoAcabou():
                return True
            self.janela.temTempo(self.jogo.tempoJogado)
            # self.imprimirTexto(f"Ufa, você ainda tem {self.jogo.tempoJogado} horas. Corre lá.")
            return False

    def checarSeDeveDesligar(self):
        """
        Função que checa se deve desligar. Ela é usada sempre que temos um menu de escolha no início
        :return: True
        """
        if self.opcao == "d":
            self.desligar = True
            return True
        else:
            return False
    def checarSeDeveSair(self):
        """
        Função que checa se deve sair do jogo e iniciar um novo. Ela é usada sempre que temos um menu de escolha no início
        :return: True
        """
        if self.opcao == "s" or self.sair == True:
            self.sair = True
            return True
        else:
            return False

    def checandoStatusJogo(self):
        """
        Função que checa se o jogo deve continuar ou reiniciar ou desligar,
        Usada sempre que há um menu de escolha
        :return: Boolean
        """
        if self.checarSeDeveDesligar() or self.checarSeDeveSair():
            return True
        else:
```

```
            return False
```

Pacote Model:

Classe ConexaoBD:

```
import sqlite3 # import da biblioteca para lidar com SQLite
import traceback # é uma boa biblioteca para fazer debugging

class ConexaoBD:
    """
    Está classe faz a conexão com a base de dados
    Atributos:
    ConexaoSQLite: conexão com o SQLite
    cursor: cursor para executar as buscas (queries)
    conectad: booleano que indica se há uma conexão
    """
    erro = ""
    def __init__(self):
        self.conectado = self.conectar()

    def conectar(self):
        """
        Faz a conexão com a base de dados
        :return: Booleano de sucesso (True) ou erro (False)
        """
        try:
            self.conexao = sqlite3.connect('db/zefa.sqlite')
            self.conexao.row_factory = sqlite3.Row
            self.cursor = self.conexao.cursor()
            print("A conexão com a base de dados foi um sucesso")
            return True

        except sqlite3.Error as erro:
            self.erro = erro
            print("Erro ao conectar ao sqlite", self.erro)
            return False



    def lerDados(self, sql):
        """
        Usada para ler dados quando o sql não usar variáveis
        """
        if(self.conectado):
            try:
                self.cursor.execute(sql)
```

```
            self.registros = self.cursor.fetchall()
            print("Os dados da base são: ")
            for registro in self.registros:
                for chave in registro.keys():
                    print(f"Coluna: {chave} | Valor: {registro[chave]}")
            return self.registros
        except sqlite3.Error as erro:
            self.erro = erro
            print("Erro ao conectar ao sqlite", self.erro)
            return self.erro
    else:
        print(f"A conexão não existe: {self.erro}")
        return False


def lerDadosParam(self, sqlParametrizado, parametros):
    """
    Com os atributos cursor faremos a busca (query) na base de dados

    :param: sqlParametrizado: sql já com os placeholders para os paramêtros. Ex: "select * from tabela where id = :id"
    :param: parametros: dict com os parâmetros, sendo as chaves os nomes dos paramêtros. Exemplo: {"id": 2}
    Mais informação: https://docs.python.org/3/library/sqlite3.html#sqlite3-placeholders
    :return: resultados ou erro
    """
    if(self.conectado):
        try:
            self.cursor.execute(sqlParametrizado, parametros)
            self.registros = self.cursor.fetchall()
            print("Os dados da base são: ")
            for registro in self.registros:
                for chave in registro.keys():
                    print(f"Coluna: {chave} | Valor: {registro[chave]}")
            return self.registros
        except sqlite3.Error as erro:
            self.erro = erro
            print("Erro ao conectar ao sqlite", self.erro)
            return self.erro
    else:
        print(f"A conexão não existe: {self.erro}")
        return False
```

Classe Cidade:

```
import random
import haversine as hs

from Model.Local import Local
```

```
class Cidade:
    def __init__(self, conexaoBD, id_cidade, destino):
        """
        Inicia a classe cidade
        :param: conexãoBD: o objeto de conexão à base de dados
        :param: id: o id da Cidade buscada
        :param: destino: o indíce no array cidades da cidade de destino
        """
        self.destino = destino
        self.conexaoBD = conexaoBD

        self.locais = []
        self.localAtual = ""

        sql = f"select * from cidades as c INNER JOIN paises as p ON (c.fk_id_pais = p.pk_id_pais) where c.pk_id_cidade =  :id"
        parametros = {"id": id_cidade}
        resultado = self.conexaoBD.lerDadosParam(sql, parametros)


        if len(resultado) >= 1:

            self.id = resultado[0]["pk_id_cidade"]
            self.cidade = resultado[0]["cidade"]
            self.latitude = resultado[0]["latitude"]
            self.longitude = resultado[0]["longitude"]
            self.cores_bandeira = resultado[0]["cores_bandeira"]
            self.imagem_bandeira = resultado[0]["imagem_bandeira"]

            if self.imagem_bandeira !=  None:
                # self.imagem_bandeira == False
                print("Imagem bandeira não era None")
            self.moeda = resultado[0]["moeda"]

            sql = f"select * from curiosidades_cidades where fk_id_cidade = :id"
            parametros = {"id": self.id}
            curiosidades = conexaoBD.lerDadosParam(sql, parametros)
            # print(curiosidades)
            quantidadeDeCuriosidades = len(curiosidades)

            if quantidadeDeCuriosidades > 1:
                indice = random.randint(0, quantidadeDeCuriosidades - 1)
                self.curiosidade = curiosidades[indice]["curiosidade"] # Outra opção aqui é usar a função choice da biblioteca random que retorna um elemento de uma sequência não vazia de modo aleatório. Se quiser saber mais dê uma olhada na documentação: https://docs.python.org/3/library/random.html#random.choice
            else:
                self.curiosidade = curiosidades[0]["curiosidade"]
            #Criando os locais desta cidade - as pistas serão atribuidas a estes locais na classe Jogo
```

```
        self.criarLocais()
        #Aqui estou buscando a pista da cidade para para depois atribuir onde esta cidade
for destino
        self.buscarPistasCidade()

    def criarCaso(self):
        """
        Cria o caso para a cidade
        :return: informação de caso atribuida às propriedades da cidade
        """
        sql = f"select * from casos where fk_id_cidade = :id"
        parametros = {"id": self.id}
        casos = self.conexaoBD.lerDadosParam(sql, parametros)

        quantidadeDeCasos = len(casos)
        indice = 0
        if quantidadeDeCasos > 0:
            if quantidadeDeCasos > 1:
                indice = random.randint(0, quantidadeDeCasos - 1)
            self.tituloCaso = casos[indice]["titulo"]
            self.caso = casos[indice]["caso"]
            self.tarefaCaso = casos[indice]["tarefa"]
            self.vitoriaCaso = casos[indice]["mensagem_de_vitoria"]

        else:
            self.erro = "Não há um caso com esse id de cidade"

    def buscarPistasCidade(self):
        """
        Buscando as pistas que dizem respeito a esta cidade, para serem usadas quando a
cidade for destino
        :return: lista de pistas da cidade
        """
        self.pistasCidade = []
        sql = f"select pista from pista_da_cidade where fk_id_cidade = :id"
        parametros = {"id": self.id}
        pistas = self.conexaoBD.lerDadosParam(sql,parametros)
        print("Pistas da base")
        print(pistas)
        for pista in pistas:
            self.pistasCidade.append(pista["pista"])
        print("As pistas da cidade são:")
        print(self.pistasCidade)
    def criarLocais(self):
        """
        Criando os locais da cidade
        :return: a atribuição dos locais a propriedade locais
        """
        sql = "select * from locais"
        locaisBase = self.conexaoBD.lerDados(sql)
```

```
        # print("Locais Base")
        # print(locaisBase)
        if len(locaisBase) > 0:
            for indice in range(0,3):
                indice = random.randint(0, len(locaisBase) - 1)
                print(f"O local que vai ser colocado")
                print(locaisBase[indice])
                self.locais.append(Local(self.conexaoBD,
locaisBase[indice]["pk_id_local"],locaisBase[indice]["local"], locaisBase[indice]["bandeira"],
locaisBase[indice]["pista_cidade"], locaisBase[indice]["moeda"]))
                locaisBase.pop(indice)
            print("Os locais")

            print(len(self.locais))
        else:
            self.erro = "Não há locais na base"
            print(self.erro)
        print(f"Os locais escolhidos para a cidade {self.cidade}:")
        for local in self.locais:
            print(local.local)


    def criandoAsPistas(self, destinos):
        """
        Atribuindo as pistas aos locais
        :param destinos: o array com todas as cidades do caso para poder verificar a cidade
para a qual a Zefa partiu desta cidade, e a qual as pistas dirão respeito a está cidade
        :return: atribuição das pistas a propriedade pista de cada Local
        """
        #Variáveis de controle
        coresBandeiraUsada = False
        imagemBandeiraUsada = False
        pistasBandeiras = 0
        moedaUsada = False
        print(f"Criando as pistas para {self.cidade}")
        print("Locais da cidade onde pistas vão ser criadas")
        for loc in self.locais:
            print(f"{loc.local} com {loc.personagem}")
        for local in self.locais:
            print(f"Pegando pista para: {local.local} com id: {local.id} Bandeira: {local.bandeira}
Moeda: {local.moeda} Personagem: {local.personagem}")
            if local.bandeira == 1:
                print(f"Imagem bandeira: {destinos[self.destino].imagem_bandeira}")
                if destinos[self.destino].imagem_bandeira !=  None:
                    print(f"Entrou na imagem bandeira")
                    if imagemBandeiraUsada == False:
                        if (random.randint(0,1) == 0 and pistasBandeiras == 0) or pistasBandeiras ==
1:
                            local.criaPistaBandeira(destinos[self.destino].imagem_bandeira, True)
                            imagemBandeiraUsada = True
```

```
                    pistasBandeiras += 1
                    continue
                if coresBandeiraUsada == False:
                    local.criaPistaBandeira(destinos[self.destino].cores_bandeira, False)
                    coresBandeiraUsada = True
                    pistasBandeiras += 1
                    continue

            if local.moeda == 1 and moedaUsada == False:
                local.criarPistaMoeda(destinos[self.destino].moeda)
                moedaUsada = True
                continue

            print("Pistas antes do pop",destinos[self.destino].pistasCidade)
            local.criarPistaCidade(destinos[self.destino].pistasCidade)
            print("Pistas depois do pop",destinos[self.destino].pistasCidade)
        print("Resultado montagem locais")
        for newLocal in self.locais:
            print(f"A pista para o local: {newLocal.local} da cidade {self.cidade} é:
{newLocal.pista}")


    def criarPistasCidadeFinal(self):
        """
        Caso está seja a cidade final onde a Zefa se encontra, atribuindo as pistas a propriedade
pista dos locais
        :return: atribuição das pistas a propriedade pista de cada Local
        """
        localFinal = random.randint(0,2)
        print(f"Indice do local final: {localFinal}")

        for indice in range(0,3):
            print(f"Indice para pista de {self.locais[indice].local}: {indice}")
            if indice != localFinal:
                print("Criou pistas de cidade final")
                self.locais[indice].criarPistaCidadeFinal()
            else:
                print("Criou pista prender a zefa")
                self.locais[indice].criarPistaPrendeuAZefa()
        print(f"Pistas cidade final de {self.cidade}: ")
        for loc in self.locais:
            print(f"Local: {loc.local} Pista: {loc.pista}")

    def criarPistasCidadeFalsa(self):
        """
        Caso est seja uma das 3 cidades falsas. Atribuindo as pistas de cidade falsa a
propriedade pista dos locais
        :return: atribuição das pistas a propriedade pista de cada Local
        """
        for local in self.locais:
```

```
            local.criarPistaFalsa()
        print(f"Pistas cidade falsa de {self.cidade}: ")
        for loc in self.locais:
            print(f"Local: {loc.local} Pista: {loc.pista}")


    def criarCidadesDestino(self, cidadesDestino):
        """
        Atribuindo as cidades para onde se pode viajar para a cidade atual
        :param cidadesDestino: array com as cidades para onde se pode viajar
        :return: atribui a lsita de cidades para a propriedade cidadesDeDestino
        """
        self.cidadesDeDestino = []
        for cidade in cidadesDestino:
            self.cidadesDeDestino.append(cidade)


    def embaralharDestinos(self):
        """
        Embaralha os destinos para que não fique previsivel a rota (de modo similar, você pode
verificar o método do Python shuffle que têm o mesmo princípio)
        :return:
        """
        destinosTemp = []
        elementos = len(self.cidadesDeDestino)
        print(f"Embaralhando destinos para {self.cidade}")
        print(f"Destinos antes de embaralhar")
        for d in self.cidadesDeDestino:
            print(d.cidade)
        for i in range(elementos):
            indice = random.randint(0, (len(self.cidadesDeDestino) - 1))
            destinosTemp.append(self.cidadesDeDestino.pop(indice))

        print(f"Depois de emparalhar no temp os destinos ficaram: ")
        print(self.cidadesDeDestino)
        for destino in destinosTemp:
            self.cidadesDeDestino.append(destino)
        print(f"Depois de embaralhar os destinos ficaram")
        for dest in self.cidadesDeDestino:
            print(dest.cidade)


    def calcularTempoDeVoo(self, cidade):
        """
        Calcular o tempo de voo entre a cidade atual e a cidade de destino
        :param: cidade de destino
        :return:
        """
        velocidadeAviao = 850 # velocidade em km por hora
        origem = (float(self.latitude),float(self.longitude))
```

```
        destino = (float(cidade.latitude),float(cidade.longitude))
        distancia = hs.haversine(origem,destino) # retorna valor em km
        print(f"Distância: {distancia}")
        horas = round(distancia/velocidadeAviao) # nosso avião é miraculoso e anda sempre na
mesma velocidade sem aceleração nenhuma, e as horas sempre dão um número redondinho
        print(f"Horas de vôo {cidade.cidade}: {horas}")
        return horas
```

Classe Local:

```
import random
class Local:
    """
    Esta classe que representa o local de pista
    """
    def __init__(self, conexaoBD, id, local, bandeira, pistaCidade, moeda):
        self.zefaEstaAqui = False
        self.conexaoBD = conexaoBD
        self.id = id
        self.local = local
        self.bandeira = bandeira
        self.pistaCidade = pistaCidade
        self.moeda = moeda
        self.personagem = ""
        self.criarPersonagem()









    def criarPersonagem(self):
        sql = f"select * from personagens as p INNER JOIN local_personagem as lp on
(p.pk_id_personagem = lp.fk_id_personagem) where lp.fk_id_local = :id"
        parametros = {"id": self.id}
        personagens = self.conexaoBD.lerDadosParam(sql, parametros)
        # print(curiosidades)
        quantidadeDePersonagens = len(personagens)
        indice = 0
        if quantidadeDePersonagens > 0:
            if quantidadeDePersonagens > 1:
                indice = random.randint(0, quantidadeDePersonagens - 1)
                # print("Pra saber o indice")
                # print(personagens[indice])
            self.personagem = personagens[indice]["personagem"]
        else:
            self.personagem = "Não há um personagem com esse id"
```

```
    def criaPistaBandeira(self, bandeira, imagem):
        if self.local == "Aeroporto":
            veiculo = "O avião que ela embarcou tinha"
        elif self.local == "Estação de ônibus":
            veiculo = "Ela pegou um ônibus com"
        else:
            veiculos = ["Ela foi embora em um carro com", "Ela partiu disparada em uma moto
com"]
            veiculo = veiculos[random.randint(0,1)]
        if imagem == True:
            self.pista = f"{veiculo} uma bandeira com {bandeira}"
        else:
            self.pista = f"{veiculo} uma bandeira {bandeira}"

        print(f"A pista da bandeira para {self.local} é: {self.pista}")

    def criarPistaMoeda(self, moeda):
        self.pista = f"Só sei que ela comprou {moeda}"
        print(f"A pista da moeda para {self.local} é: {self.pista}")

    def criarPistaCidade(self, pistas):
        """
        Sorteia uma pista da cidade e atribui ela como pista do local.
        Além disso, remove essa pista do array de pistas para que não possa ser usada
novamente e repetida.
        :param pistas: array de pistas da cidade
        """

        print("Pistas da cidade de destino")
        print(pistas)
        quantidadeDePistas = len(pistas)
        if quantidadeDePistas > 0 :
            if quantidadeDePistas > 1:
                indice = random.randint(0, quantidadeDePistas - 1)
                print(f"Random int pista cidade: {indice}")
                self.pista = pistas[indice]
                pistas.pop(indice)
            else:
                self.pista = pistas[0]
        else:
            self.pista = "Não há pistas para esta cidade"

        print(f"A pista da cidade para {self.local} é: {self.pista}")




    def criarPistaFalsa(self):
        naoVi = ["Nunca vi alguém assim.", "Nunca vi essa pessoa.", "Não conheço não.",
```

```
"Você tem certeza que ela esteve aqui?", "Acho que você se enganou, não passou ninguém assim por aqui.", "Quem?", "Tô sabendo de nada não.", "Acho que você está na pista errada."]
        self.pista = naoVi[random.randint(0, len(naoVi) - 1)]
        print(f"A pista da falsa para {self.local} é: {self.pista}")

    def criarPistaCidadeFinal(self):
        cuidado = ["Se eu fosse você eu tomava cuidado.", "A Zefa já está sabendo que você está atrás dela e não está feliz.", "Tem uma recompensa pela sua cabeça.", "Você está cutucando vespeiro", "Não me envolve nisso, não quero rolo pro meu lado.", "Se eu fosse você fugia.", "Some daqui, não quero saber de problemas."]
        self.pista = cuidado[random.randint(0, len(cuidado) - 1)]
        print(f"A pista da cidade final para {self.local} é: {self.pista}")

    def criarPistaPrendeuAZefa(self):
        self.pista = "Parabéns! Você capturou a Zefa!"
        self.zefaEstaAqui = True
        # self.localFinal = True
        print(f"A pista de prendeu a Zefa para {self.local} é: {self.pista}")
```

Classe Jogo:

```
from Model.ConexaoBD import ConexaoBD
from Model.Cidade import Cidade
import random
class Jogo:
    """
    Classe que criaa o jogo. Criação e Lógica de jogo
    """
    # Com este roteiro sabemos para que cidades podemos viajar de cada cidade. Isso porque os destinos possíveis refere-se sempre aos indices, que batem com os índices da lista self.cidades. Então, por exemplo, os destinos para onde se pode viajar desde a cidade que estiver no índice 1 do array selfcidades são as dos índices 5, 2 e 8 do mesmo array
    destinosPossiveis = [[2,4,7],[5,2,8],[3,1,0],[4,2,7],[0,3,5],[1,6,4],[0,2,5],[0,3,8],[0,1,7]]
    # Com este roteiro, buscamos a pista da cidade. Exemplo, se o detevive está na cidade do índice 0 de self.cidades então as pists mostradas devem ser as da cidade do índice 4
    roteiroZefa = [4,5,1,2,3,False, False, False, False]
    def __init__(self):
        self.conexaoBD = ConexaoBD()
        # self.tempoDeJogo = 152
        self.acordouEm = 152
        self.tempoJogado = 152
        self.criarJogo()


    def sortearCidades(self):
        sql = "select pk_id_cidade from cidades"
        resultados = self.conexaoBD.lerDados(sql)
        print(resultados)
        indicesCidades = []
```

```
        # Sorteando os indices das cidades
        if len(resultados) > 0:
            for indiceCidade in range(0,9):
                indice = random.randint(0, len(resultados) - 1) # Aqui decidi não usar a função choice, também da biblioteca random, pois além de sortear quero fazer pop no elemento. Mas se te interessar você pode ir descobrir mais sobre essa função na documentação do Python em: https://docs.python.org/3/library/random.html#random.choice
                print(resultados[indice][0])
                indicesCidades.append(resultados[indice]["pk_id_cidade"])
                resultados.pop(indice)
            print(indicesCidades)
            return indicesCidades

        else:
            self.erro = "Não há cidades"
    def criarJogo(self):
        # Escolhendo as cidades
        indicesCidades = self.sortearCidades()
        # Criando a lista com as 9 cidades
        self.criarListaDeCidades(indicesCidades)

        # Buscando a informação de caso para a cidade no indice 0
        self.criarCaso()
        #Buscando a informação de pistas para os indices de 0 a 4
        for i in range(0,5):
            # As pistas que busco não são da cidade, mas sim da cidade para a qual se deveria viajar se eu estiver nessa cidade. Minha cidade de destino ideal. para isso eu uso o Array do Roteiro da Zefa para saber para onde a Zefa foi quando estava na cidade do índice i e recolher as pistas dessa cidade
            self.cidades[i].criandoAsPistas(self.cidades)

        # Criando a informação da cidade final do índice 5
        self.cidades[5].criarPistasCidadeFinal()

        # Criando a informação de pista para os indices de 6,7,8
        for x in range(6,9):
            self.cidades[x].criarPistasCidadeFalsa()

        # Atribuindo destinos às cidades
        self.criandoDestinosPossiveis()

        # Embaralhando cidades
        self.embaralharCidades()

        # Embaralhando os destinos
        for cidade in self.cidades:
            cidade.embaralharDestinos()
```

```
    def criandoDestinosPossiveis(self):
        """
        Atribui as cidades de destino de viagem para cada cidade da lista de cidades
        :return:
        """
        index = 0
        for cid in self.cidades:
            arrayTemp= []
            print("Indice destinos possiveis")
            print(self.destinosPossiveis[index])
            for destinos in self.destinosPossiveis[index]:
                arrayTemp.append(self.cidades[destinos])
            cid.criarCidadesDestino(arrayTemp)
            for destino in cid.cidadesDeDestino:
                print(f"Destino da cidade {cid.cidade} é {destino.cidade}")
            index += 1

    def criarListaDeCidades(self, indicesCidades):
        """
        Com os índices das cidades escolhidos, ele monta a lista com os objetos Cidade.
        Faz isto em duas etapas para não desperdiçar memória pegando toda a informação de
cidades e países antes de definir quais cidades serão as do jogo. Dessa maneira pegando a
informação só das cidades necessárias
        :param indicesCidades: o índice das cidades criado criarJogo
        :return:
        """
        self.cidades = []
        # Buscando informação de cada cidade
        for indice in range(0,9):
            self.cidades.append(Cidade(self.conexaoBD, indicesCidades[indice],
self.roteiroZefa[indice]))


        # Definindo a cidade em que o jogo se encontra, no caso a primeira que é sempre a de
indice 0
        self.cidadeAtual = self.cidades[0]
        self.cidadeInicial = self.cidades[0]

    def criarCaso(self):
        """
        Cria o caso para a cidade do índice 0
        :return:
        """
        self.cidades[0].criarCaso()

    def embaralharCidades(self):
        """
        Embaralha as cidades para que não fique previsivel a rota, (de modo similar, você pode
verificar o método do Python shuffle que têm o mesmo princípio)
        :return:
```

```
        """
        cidadesTemp = []
        elementos = len(self.cidades)
        print(f"Antes de embaralhar as cidades ficaram")
        for c in self.cidades:
            print(c.cidade)
        for i in range(elementos):
            indice = random.randint(0, (len(self.cidades) - 1))
            cidadesTemp.append(self.cidades.pop(indice))

        print(f"Depois de emparalhar no temp as cidades ficaram: ")
        print(self.cidades)
        for cidade in cidadesTemp:
            self.cidades.append(cidade)
        print(f"Depois de embaralhar as cidades ficaram")
        for cid in self.cidades:
            print(cid.cidade)
```

Pacote View:

Pacote Computador:

Classe Computador:

```
from Controller.FluxoDeJogoComp import FluxoDeJogoComp
from View.Computador.Constantes import Estilos, Elementos
# Importamos a biblioteca tkinter para fazer as interfaces gráficas
import tkinter as tk

# Para usar fontes
from tkinter import font

## Importando as outras Views
from View.Computador.InfoComEscolha import InfoComEscolha
from View.Computador.InfoSemEscolha import InfoSemEscolha
from View.Computador.Info import Info
from View.Computador.Menu import Menu
from View.Computador.Inicio import Inicio
class Computador:
    """
    Esta classe controla a visualização do jogo para desktop.
    """
    def __init__(self):
        # Variável que vai receber a escolha feita. Está variável vai ser usada pelo Fluxo de
```

```
Jogo
        # Iniciando o jogo
        self.jogo = FluxoDeJogoComp(self)
        # Criando a janela
        self.app = tk.Tk()
        # Atribuindo título a janela
        self.app.title("Cadê a Zefa, gente!")
        # Atribuindo uma cor de fundo a janela
        self.app.configure(background=Estilos.FUNDO)
        ## Descobrindo a altura e largura da janela para poder posicionar nossos elementos
        self.larguraJanela = self.app.winfo_screenwidth()
        self.alturaJanela = self.app.winfo_screenheight()
        # self.larguraJanela = 1240
        # self.alturaJanela = 600
        # Quebrando a janela em um Grid
        self.janelaColuna = self.larguraJanela / 24
        self.janelaLinha = (self.alturaJanela / 24) - 10

        # Definindo o wrap do texto
        self.wrap = self.larguraJanela - Estilos.PADX
        print(f"Largura da janela: {self.larguraJanela}")
        print(f"Altura da janela: {self.alturaJanela}")
        print(f"Altura da linha: {self.janelaLinha}")
        # Com geometry eu dou um tamanho e posiciono a janela.
        # self.app.geometry(f"{self.larguraJanela}x{self.alturaJanela}+200+200")
        # Define a janela como sendo fullscreen. Como nosso interesse é fazer um emulador que
em teoria teria somente este jogo, é interessante pois não vai aparecer aqueles tradicionais
botões de janela do sistema operacional. Porém é meio chato na hora de programar porque
ocupa tudo. Por isso o método acima para a hora da programação. E trocar no deploy
        self.app.attributes('-fullscreen',True)
        # Definindo o ícone da janela
        self.app.iconbitmap(default="recursos/imagens/zefa.ico")

        # Criando as fontes que quero usar
        self.titulo = font.Font(weight="bold", family=Estilos.FONTE, size=32, slant="italic")
        self.tituloNormal = font.Font(weight="bold", family=Estilos.FONTE, size=32)
        self.negrito = font.Font(weight="bold", family=Estilos.FONTE, size=24)
        self.textoNormal = font.Font(family=Estilos.FONTE, size=24)

        #Aqui vou criar três frames principais de informação
        # Informação
        # Botões - Opcoes
        # Tempo De Jogo Restante
        self.frameJogo = Elementos.criarFrame(self)
        self.frameOpcoes = Elementos.criarFrame(self)
        self.frameTempo = Elementos.criarFrame(self)


        # Agora tenho que posicionar os frames
        Elementos.posicionarFrame(self.frameJogo)
```

```
        Elementos.posicionarFrame(self.frameOpcoes)
        Elementos.posicionarFrame(self.frameTempo)


        # Agora vou criar as caixas de texto para a informação de tempo de jogo
        self.tituloTempo = Elementos.criarTitulo(self, self.frameTempo, "Tempo
remanescente:")
        self.tempo =
Elementos.criarTitulo(self,self.frameTempo,f"{self.jogo.jogo.tempoJogado}")

        Elementos.posicionarTitulo(self.tituloTempo)
        Elementos.posicionarTitulo(self.tempo)


        # Quero simplesmente iniciar minhas janelas investigar, viajar, destinos, e info mas não
mostrar ela ainda ou passar informação
        self.janelaInvestigar = InfoComEscolha(self, "investigar")
        self.janelaDestinos = InfoSemEscolha(self)
        self.janelaViajar = InfoComEscolha(self, "viajar")
        self.janelaInfo = Info(self)
        self.janelaMenu = Menu(self)
        self.janelaInicio = Inicio(self)
        #Mostrar a janela de caso novo

        self.janelaInicio.reiniciar()



        # Aqui eu tenho que iniciar o loop da janela

        self.app.mainloop()




    def limparTudo(self):
        self.janelaInvestigar.limparTudo()
        self.janelaViajar.limparTudo()
        self.janelaDestinos.limparTudo()
        self.janelaInfo.limparTudo()
        self.janelaMenu.limparTudo()
        self.janelaInicio.limparTudo()

    def atualizarTempo(self, info):
        """
            Atualiza a informação de tempo
            :return:
        """
        Elementos.apagarElementoDaTela([self.tempo])
```

```
        self.tempo = Elementos.criarTitulo(self, self.frameTempo,f"{info}
{self.jogo.jogo.tempoJogado}")
        Elementos.posicionarTitulo(self.tempo)

    def jogar(self):
        self.limparTudo()
        self.janelaMenu.reiniciar(f"Boas vindas a {self.jogo.jogo.cidadeAtual.cidade}",
self.jogo.jogo.cidadeInicial.curiosidade)
```

Classes: Elementos e Constantes:

```
import tkinter as tk
class Estilos:
    """
    Classe com as constantes de estilo.
    """
    FUNDO = "#6e09e2"
    FRAME = "#AA7DE3"
    INFO = "#DE97FA"
    VERMELHO = "#FF0000"
    VERMELHOCL = "#FF2D00"
    AZUL = "#0000FF"
    AZULCL = "#0051FF"
    AMARELO = "#FFFF00"
    AMARELOCL = "#FFE000"
    BRANCO = "#FFFFFF"
    BRANCOCL = "#CCCCCC"
    PRETO = "#000000"
    PRETOCL = "#444444"
    BORDA = 2
    RELEVO = "sunken"
    FONTE = "Consolas"
    PADX = 2
    PADY = 2

class Elementos:
    """
    Classe com métodos para ajudar a criar os elementos com a mesma estilização
    """
    @staticmethod
    def criarBotao(janela, texto, funcao, corBotao, corTexto):
        """
        Cria elemento de botão já com o estilo correto
```

```
        :param janela: elemento de app
        :param texto: o texto do botão
        :param funcao: a função que será chamada pelo botão - sem ()
        :param argumento: argumento para funcao.
        :param corBotao: a cor do botão
        :param corTexto: a cor do texto
        :return:
        """
        return tk.Button(janela.frameOpcoes, text=texto, foreground=corTexto,
background=corBotao, command=lambda: funcao(), padx=Estilos.PADX,
pady=Estilos.PADY,font=janela.textoNormal, takefocus=True)

    @staticmethod
    def posicionarBotao(botao):
        """
        Posicionar o botão
        :param botao: o botão criado
        :param pos: posição do botão
        :return:
        """

        botao.pack(fill=tk.BOTH, expand=1, side=tk.LEFT, padx=Estilos.PADX,
pady=Estilos.PADY)
    @staticmethod
    def criarTitulo(janela, frame, titulo):
        """
        Criar um elemento de título já com os estilos
        :param janela: elemento de app
        :param frame: o frame onde o título vai ser posicionado
        :param titulo: o título
        :return:
        """

        return tk.Label(frame, text=titulo, font=janela.titulo,bg=Estilos.FRAME,
foreground=Estilos.BRANCO, wraplength=janela.wrap, anchor="center")
    @staticmethod
    def posicionarTitulo(titulo):
        """
        Posicionar título
        :param titulo: o título criado
        :param pos: posicão
        :return:
        """
        titulo.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)

    @staticmethod
    def criarTexto(janela, frame, texto):
        """
        Criar um elemento de texto já com estilso
        :param janela: elemento de app
```

```
        :param frame: frame onde o texto será posicionado
        :param texto: texto
        :return:
        """
        return tk.Label(frame, text=texto, font=janela.textoNormal, bg=Estilos.FRAME,
foreground=Estilos.BRANCO, wraplength=janela.wrap, anchor="center")

    @staticmethod
    def posicionarTexto(texto):
        """
        Posiciona o texto
        :param texto: elemento de texto
        :param pos: posição
        :return:
        """
        texto.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)
    @staticmethod
    def criarFrame(janela):
        """
        Cria um elemento de Frame já com os estilos
        :param janela: elemento de app
        :return:
        """
        return tk.Frame(janela.app, width=janela.larguraJanela, height=(janela.janelaLinha * 8),
bg=Estilos.FRAME, borderwidth=Estilos.BORDA, relief=Estilos.RELEVO, takefocus=True)
    @staticmethod
    def posicionarFrame(frame):
        """
        Posiciona o frame
        :param frame: elemento de frame
        :param pos: posição
        :return:
        """
        frame.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)
        # frame.grid(row=pos, column=0, padx=Estilos.PADX, pady=Estilos.PADY)
    @staticmethod
    def apagarElementoDaTela(elementos):
        """
        "Esquece" o elemento da tela para que ele possa ser alterado. Método para facilitar a
alteração de telas e para facilitar caso se queira alterar o modo de geometria do Tkinter
(place, pack, grid)
        :param elemento: array com os elemento a ser esquecido
        :return:
        """
        for elemento in elementos:
            elemento.forget()
```

Classe Info:

```
from View.Computador.Constantes import Estilos, Elementos
class Info:
    """
    Esta classe mostra a informação final
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
        :param titulo: O título que precisa mostrar
        :param info: O texto que precisa mostrar
        """
        self.janela = janela

        # Variável de controle
        self.on = False




    def mostrarInstrucoes(self, titulo, texto):
        """
        Mostra as informações
        :return:
        """
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)




    def limparTudo(self):
        """
        Apaga os frames com informação e botões
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self, titulo, texto):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
```

```
        :return:
        """
        self.on = True
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """
        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe InfoComEscolha:

```
from View.Computador.Constantes import Estilos, Elementos
class InfoComEscolha:
    """
    Esta classe mostra as informações de opções com escolhas como Investigar e Viajar e o
principal
    """
    def __init__(self, janela, tipo):
        """
        Mostra as informações que terão uma escolha
        :param janela: A janela do app
        :param titulo: O título que precisa colocar
        :param tipo: "investigar" ou "viajar". A decisão do tipo serve para saber quanto
descontar do tempo de viagem
        """
        self.janela = janela
        self.tipo = tipo
        # Aqui iniciamos a variável com valores padrões que depois serão trocados
simplesmente para não causar bugs de inicialização
        if self.tipo == "investigar":
            self.tempoViagem = 1
        else:
            self.tempoViagem = 10
        self.opcoes = []
```

```
        # Variável de controle
        self.on = False



    def atualizarInfo(self, opcoes):
        """
        Atualiza a informação das variáveis
        :param titulo:
        :param texto:
        :param locais:
        :return:
        """

        self.opcoes.clear()
        for opcao in opcoes:
            self.opcoes.append(opcao)

    def mostrarInstrucoes(self, titulo, texto):
        # Para apagar o texto que tinha antes
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)



    def limparInfo(self):
        Elementos.apagarElementoDaTela([self.titulo, self.info])

    def limparTudo(self):
        """
        Apaga toda a informação
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botao1, self.botao2,
self.botao3, self.botaoV, self.botaoS, self.botaoD])
            self.on = False



    def trocarInfo(self, titulo, texto):
        self.limparInfo()
        self.mostrarInstrucoes(titulo, texto)

    def deslocar(self, opcao):

        if self.tipo == "investigar":
```

```
            self.localDestino = self.janela.jogo.jogo.cidadeAtual.locais[opcao].local
            if self.testarTempo() == False:
                return
            self.trocarInfo(self.janela.jogo.jogo.cidadeAtual.locais[opcao].local,
f"{self.janela.jogo.jogo.cidadeAtual.locais[opcao].personagem}:
{self.janela.jogo.jogo.cidadeAtual.locais[opcao].pista}")


            # Vendo se ganhou
            if self.janela.jogo.jogo.cidadeAtual.locais[opcao].zefaEstaAqui:
self.janela.jogo.ganhou()

        else:


            self.localDestino = self.janela.jogo.jogo.cidadeAtual.cidadesDeDestino[opcao].cidade
            self.tempoViagem =
self.janela.jogo.jogo.cidadeAtual.calcularTempoDeVoo(self.janela.jogo.jogo.cidadeAtual.cida
desDeDestino[opcao])

            if self.testarTempo() == False:
                return

            # self.janela.atualizarTempo()

self.janela.jogo.novaCidade(self.janela.jogo.jogo.cidadeAtual.cidadesDeDestino[opcao])



    def testarTempo(self):
        """
        Testa o tempo para ver se finalizou o jogo, e se precisa dormir
        :return: True - pode continuar o jogo | False: acabou o tempo de jogo
        """
        if self.janela.jogo.deslocar(self.tempoViagem, self.localDestino) == False:
            self.janela.jogo.acabouJogo()
            return False

        dormiu = self.janela.jogo.dormir()
        if dormiu > 0:
            print("Entrou em tem que dormir")
            if dormiu == 1:
                self.janela.jogo.acabouJogo()
                return False
        return True
    def reiniciar(self, titulo, texto, opcoes):
        # self.limparTudo()
        self.atualizarInfo(opcoes)
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
```

```
        self.on = True
    def criandoBotoes(self):

        # Criando os botões
        self.botao1 = Elementos.criarBotao(self.janela, self.opcoes[0], lambda: self.deslocar(0),
Estilos.AMARELO, Estilos.PRETO)

        self.botao2 = Elementos.criarBotao(self.janela, self.opcoes[1], lambda: self.deslocar(1),
Estilos.BRANCO, Estilos.PRETO)

        self.botao3 = Elementos.criarBotao(self.janela, self.opcoes[2], lambda: self.deslocar(2),
Estilos.AZUL, Estilos.BRANCO)

        self.botaoV = Elementos.criarBotao(self.janela, "Voltar", self.janela.jogo.voltar,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões
        Elementos.posicionarBotao(self.botao1)
        Elementos.posicionarBotao(self.botao2)
        Elementos.posicionarBotao(self.botao3)
        Elementos.posicionarBotao(self.botaoV)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe InfoSemEscolha:

```
from View.Computador.Constantes import Estilos, Elementos
class InfoSemEscolha:
    """
    Esta classe mostra a informação de opções sem escolha como Destinos
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
```

```
    :param titulo: O título que precisa mostrar
    :param info: O texto que precisa mostrar
    """
    self.janela = janela
    self.info = []

    # Variável de controle
    self.on = False




def atualizarInfo(self, titulo, info):
    """
    Atualiza a informação das variáveis
    :param titulo:
    :param texto:
    :param locais:
    :return:
    """
    self.tituloTexto = titulo
    self.info.clear()
    for opcao in info:
        self.info.append(opcao)

def mostrarInstrucoes(self):
    """
    Mostra as informações
    :return:
    """
    self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, self.tituloTexto)
    self.info1 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[0])
    self.info2 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[1])
    self.info3 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[2])

    Elementos.posicionarTitulo(self.titulo)
    Elementos.posicionarTexto(self.info1)
    Elementos.posicionarTexto(self.info2)
    Elementos.posicionarTexto(self.info3)




def limparTudo(self):
    """
    Apaga os frames com informação e botões
    :return:
    """
```

```
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info1, self.info2, self.info3,
self.botaoV, self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self, titulo, info):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
        :return:
        """
        self.on = True
        self.atualizarInfo(titulo, info)
        self.mostrarInstrucoes()
        self.criandoBotoes()

    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """

        # Criando os botões
        # Criando os botões

        self.botaoV = Elementos.criarBotao(self.janela, "Voltar", self.janela.jogo.voltar,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões


        Elementos.posicionarBotao(self.botaoV)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe Inicio:

```
from View.Computador.Constantes import Estilos, Elementos
class Inicio:
    """
    Esta classe mostra a informação inicial
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
        :param titulo: O título que precisa mostrar
        :param info: O texto que precisa mostrar
        """
        self.janela = janela

        # Variável de controle
        self.on = False




    def mostrarInstrucoes(self):
        """
        Mostra as informações
        :return:
        """
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo,
self.janela.jogo.jogo.cidadeInicial.tituloCaso)
        self.caso = Elementos.criarTexto(self.janela, self.janela.frameJogo,
self.janela.jogo.jogo.cidadeInicial.caso)
        self.tarefa = Elementos.criarTexto(self.janela, self.janela.frameJogo,
                                           f"Sua tarefa: {self.janela.jogo.jogo.cidadeInicial.tarefaCaso}")
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.caso)
        Elementos.posicionarTexto(self.tarefa)




    def limparTudo(self):
        """
        Apaga os frames com informação e botões
        :return:
        """
```

```
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.caso, self.tarefa, self.botaoJ,
self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
        :return:
        """
        self.on = True
        self.mostrarInstrucoes()
        self.criandoBotoes()
    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """

        self.botaoJ = Elementos.criarBotao(self.janela, "Jogar", self.janela.jogar, Estilos.AZUL,
Estilos.BRANCO)
        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        Elementos.posicionarBotao(self.botaoJ)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe Menu:

```
from View.Computador.Constantes import Estilos, Elementos
class Menu:
    def __init__(self, janela):
        """
        Esta classe mostra o Menu Principal
        :param janela: A janela do app
        :param titulo: O título a ser mostrado
```

```
        :param texto:  O texto a ser mostrado
        """
        self.janela = janela
        # Variável de controle
        self.on = False


    def mostrarInstrucoes(self, titulo, texto):
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)

        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)




    def limparTudo(self):
        """
        Apaga toda a informação
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botao1, self.botao2,
self.botao3, self.botaoS, self.botaoD])
            self.on = False



    def reiniciar(self, titulo, texto):
        self.limparTudo()
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
        self.on = True
    def criandoBotoes(self):

        # Criando os botões
        self.botao1 = Elementos.criarBotao(self.janela, "Investigar", self.janela.jogo.investigar,
Estilos.AMARELO, Estilos.PRETO)

        self.botao2 = Elementos.criarBotao(self.janela, "Viajar", self.janela.jogo.viajar,
Estilos.BRANCO, Estilos.PRETO)

        self.botao3 = Elementos.criarBotao(self.janela, "Destinos", self.janela.jogo.destinos,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
```

```
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões
        Elementos.posicionarBotao(self.botao1)
        Elementos.posicionarBotao(self.botao2)
        Elementos.posicionarBotao(self.botao3)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Pacote Display7:

Classe Display7:

```
from Controller.FluxoDeJogoBotoes7 import FluxoDeJogoBotoes7
from View.Display7.Constantes import Estilos, Elementos
# Importamos a biblioteca tkinter para fazer as interfaces gráficas
import tkinter as tk

# Para usar fontes
from tkinter import font

## Importando as outras Views
from View.Display7.InfoComEscolha import InfoComEscolha
from View.Display7.InfoSemEscolha import InfoSemEscolha
from View.Display7.Info import Info
from View.Display7.Menu import Menu
from View.Display7.Inicio import Inicio
class Display7:
    """
    Esta classe controla a visualização para o display de 7 polegadas.
    """
    def __init__(self):
        # Variável que vai receber a escolha feita. Está variável vai ser usada pelo Fluxo de
Jogo
        # Iniciando o jogo
        self.jogo = FluxoDeJogoBotoes7(self)
        # Criando a janela
        self.app = tk.Tk()
        # Atribuindo título a janela
        # self.app.title("Cadê a Zefa, gente!")
        # Atribuindo uma cor de fundo a janela
        self.app.configure(background=Estilos.FUNDO)
        ## Descobrindo a altura e largura da janela para poder posicionar nossos elementos
        self.larguraJanela = self.app.winfo_screenwidth()
        self.alturaJanela = self.app.winfo_screenheight()
        # self.larguraJanela = 1240
        # self.alturaJanela = 600
```

```
        # Quebrando a janela em um Grid
        self.janelaColuna = self.larguraJanela / 24
        self.janelaLinha = (self.alturaJanela / 24) - 10

        # Definindo o wrap do texto
        self.wrap = self.larguraJanela - Estilos.PADX
        print(f"Largura da janela: {self.larguraJanela}")
        print(f"Altura da janela: {self.alturaJanela}")
        print(f"Altura da linha: {self.janelaLinha}")
        # Com geometry eu dou um tamanho e posiciono a janela.
        # self.app.geometry(f"{self.larguraJanela}x{self.alturaJanela}+200+200")
        # Define a janela como sendo fullscreen. Como nosso interesse é fazer um emulador que em teoria teria somente este jogo, é interessante pois não vai aparecer aqueles tradicionais botões de janela do sistema operacional. Porém é meio chato na hora de programar porque ocupa tudo. Por isso o método acima para a hora da programação. E trocar no deploy
        self.app.attributes('-fullscreen',True)
        # Definindo o ícone da janela
        # self.app.iconbitmap(default="recursos/imagens/zefa.ico")

        # Criando as fontes que quero usar
        self.titulo = font.Font(weight="bold", family=Estilos.FONTE, size=32, slant="italic")
        self.tituloNormal = font.Font(weight="bold", family=Estilos.FONTE, size=32)
        self.negrito = font.Font(weight="bold", family=Estilos.FONTE, size=24)
        self.textoNormal = font.Font(family=Estilos.FONTE, size=24)

        #Aqui vou criar três frames principais de informação
        # Informação
        # Botões - Opcoes
        # Tempo De Jogo Restante
        self.frameJogo = Elementos.criarFrame(self)
        self.frameOpcoes = Elementos.criarFrame(self)
        self.frameTempo = Elementos.criarFrame(self)


        # Agora tenho que posicionar os frames
        Elementos.posicionarFrame(self.frameJogo)
        Elementos.posicionarFrame(self.frameOpcoes)
        Elementos.posicionarFrame(self.frameTempo)


        # Agora vou criar as caixas de texto para a informação de tempo de jogo
        self.tituloTempo = Elementos.criarTitulo(self, self.frameTempo, "Tempo
remanescente:")
        self.tempo =
Elementos.criarTitulo(self,self.frameTempo,f"{self.jogo.jogo.tempoJogado}")

        Elementos.posicionarTitulo(self.tituloTempo)
        Elementos.posicionarTitulo(self.tempo)
```

```
        # Quero simplesmente iniciar minhas janelas investigar, viajar, destinos, e info mas não
mostrar ela ainda ou passar informação
        self.janelaInvestigar = InfoComEscolha(self, "investigar")
        self.janelaDestinos = InfoSemEscolha(self)
        self.janelaViajar = InfoComEscolha(self, "viajar")
        self.janelaInfo = Info(self)
        self.janelaMenu = Menu(self)
        self.janelaInicio = Inicio(self)
        #Mostrar a janela de caso novo

        self.janelaInicio.reiniciar()



        # Aqui eu tenho que iniciar o loop da janela

        self.app.mainloop()




    def limparTudo(self):
        self.janelaInvestigar.limparTudo()
        self.janelaViajar.limparTudo()
        self.janelaDestinos.limparTudo()
        self.janelaInfo.limparTudo()
        self.janelaMenu.limparTudo()
        self.janelaInicio.limparTudo()

    def atualizarTempo(self, info):
        """
            Atualiza a informação de tempo
            :return:
        """
        Elementos.apagarElementoDaTela([self.tempo])

        self.tempo = Elementos.criarTitulo(self, self.frameTempo,f"{info}
{self.jogo.jogo.tempoJogado}")
        Elementos.posicionarTitulo(self.tempo)

    def jogar(self):
        self.limparTudo()
        self.janelaMenu.reiniciar(f"Boas vindas a {self.jogo.jogo.cidadeAtual.cidade}",
self.jogo.jogo.cidadeInicial.curiosidade)
```

Classes: Elementos e Constantes:

```
import tkinter as tk
class Estilos:
    """
    Classe com as constantes de estilo
    """
    TEMPO = 100
    FUNDO = "#6e09e2"
    FRAME = "#AA7DE3"
    INFO = "#DE97FA"
    VERMELHO = "#FF0000"
    VERMELHOCL = "#FF2D00"
    AZUL = "#0000FF"
    AZULCL = "#0051FF"
    AMARELO = "#FFFF00"
    AMARELOCL = "#FFE000"
    BRANCO = "#FFFFFF"
    BRANCOCL = "#CCCCCC"
    PRETO = "#000000"
    PRETOCL = "#444444"
    BORDA = 2
    RELEVO = "sunken"
    FONTE = "Consolas"
    PADX = 2
    PADY = 2

class Elementos:
    """
    Classe com métodos para ajudar a criar os elementos com a mesma estilização
    """
    @staticmethod
    def criarBotao(janela, texto, funcao, corBotao, corTexto):
        """
        Cria elemento de botão já com o estilo correto
        :param janela: elemento de app
        :param texto: o texto do botão
        :param funcao: a função que será chamada pelo botão - sem ()
        :param argumento: argumento para funcao.
        :param corBotao: a cor do botão
        :param corTexto: a cor do texto
        :return:
        """
        return tk.Button(janela.frameOpcoes, text=texto, foreground=corTexto,
background=corBotao, command=lambda: funcao(), padx=Estilos.PADX,
pady=Estilos.PADY,font=janela.textoNormal, takefocus=True)
```

```
    @staticmethod
    def posicionarBotao(botao):
        """
        Posicionar o botão
        :param botao: o botão criado
        :param pos: posição do botão
        :return:
        """

        botao.pack(fill=tk.BOTH, expand=1, side=tk.LEFT, padx=Estilos.PADX,
pady=Estilos.PADY)
    @staticmethod
    def criarTitulo(janela, frame, titulo):
        """
        Criar um elemento de título já com os estilos
        :param janela: elemento de app
        :param frame: o frame onde o título vai ser posicionado
        :param titulo: o título
        :return:
        """

        return tk.Label(frame, text=titulo, font=janela.titulo,bg=Estilos.FRAME,
foreground=Estilos.BRANCO, wraplength=janela.wrap, anchor="center")
    @staticmethod
    def posicionarTitulo(titulo):
        """
        Posicionar título
        :param titulo: o título criado
        :param pos: posicão
        :return:
        """
        titulo.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)

    @staticmethod
    def criarTexto(janela, frame, texto):
        """
        Criar um elemento de texto já com estilso
        :param janela: elemento de app
        :param frame: frame onde o texto será posicionado
        :param texto: texto
        :return:
        """
        return tk.Label(frame, text=texto, font=janela.textoNormal, bg=Estilos.FRAME,
foreground=Estilos.BRANCO, wraplength=janela.wrap, anchor="center")

    @staticmethod
    def posicionarTexto(texto):
        """
        Posiciona o texto
```

```
        :param texto: elemento de texto
        :param pos: posição
        :return:
        """
        texto.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)
    @staticmethod
    def criarFrame(janela):
        """
        Cria um elemento de Frame já com os estilos
        :param janela: elemento de app
        :return:
        """
        return tk.Frame(janela.app, width=janela.larguraJanela, height=(janela.janelaLinha * 8),
bg=Estilos.FRAME, borderwidth=Estilos.BORDA, relief=Estilos.RELEVO, takefocus=True)
    @staticmethod
    def posicionarFrame(frame):
        """
        Posiciona o frame
        :param frame: elemento de frame
        :param pos: posição
        :return:
        """
        frame.pack(fill=tk.BOTH, expand=1, padx=Estilos.PADX, pady=Estilos.PADY)
        # frame.grid(row=pos, column=0, padx=Estilos.PADX, pady=Estilos.PADY)
    @staticmethod
    def apagarElementoDaTela(elementos):
        """
        "Esquece" o elemento da tela para que ele possa ser alterado. Método para facilitar a
alteração de telas e para facilitar caso se queira alterar o modo de geometria do Tkinter
(place, pack, grid)
        :param elemento: array com os elemento a ser esquecido
        :return:
        """
        for elemento in elementos:
            elemento.forget()
```

Classe Info:

```
from View.Display7.Constantes import Estilos, Elementos
class Info:
    """
    Esta classe mostra a informação final
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
        :param titulo: O título que precisa mostrar
        :param info: O texto que precisa mostrar
```

```
        """
        self.janela = janela

        # Variável de controle
        self.on = False

        # Criando e definindo os botões
        self.criandoBotoes()
        self.definindoBotoes()
        Elementos.apagarElementoDaTela([self.botaoS, self.botaoD]) # Vamos rapidamente
apagar os botões para que eles não apareçam




    def mostrarInstrucoes(self, titulo, texto):
        """
        Mostra as informações
        :return:
        """
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)



    def limparTudo(self):
        """
        Apaga os frames com informação e botões
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self, titulo, texto):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
        :return:
        """
        self.on = True
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
```

```
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))



    def definindoBotoes(self):
        self.possibilidades = ["s", "d"]
        self.botoes = {
            "s": self.botaoS,
            "d": self.botaoD,
        }

    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """
        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe InfoComEscolha:

```
from View.Display7.Constantes import Estilos, Elementos
class InfoComEscolha:
    """
    Esta classe mostra as informações de opções com escolhas como Investigar e Viajar e o
principal
    """
    def __init__(self, janela, tipo):
        """
        Mostra as informações que terão uma escolha
        :param janela: A janela do app
        :param titulo: O título que precisa colocar
        :param tipo: "investigar" ou "viajar". A decisão do tipo serve para saber quanto
descontar do tempo de viagem
        """
        self.janela = janela
        self.tipo = tipo
```

```
        # Aqui iniciamos a variável com valores padrões que depois serão trocados
simplesmente para não causar bugs de inicialização
        if self.tipo == "investigar":
            self.tempoViagem = 1
        else:
            self.tempoViagem = 10
        self.opcoes = ["1", "2", "3"] # Esses são valores temporários para poder criar os botões.
Pois como queremos ter que definir is botões somente uma vez, precisamos ter este array jpa
com 3 elementos

        # Variável de controle
        self.on = False
        # Criando e definindo os botões
        self.criandoBotoes()
        self.definindoBotoes()
        Elementos.apagarElementoDaTela([self.botao1, self.botao2, self.botao3, self.botaoV,
self.botaoS, self.botaoD]) # Vamos rapidamente apagar os botões para que eles não
apareçam



    def atualizarInfo(self, opcoes):
        """
        Atualiza a informação das variáveis
        :param titulo:
        :param texto:
        :param locais:
        :return:
        """

        self.opcoes.clear()
        for opcao in opcoes:
            self.opcoes.append(opcao)

    def mostrarInstrucoes(self, titulo, texto):
        # Para apagar o texto que tinha antes
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)



    def limparInfo(self):
        Elementos.apagarElementoDaTela([self.titulo, self.info])

    def limparTudo(self):
        """
        Apaga toda a informação
        :return:
```

```
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botao1, self.botao2,
self.botao3, self.botaoV, self.botaoS, self.botaoD])
            self.on = False



    def trocarInfo(self, titulo, texto):
        self.limparInfo()
        self.mostrarInstrucoes(titulo, texto)
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))

    def deslocar(self, opcao):

        if self.tipo == "investigar":
            self.localDestino = self.janela.jogo.jogo.cidadeAtual.locais[opcao].local
            if self.testarTempo() == False:
                return
            self.trocarInfo(self.janela.jogo.jogo.cidadeAtual.locais[opcao].local,
f"{self.janela.jogo.jogo.cidadeAtual.locais[opcao].personagem}:
{self.janela.jogo.jogo.cidadeAtual.locais[opcao].pista}")


            # Vendo se ganhou
            if self.janela.jogo.jogo.cidadeAtual.locais[opcao].zefaEstaAqui:
self.janela.jogo.ganhou()

        else:


            self.localDestino = self.janela.jogo.jogo.cidadeAtual.cidadesDeDestino[opcao].cidade
            self.tempoViagem =
self.janela.jogo.jogo.cidadeAtual.calcularTempoDeVoo(self.janela.jogo.jogo.cidadeAtual.cida
desDeDestino[opcao])

            if self.testarTempo() == False:
                return

            # self.janela.atualizarTempo()

self.janela.jogo.novaCidade(self.janela.jogo.jogo.cidadeAtual.cidadesDeDestino[opcao])



    def testarTempo(self):
        """
        Testa o tempo para ver se finalizou o jogo, e se precisa dormir
        :return: True - pode continuar o jogo | False: acabou o tempo de jogo
```

```
        """
        if self.janela.jogo.deslocar(self.tempoViagem, self.localDestino) == False:
            self.janela.jogo.acabouJogo()
            return False

        dormiu = self.janela.jogo.dormir()
        if dormiu > 0:
            print("Entrou em tem que dormir")
            if dormiu == 1:
                self.janela.jogo.acabouJogo()
                return False
        return True
    def reiniciar(self, titulo, texto, opcoes):
        # self.limparTudo()
        self.atualizarInfo(opcoes)
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
        self.on = True
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))
    def definindoBotoes(self):
        """
        Definindo os botões a serem escolhidos
        """
        self.possibilidades = ["1", "2", "3", "v", "s", "d"]
        self.botoes = {
            "1": self.botao1,
            "2": self.botao2,
            "3": self.botao3,
            "v": self.botaoV,
            "s": self.botaoS,
            "d": self.botaoD,

        }
    def criandoBotoes(self):

        # Criando os botões
        self.botao1 = Elementos.criarBotao(self.janela, self.opcoes[0], lambda: self.deslocar(0),
Estilos.AMARELO, Estilos.PRETO)

        self.botao2 = Elementos.criarBotao(self.janela, self.opcoes[1], lambda: self.deslocar(1),
Estilos.BRANCO, Estilos.PRETO)

        self.botao3 = Elementos.criarBotao(self.janela, self.opcoes[2], lambda: self.deslocar(2),
Estilos.AZUL, Estilos.BRANCO)

        self.botaoV = Elementos.criarBotao(self.janela, "Voltar", self.janela.jogo.voltar,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
```

```
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões
        Elementos.posicionarBotao(self.botao1)
        Elementos.posicionarBotao(self.botao2)
        Elementos.posicionarBotao(self.botao3)
        Elementos.posicionarBotao(self.botaoV)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe InfoSemEscolha:

```
from View.Display7.Constantes import Estilos, Elementos
class InfoSemEscolha:
    """
    Esta classe mostra a informação de opções sem escolha como Destinos
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
        :param titulo: O título que precisa mostrar
        :param info: O texto que precisa mostrar
        """
        self.janela = janela
        self.info = []

        # Variável de controle
        self.on = False

        # Criando e definindo os botões
        self.criandoBotoes()
        self.definindoBotoes()
        Elementos.apagarElementoDaTela([self.botaoV, self.botaoS, self.botaoD]) # Vamos
rapidamente apagar os botões para que eles não apareçam
```

```
    def atualizarInfo(self, titulo, info):
        """
        Atualiza a informação das variáveis
        :param titulo:
        :param texto:
        :param locais:
        :return:
        """
        self.tituloTexto = titulo
        self.info.clear()
        for opcao in info:
            self.info.append(opcao)

    def mostrarInstrucoes(self):
        """
        Mostra as informações
        :return:
        """
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, self.tituloTexto)
        self.info1 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[0])
        self.info2 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[1])
        self.info3 = Elementos.criarTexto(self.janela, self.janela.frameJogo, self.info[2])

        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info1)
        Elementos.posicionarTexto(self.info2)
        Elementos.posicionarTexto(self.info3)




    def limparTudo(self):
        """
        Apaga os frames com informação e botões
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info1, self.info2, self.info3,
self.botaoV, self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self, titulo, info):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
```

```
        :return:
        """
        self.on = True
        self.atualizarInfo(titulo, info)
        self.mostrarInstrucoes()
        self.criandoBotoes()
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))


    def definindoBotoes(self):
        """
        Definindo os botões a serem escolhidos
        """
        self.possibilidades = ["v", "s", "d"]
        self.botoes = {
            "v": self.botaoV,
            "s": self.botaoS,
            "d": self.botaoD,

        }
    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """

        # Criando os botões
        # Criando os botões

        self.botaoV = Elementos.criarBotao(self.janela, "Voltar", self.janela.jogo.voltar,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões


        Elementos.posicionarBotao(self.botaoV)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe Inicio:

```
from View.Display7.Constantes import Estilos, Elementos
class Inicio:
    """
    Esta classe mostra a informação inicial
    """
    def __init__(self, janela):
        """
        Para as opções de visualização que terão somente os botões de Voltar, Novo Jogo e
Desligar como por exemplo "Mostrar Destinos"
        :param janela: A janela do app
        :param titulo: O título que precisa mostrar
        :param info: O texto que precisa mostrar
        """
        self.janela = janela

        # Variável de controle
        self.on = False

        # Criando e definindo os botões
        self.criandoBotoes()
        self.definindoBotoes()
        Elementos.apagarElementoDaTela([self.botaoJ, self.botaoS, self.botaoD]) # Vamos
rapidamente apagar os botões para que eles não apareçam




    def mostrarInstrucoes(self):
        """
        Mostra as informações
        :return:
        """
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo,
self.janela.jogo.jogo.cidadeInicial.tituloCaso)
        self.caso = Elementos.criarTexto(self.janela, self.janela.frameJogo,
self.janela.jogo.jogo.cidadeInicial.caso)
        self.tarefa = Elementos.criarTexto(self.janela, self.janela.frameJogo,
                                           f"Sua tarefa: {self.janela.jogo.jogo.cidadeInicial.tarefaCaso}")
        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.caso)
        Elementos.posicionarTexto(self.tarefa)
```

```
    def limparTudo(self):
        """
        Apaga os frames com informação e botões
        :return:
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.caso, self.tarefa, self.botaoJ,
self.botaoS, self.botaoD])
            self.on = False


    def reiniciar(self):
        """
        Recoloca toda a informação nos seus frames
        :param titulo: título que precisa mostrar
        :param texto: texto que precisa mostrar
        :return:
        """
        self.on = True
        self.mostrarInstrucoes()
        self.criandoBotoes()
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))

    def definindoBotoes(self):
        """
        Definindo os botões a serem escolhidos
        """
        self.possibilidades = ["v", "s", "d"]
        self.botoes = {
            "v": self.botaoJ,
            "s": self.botaoS,
            "d": self.botaoD,

        }
    def criandoBotoes(self):
        """
        Cria os botões
        :return:
        """

        self.botaoJ = Elementos.criarBotao(self.janela, "Jogar", self.janela.jogar, Estilos.AZUL,
Estilos.BRANCO)
        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)
```

```
        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        Elementos.posicionarBotao(self.botaoJ)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Classe Menu:

```
from View.Display7.Constantes import Estilos, Elementos
class Menu:
    def __init__(self, janela):
        """
        Esta classe mostra o Menu Principal
        :param janela: A janela do app
        :param titulo: O título a ser mostrado
        :param texto:  O texto a ser mostrado
        """
        self.janela = janela
        # Variável de controle
        self.on = False

        # Criando e definindo os botões
        self.criandoBotoes()
        self.definindoBotoes()
        Elementos.apagarElementoDaTela([self.botao1, self.botao2, self.botao3, self.botaoS,
self.botaoD]) # Vamos rapidamente apagar os botões para que eles não apareçam


    def mostrarInstrucoes(self, titulo, texto):
        self.titulo = Elementos.criarTitulo(self.janela, self.janela.frameJogo, titulo)
        self.info = Elementos.criarTexto(self.janela, self.janela.frameJogo, texto)

        Elementos.posicionarTitulo(self.titulo)
        Elementos.posicionarTexto(self.info)




    def limparTudo(self):
        """
        Apaga toda a informação
        :return:
```

```
        """
        if self.on == True:
            Elementos.apagarElementoDaTela([self.titulo, self.info, self.botao1, self.botao2,
self.botao3, self.botaoS, self.botaoD])
            self.on = False



    def reiniciar(self, titulo, texto):
        self.limparTudo()
        self.mostrarInstrucoes(titulo, texto)
        self.criandoBotoes()
        self.on = True
        self.janela.app.after(Estilos.TEMPO, lambda: self.janela.jogo.escolha(self.possibilidades,
self.botoes))

    def definindoBotoes(self):
        """
        Definindo os botões a serem escolhidos
        """
        self.possibilidades = ["1", "2", "3", "s", "d"]
        self.botoes = {
            "1": self.botao1,
            "2": self.botao2,
            "3": self.botao3,
            "s": self.botaoS,
            "d": self.botaoD,

        }
    def criandoBotoes(self):

        # Criando os botões
        self.botao1 = Elementos.criarBotao(self.janela, "Investigar", self.janela.jogo.investigar,
Estilos.AMARELO, Estilos.PRETO)

        self.botao2 = Elementos.criarBotao(self.janela, "Viajar", self.janela.jogo.viajar,
Estilos.BRANCO, Estilos.PRETO)

        self.botao3 = Elementos.criarBotao(self.janela, "Destinos", self.janela.jogo.destinos,
Estilos.AZUL, Estilos.BRANCO)

        self.botaoS = Elementos.criarBotao(self.janela, "Novo Jogo", self.janela.jogo.novoJogo,
Estilos.PRETO, Estilos.BRANCO)

        self.botaoD = Elementos.criarBotao(self.janela, "Desligar", self.janela.app.destroy,
Estilos.VERMELHO, Estilos.BRANCO)

        # Posicionando os botões
        Elementos.posicionarBotao(self.botao1)
        Elementos.posicionarBotao(self.botao2)
```

```
        Elementos.posicionarBotao(self.botao3)
        Elementos.posicionarBotao(self.botaoS)
        Elementos.posicionarBotao(self.botaoD)
```

Pacote LinhaDeComando:

Classe LinhaDeComando:

```
from Controller.FluxoDeJogoLC import FluxoDeJogoLC
class JogoLC:
    """
    Esta classe que cuida de como será feita a visualização do jogo na Linha de comando
    """
    def __init__(self):
        self.jogo = FluxoDeJogoLC(self)
        self.jogo.jogar()



    def mostraNovoJogo(self, caso, tarefa, tempo):
        """
        Imprime a informação de um novo jogo
        :param caso:
        :param tarefa:
        :param tempo:
        :return:
        """
        print("\n/**********************/")
        print("Cadê a Zefa?")
        print("/**********************/\n")
        print("A notória ladra internacional de origem brasileira, Zefa da Silva, está foragida
desde 1983 quando ajudou seu pai a roubar a taça Jules Rimet com a tenra idade de 10 anos.
Desde então ela tem aplicado roubos cada vez mais ousados pelo Brasil e o mundo e a nossa
dedicada PF nunca deixou de persegui-la décadas afora. Será você a pessoa que finalmente irá
aprender está elusiva ladra? Boa sorte!")
        print("/**********************/\n")
        print(caso)
        print(tarefa)
        print(f"Você tem {tempo} horas para apreender a Zefa! Boa sorte!")
        print("/**********************/\n")
    def mostraInfo(self, infoTitulo, info):
        print("\n/**********************/")
        print(infoTitulo)
        print("/**********************/\n")
        print(info)

    def mostraOpcoes(self, titulo, opcoes):
        """
        Imprimime as opcões que podem ser escolhidas
        :return:
```

```
        """
        print("\n/**********************/")
        print(titulo)
        print("/**********************/\n")
        contador = 1
        for opcao in opcoes:
            print(f"{contador} - {opcao}")
            contador += 1

    def mostraGanhou(self, vitoria, pontuacao):
        """
        Imprime o texto de vitória e a pontuação
        :param vitoria:
        :param pontuacao:
        :return:
        """
        print("\n/**********************/")
        print(vitoria)
        print("\n/**********************/")
        print(f"YAY! Você ganhou! Parabéns! Sua pontuação foi de {pontuacao} pontos")
        print("/**********************/\n")

    def mostraPerdeu(self):
        print("\n/**********************/")
        print("Ah, que pena. Você extrapolou o tempo de jogo. Mas não se preocupe, você pode
tentar novamente com outro caso!")
        print("\n/**********************/")

    def mostraMenu(self):
        """
        Imprime as opções principais
        :return:
        """
        print("\n/**********************/")
        print("Digite a opção:\n1 - Investigar\n2 - Viajar\n3 - Ver Destinos\nE a qualquer
momento digite: \ns - Novo Jogo\nd - Desligar")
        print("\n/**********************/")

    def mostraInfoTranslado(self, tempo, destino):
        if tempo == 1:
            texto = "hora"
        else:
            texto = "horas"
        print(f"Você vai levar {tempo} {texto} para chegar a {destino}")
    def mostraTempoRestante(self, tempo, tempoJogo):
        if tempo == 1:
            texto = "hora"
        else:
            texto = "horas"
        print(f"Você acaba de viajar {tempo} {texto}. Você só tem {tempoJogo} horas para
```

```
achar a Zefa")

    def mostraDormir(self, tempoJogado):
        print("Está na hora de dormir. Bons sonhos!")
        for x in range(5):
            print("ZZZZZ...")
        if tempoJogado == 1:
            texto = "hora"
        else:
            texto = "horas"
        print(f"Bom dia. Você só tem {tempoJogado} {texto} para achar a Zefa. Vamos ver se
ainda dá tempo?")

    def temTempo(self, tempoJogado):
        if tempoJogado == 1:
            texto = "hora"
        else:
            texto = "horas"
        print(f"Ufa, você ainda tem {tempoJogado} {texto}. Corre lá.")

    def escolhaIvalida(self):
        print("Essa escolha não é válida. Favor entrar um dos valores válidos.")
```

Arquivo main.py:

```
from View.Computador.Computador import Computador
from View.Display7.Display7 import Display7
from View.LinhaDeComando.JogoLC import JogoLC

# Inicializando o Game desejado. Para tanto é só descomentar o desejado, e comentar os
outros
computador = Computador()
# display7 = Display7()
# linhaDeComando = JogoLC()
```