INSTITUTO DE DESENVOLVIMENTO EDUCACIONAL DO ALTO URUGUAI CENTRO UNIVERSITÁRIO - UNIDEAU

Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas - Nível: I

Projeto Interdisciplinar de Aperfeiçoamento Teórico e Prático

LEVANTAMENTO DE REQUISITOS E DESENVOLVIMENTO DE UM JOGO - FLAPPY MÁRCIO

JESUS, Jonathan de[1]
MIOTTO, Augusto[1]
SOUZA, Vinícius de[1]
BRAMBATTI, Marcel[2]
DE PARIS, Alaércio[2]

RESUMO
Este artigo apresenta uma abordagem prática para desenvolver o jogo Flappy Bird usando modelagem UML. Conhecido por sua jogabilidade simples e viciante, o Flappy Bird serve como exemplo para demonstrar como a UML pode ser aplicada desde a análise inicial até a implementação. Através de modelos UML, representamos personagens, cenários, mecânicas de jogo e fluxo de interação, facilitando a comunicação entre os membros da equipe e garantindo uma implementação eficiente e de alta qualidade.
Palavras-chave: Python, FlappyBird, Márcio, Game, Codificação.

ABSTRACT
This article presents a practical approach to developing the Flappy Bird game using UML modeling techniques. Known for its simple and addictive gameplay, Flappy Bird serves as an example to demonstrate how UML can be applied from initial analysis to implementation. Through UML models, we represent characters, scenarios, game mechanics, and interaction flow, facilitating communication among team members and ensuring efficient and high-quality implementation.
Keywords: Python, FlappyBird, Márcio, Game, Codification.

1 INTRODUÇÃO

Este artigo apresenta uma abordagem prática para o desenvolvimento do jogo Flappy Bird utilizando técnicas de modelagem UML. O Flappy Bird é conhecido por sua jogabilidade simples e altamente viciante. Exploramos como a UML pode ser aplicada desde as fases iniciais de análise até a implementação do jogo, fornecendo uma estrutura sólida para compreender requisitos, arquitetura e interações do sistema. Demonstramos como criar modelos UML para representar personagens, cenários, mecânicas de jogo e fluxo de interação, facilitando a comunicação entre os membros da equipe e garantindo uma implementação eficiente e de alta qualidade.

2. REFERENCIAL TEÓRICO

2.1 O que é software?

Software é um conjunto de instruções ou programas de computador que direcionam o funcionamento de um sistema de hardware, permitindo que ele execute tarefas específicas. Ele pode assumir diversas formas, como aplicativos, sistemas operacionais, drivers e firmware, e é fundamental para controlar, gerenciar e facilitar uma variedade de atividades em computadores e dispositivos eletrônicos. Segundo Matheus Bigogno Costa (2020), o software é uma coletânea de instruções ou dados, que definem como o sistema vai trabalhar.

Existem três categorias de software:

- Software de programação: são as ferramentas utilizadas pelo programador para criar novos softwares e programas, nessa categoria são utilizadas diversas linguagens de programação, como Java e Python.
- Software de sistema: são os programas com a função de criar a conexão entre o computador e o usuário, é a base na qual outros softwares vão funcionar,sendo a categoria mais importante.
- Software de aplicação: são aqueles utilizados para realizar tarefas específicas, como processamento de textos, edição de imagens, navegação na web e jogos.

2.2 Engenharia de Software

Engenharia de Software é a aplicação sistemática de abordagens e métodos de engenharia para o desenvolvimento de sistemas. Ela envolve a criação de processos, metodologias e ferramentas para gerenciar o desenvolvimento de software, desde a sua concepção até a sua entrega e manutenção. Além de ser essencial para garantir a qualidade do sistema, assegura a eficiência do desenvolvimento do software, a satisfação do cliente e a competitividade da empresa no mercado.

2.2.1 Engenharia de Requisitos

A Engenharia de Requisitos, segundo Roger Pressman (2016), é uma das primeiras atividades que devem ser realizadas no desenvolvimento de software. Tem por objetivo compreender as necessidades do cliente, identificar os requisitos do sistema e definir as funcionalidades que serão desenvolvidas. Para isso, é necessário utilizar técnicas e ferramentas que auxiliem na elicitação, análise e documentação dos requisitos, tais como entrevistas, questionários, prototipagem, entre outras. É essencial para o sucesso do projeto, pois garante

que o software desenvolvido atenda às necessidades do cliente e esteja alinhado aos seus objetivos.

2.2.1.1 Requisitos Funcionais

Requisitos funcionais, segundo a definição de Ian Sommerville (2011), são as especificações que as funcionalidades que um software deve possuir para atender às necessidades e objetivos do cliente. Esses requisitos descrevem as entradas, processamentos e saídas do sistema e podem ser apresentados em forma de casos de uso, diagramas de fluxo, especificações textuais, entre outras formas. Os requisitos funcionais são essenciais para o desenvolvimento de um software eficiente e eficaz, pois eles garantem que o sistema realize as tarefas necessárias de forma correta e adequada. A elicitação dos requisitos funcionais é feita por meio de técnicas como entrevistas com os usuários, questionários, prototipagem, entre outras. A análise e validação dos requisitos é feita para garantir que eles sejam completos, consistentes e corretos. Além disso, a gestão dos requisitos é realizada ao longo do ciclo de vida do software, garantindo que as necessidades do cliente sejam atendidas e que o software esteja em conformidade com as especificações e exigências do projeto.

2.2.1.2 Requisitos Não Funcionais

Requisitos não funcionais são as especificações que descrevem as características e atributos do sistema que não estão diretamente relacionados às funcionalidades, mas importantes para a qualidade, desempenho, usabilidade e segurança do software. Eles descrevem os aspectos técnicos, operacionais, legais e de negócios que o sistema deve possuir, e podem incluir requisitos de desempenho, segurança, usabilidade, manutenção, entre outros. Para elicitar esses requisitos, são utilizadas técnicas como entrevistas com especialistas, análise de documentos, brainstorming, entre outras. A análise e validação dos requisitos não funcionais são realizadas para garantir que eles sejam completos, consistentes e corretos. É importante destacar que os requisitos não funcionais devem ser considerados desde o início do processo de desenvolvimento do software, pois eles afetam diretamente a arquitetura, o design e a implementação do sistema. Dessa forma, é fundamental que os requisitos não funcionais sejam identificados, analisados e documentados de forma clara e precisa (SOMMERVILLE, 2011).

2.2.2 UML

UML (*Unified Modeling Language*) é uma linguagem de modelagem visual, desenvolvida por um consórcio de empresas de software liderado pela Rational Software Corporation, que é usada para representar e projetar softwares. Ela fornece uma notação gráfica padronizada para descrever as várias perspectivas de um sistema, incluindo sua estrutura, comportamento e interações com outros sistemas. Por ser uma linguagem de modelagem muito poderosa e flexível, a UML permite que os desenvolvedores de software capturem uma ampla gama de conceitos e relacionamentos em seus modelos. Ela é amplamente usada em metodologias de desenvolvimento de software como o RUP (Rational Unified Process) e o Agile. Além disso, muitas ferramentas de modelagem de software, como o Visual Paradigm, o IBM Rational Rose e o Enterprise Architect, suportam a UML (FOWLER, 2003).

2.2.2.1 Casos de Uso

> Os diagramas de caso de uso permitem a visualização das interações que um usuário ou cliente pode ter com um sistema. Um diagrama de caso de uso deve permitir a visualização de uma razão (caso de uso) pela qual um indivíduo (ator) interagiu com sua organização (sistema) e os relacionamentos entre o negócio e os indivíduos (GASKIN, 2022).

Os elementos de um diagrama de casos de uso são atores, casos de uso e relacionamentos entre eles. Os atores representam usuários externos, os casos de uso representam funcionalidades do sistema e os relacionamentos mostram como eles interagem. Esse diagrama serve para visualizar e descrever as interações entre os usuários e o sistema, ajudando a entender os requisitos e funcionalidades do sistema de forma clara e concisa.

2.2.2.2 Diagrama de Atividades

De acordo com ANG (2023), um diagrama de atividades ou diagrama de atividades UML:

> "[...] ilustra o fluxo ou sequência de ações que são realizadas em um sistema. Os diagramas de atividades UML se enquadram nos diagramas de comportamento porque modelam como um sistema se comporta quando as ações são executadas para concluir uma atividade ou processo. Um diagrama de atividades também pode ser usado para apresentar o fluxo de eventos e identificar os requisitos em um processo de negócios." (ANG, 2023).

Os elementos que compõem um diagrama de atividades são:

- Atividades: Representam as ações ou etapas realizadas no processo. São representadas por retângulos com o nome da atividade.

- Transições: Indicam a ordem das atividades e como o controle passa de uma atividade para outra. São representadas por setas.
- Decisões: Permitem que o fluxo do processo siga diferentes caminhos com base em condições. São representadas por losangos.
- Bifurcações (fork) e Junções (join): Permitem dividir ou unir o fluxo do processo em múltiplos caminhos. São representadas por barras horizontais e verticais, respectivamente.
- Ponto de Início e Fim: Indicam o início e o término do processo. Geralmente são representados por círculos.

2.3 Linguagem de Programação

Segundo o autor Robert W. Sebesta (2018), uma linguagem de programação pode ser definida como "um conjunto de regras, símbolos e convenções usados para escrever programas de computador". Ele explica que a linguagem de programação permite que um programador descreva uma sequência de instruções que um computador pode executar para realizar uma tarefa. As linguagens de programação são criadas para serem usadas em diferentes contextos, como aplicações de negócios, sistemas embarcados, inteligência artificial, jogos, entre outros. Cada linguagem de programação tem suas próprias regras de sintaxe e semântica, que definem como as instruções devem ser escritas e interpretadas pelo computador.

Existem várias gerações de linguagens de programação, desde as primeiras linguagens de programação de baixo nível, como a linguagem Assembly, até as linguagens de programação de alto nível, como Java, Python, Ruby e C++. Cada geração de linguagem de programação tem suas próprias características e vantagens, e a escolha de uma linguagem de programação depende do contexto e dos requisitos do projeto a ser desenvolvido.

2.5.1 O que é compilação e interpretação?

A compilação refere-se à coleta e organização de dados brutos ou informações relevantes para um estudo específico. Geralmente é a primeira etapa de um projeto de pesquisa. Envolve reunir todos os dados disponíveis que podem ser relevantes para a investigação em questão. Após a compilação, os dados são organizados, revisados e preparados para análise.

A interpretação refere-se à análise e compreensão dos dados compilados, após a coleta e organização dos dados, os pesquisadores buscam entender o significado subjacente aos dados, identificar padrões, relações e tendências.

2.5.1.1 Linguagens compiladas

As linguagens compiladas são convertidas diretamente na máquina em um código de máquina que o processador pode executar. Como resultado, elas tendem a ser mais rápidas e mais eficientes em sua execução do que as linguagens interpretadas. Elas também dão ao desenvolvedor mais controle sobre alguns aspectos do hardware, como o gerenciamento da memória e o uso da CPU.

As linguagens compiladas necessitam de uma etapa de "build" (montagem) – elas precisam, primeiramente, ser compiladas manualmente. Você precisa "remontar" o programa sempre que precisar fazer uma alteração. Em nosso exemplo do molho, toda a tradução já está escrita antes de chegar até você. Se o autor original decidir usar um tipo diferente de óleo de oliva, a receita inteira precisaria ser traduzida novamente e reenviada a você.

Exemplos de linguagens compiladas puras são o C, o C++, o Erlang, o Haskell, o Rust e o Go. (LINGUAGENS… . 2021)

2.5.1.2 Linguagens interpretadas

Os interpretadores passam por um programa linha por linha e executam cada comando. Aqui, se o autor decidir que quer usar um tipo diferente de óleo de oliva, só precisaria remover o antigo e adicionar o novo. Seu amigo tradutor poderia informar isso a você quando a mudança acontecesse.

Linguagens interpretadas, antigamente, eram significativamente mais lentas do que as linguagens compiladas. Porém, com o desenvolvimento da compilação just-in-time, essa distância vem diminuindo.

Exemplos de linguagens interpretadas comuns são o PHP, Ruby, Python e o JavaScript. (LINGUAGENS… . 2021)

2.5.2 Linguagem Python

Criada por Guido van Rossum em 1989, Python é uma linguagem de programação interpretada, de alto nível, multiplataforma, dinâmica, orientada a objetos e funcional. A sintaxe limpa e simples de Python, tornam a linguagem fácil de aprender e usar, permitindo que os desenvolvedores criem programas complexos com poucas linhas de código. Python é uma

linguagem altamente versátil, com aplicações em várias áreas, desde desenvolvimento web e científico até automação de tarefas e inteligência artificial (LUTZ, 2013).

2.5.2.1 Bibliotecas Python

A linguagem Python tem uma grande quantidade de bibliotecas e pacotes produzidos em campos como visualização de dados, machine learning, processamento de linguagem natural, análise complexa de dados, cálculos matemáticos, web scraping e muito mais.

As bibliotecas no Python são conjuntos de código pré-escrito que nos permitem realizar diversas tarefas sem escrever o código do zero, são componentes essenciais para o desenvolvimento de projetos em Python. Elas permitem expandir as capacidades da linguagem, resolver problemas complexos, e com isso, acelerar o processo de desenvolvimento.

3.2.2 PyGame

Pygame é uma biblioteca em Python voltada para o desenvolvimento de jogos. Ela fornece ferramentas e funcionalidades que facilitam a criação e o gerenciamento de gráficos, sons, colisões, animações e outras características presentes nos jogos. Com o Pygame, é possível criar jogos 2D de forma simples e intuitiva.

Uma das principais vantagens do Pygame é a sua facilidade de uso. Mesmo para aqueles que estão começando a programar, é possível desenvolver jogos de forma rápida e eficiente. Além disso, o Pygame é uma biblioteca de código aberto e possui uma comunidade ativa, o que significa que há uma abundância de recursos, tutoriais e exemplos disponíveis para ajudar os desenvolvedores iniciantes.

2.4 O Jogo

O jogo “Flappy Márcio”, é uma adaptação do Flappy Bird, onde os jogadores assumem o controle de um personagem enquanto ele avança por um ambiente cheio de obstáculos. A mecânica central do jogo envolve pressionar uma tecla para fazer o personagem voar para cima e soltar para fazê-lo descer, tudo isso enquanto tenta desviar dos obstáculos ao longo do caminho, como tubos e outros objetos. O objetivo é alcançar a maior distância possível, acumulando pontos conforme avança. Será contabilizada uma pontuação, referente a quantidade de obstáculos desviados, caso o personagem colida com o mesmo ou simplesmente caia do céu, o jogador terá a oportunidade de recomeçar.

3. METODOLOGIA

A metodologia para criar esse jogo envolve uma série de etapas, cada uma contribuindo para o desenvolvimento do projeto.

3.1 Objetivo Geral

Realizar levantamento de requisitos e o desenvolvimento de um jogo estilo flappy bird em modo console ou gráfica.

3.2 Objetivos Específicos

- Desenvolver habilidade de interpretação e resolução de problemas computacionais, empregando recursos lógicos e/ou matemáticos;
- Capacitar o aluno em especificação de software através da gerência de requisitos de software.
- Permitir o envolvimento dos alunos e professores em um projeto único, buscando também, integração dentro da sala de aula;
- Capacitar o aluno na percepção e aplicabilidade dos conceitos adquiridos em situações do mercado profissional;
- Proporcionar ambiente para os alunos desenvolverem aplicações e analisar alternativas de construção de soluções aplicáveis às necessidades demandadas;
- Aprender sobre a biblioteca *Pygame;*
- Aprofundar conhecimentos sobre a linguagem *Pyhon.*

3.3 Procedimentos Metodológicos

O primeiro passo é definir o conceito do jogo. Isso envolve decidir sobre o gênero do jogo, como ação, aventura, quebra-cabeça, entre outros. Além disso, é importante estabelecer a história, os personagens principais e as mecânicas de jogo que serão incorporadas.

Após definir o conceito do jogo, é necessário configurar um ambiente de desenvolvimento adequado. Isso inclui instalar o Python em seu sistema e instalar a biblioteca Pygame. Antes de começar a escrever o código, é importante planejar a arquitetura do jogo. Isso inclui a definição da estrutura do jogo, como classes para personagens, cenários, itens e

elementos de interface do usuário. Também é necessário estabelecer os diferentes estados do jogo, como tela de menu, jogo principal e tela de pontuação, e como os jogadores irão transitar entre eles

Com a arquitetura do jogo planejada, é hora de começar a implementar o código. Utilizando a biblioteca Pygame é possível criar e manipular elementos gráficos, sons e eventos de entrada. O código deve ser organizado de forma modular, com ênfase na legibilidade e manutenção.

4. DESENVOLVIMENTO

O desenvolvimento do jogo em Python foi possível com o uso do Pygame, seguindo uma abordagem estruturada, começando com a definição clara dos objetivos e conceitos do jogo. Isso inclui a elaboração do enredo, a definição dos personagens, ambientes e mecânicas de jogo, bem como a identificação dos recursos gráficos e sonoros.

Se fará uso da biblioteca Pygame, que desempenha um papel central no desenvolvimento do jogo, fornecendo uma ampla gama de funcionalidades essenciais, como manipulação de gráficos, detecção de colisões, processamento de entrada do usuário e reprodução de áudio.

4.1 Cenário

O cenário do programa será um aplicativo baseado no jogo Flappy Bird. O jogo tem como personagem principal o Márcio (figura artística da cidade de Erechim -RS), o mesmo irá substituir o pássaro. O intuito do jogo é desafiar o usuário a passar por obstáculos e permanecer vivo pelo maior tempo possível.

4.2 Requisitos

4.2.1 Requisitos Funcionais

Quadro 1. Requisito Funcional: Interface para inicialização do jogo

RF1	**Interface para inicialização do jogo**
Descrição	O sistema deve possuir uma interface para que seja possível dar início ao jogo.

Prioridade	Essencial
Atores	usuário
Fluxo Principal	O jogador abre a interface de inicialização do jogo. O jogador visualiza as instruções na interface. O jogador pressiona o botão iniciar jogo. o jogo é iniciado.
Pré - Condições	O jogador tem acesso à interface de inicialização do jogo.
Pós - Condições	O jogo é iniciado e o jogador é levado a tela inicial do jogo.

Fonte: Autores. 2024

Quadro 2. Requisito Funcional: criação de um cenário

RF2	**Criação de um cenário**
Descrição	O sistema deve permitir a criação e personalização de um cenário para o jogo
Prioridade	essencial
Atores	usuário
Fluxo Principal	O jogador abre o jogo O jogo exibe a tela de inicialização O jogador lê as instruções na tela de inicialização O jogador inicia o jogo O jogador é direcionado a tela inicial do jogo
Pré - Condições	O jogador possui o jogo instalado em seu dispositivo

Fonte: Autores, 2024.

Quadro 3. Requisito Funcional: Controle com apenas um clique

RF3	**Controle com apenas um clique.**
Descrição	O sistema deve permitir que o jogador controle o movimento do personagem principal por meio de um único toque na tela ou clique do mouse.
Prioridade	Alta.
Atores	Usuário.

Fluxo Principal	O jogador acessa o menu principal. O jogador começa o jogo. O jogador precisa dar cliques contínuos para que o personagem continue voando.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário deve ter iniciado o jogo

Fonte: Autores, 2024.

Quadro 4. Requisito Funcional: Movimento Vertical

RF4	**Movimento vertical.**
Descrição	O personagem principal deve ser capaz de voar para cima quando o jogador toca na tela ou clica no mouse, e cair devido à gravidade quando não há entrada do jogador.
Prioridade	Alta.
Atores	Usuário.
Fluxo Principal	O jogador acessa o menu principal. O jogador começa o jogo. O jogador realiza o clique em um botão e o personagem voa.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário inicia o jogo.

Fonte: Autores, 2024.

Quadro 5. Requisito Funcional: Obstáculos

RF5	**Obstáculos.**
Descrição	O jogo deve apresentar obstáculos na forma de canos verticais, posicionados aleatoriamente na tela. Esses obstáculos devem ser espaçados de forma que o jogador precise passar pelo espaço entre eles para continuar no jogo.
Prioridade	Alta.
Atores	Usuário.
Fluxo Principal	O usuário acessa o jogo. O usuário inicia o jogo. O usuário realiza cliques para o personagem voar.

	O usuário tenta não colidir com obstáculos.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário desvia de algum obstáculo.

Fonte: Autores, 2024.

Quadro 6. Requisito Funcional: Pontuação

RF6	**Pontuação**
Descrição	Deve haver um sistema de pontuação que aumenta à medida que o jogador avança no jogo, baseado na quantidade de obstáculos que o jogador passa com sucesso.
Prioridade	Alta.
Atores	Usuário.
Fluxo Principal	O usuário acessa o jogo. O usuário inicia o jogo. O usuário passa pelos obstáculos e tem seus pontos contabilizados.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário deve ter passado por algum obstáculo.

Quadro 7. Requisito Funcional: Colisões

RF7	**Colisões**
Descrição	O jogo deve detectar colisões entre o personagem principal e os obstáculos. Se o personagem principal colidir com um obstáculo, o jogo deve terminar e mostrar a pontuação final do jogador.
Prioridade	Alta.
Atores	Usuário.
Fluxo Principal	O usuário deve acessar o jogo. O usuário deve iniciar o jogo. O usuário deve colidir com um dos obstáculos.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário não consegue desviar de um obstáculo.

Fonte: Autores, 2024.

Quadro 8. Requisito Funcional: Reinício do jogo

RF8	**Reinício do jogo.**
Descrição	Após uma colisão ou fim do jogo, o jogador deve ter a opção de reiniciar o jogo, começando novamente do início.
Prioridade	Alta.
Atores	Usuário.
Fluxo Principal	O usuário acessa o jogo. O usuário inicia o jogo. O usuário colide com algum obstáculo. O usuário tem a opção de reiniciar a partida.
Pré-Condições	O usuário deve ter acesso ao jogo.
Pós-Condições	O usuário deve colidir com algum objeto.

Fonte: Autores, 2024.

Quadro 9. Requisito Funcional: Gráfico e sons

RF9	**Gráfico e sons.**
Descrição	O jogo deve ter gráficos simples, porém atraentes, e efeitos sonoros que correspondam às ações do jogador e aos eventos do jogo, como colisões e a obtenção de pontos.
Prioridade	Importante.
Atores	Usuário.
Fluxo Principal	O desenvolvedor seleciona os elementos gráficos a serem adicionados no jogo, como imagens, animações, etc O desenvolvedor integra os gráficos aos códigos do jogo O desenvolvedor seleciona os sons a serem adicionados ao jogo, como efeitos sonoros, trilhas sonoras etc. O desenvolvedor integra os sons ao código do jogo.
Pré - Condições	O jogo está em desenvolvimento. Os elementos gráficos e sonoros estão disponíveis e prontos para serem integrados ao jogo.

Pós - Condições	Os gráficos e sons são adicionados ao jogo. Os elementos gráficos e sonoros são reproduzidos conforme necessário durante o jogo, proporcionando uma experiência mais imersiva para o jogador.

Fonte: Autores, 2024.

4.2.2 Requisitos Não Funcionais

Quadro 10. Requisito Não Funcional: Desempenho

RNF1	**Desempenho**
Descrição	O jogo deve ter um desempenho suave e responsivo, desde dispositivos de baixa potência até dispositivos mais poderosos. O tempo de resposta aos cliques do mouse deve ser mínimo para garantir uma experiência de jogo fluida.

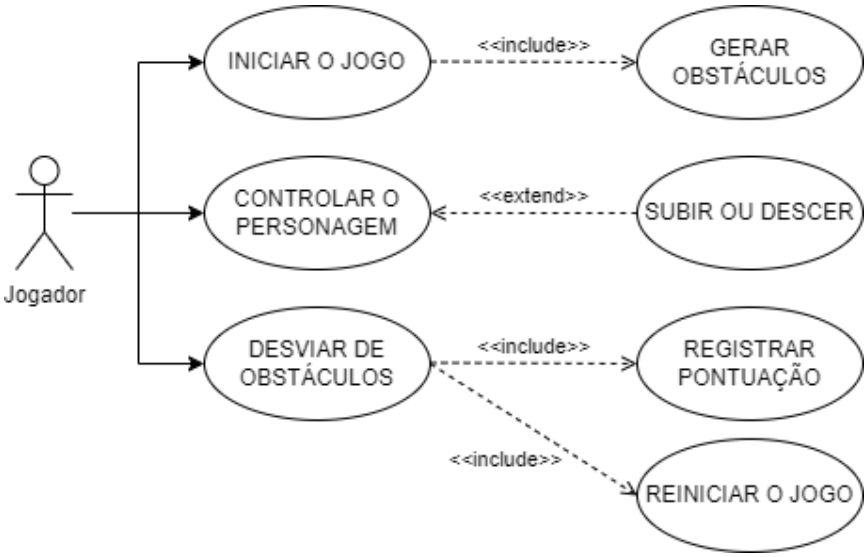
Fonte: Autores, 2024.

Quadro 13. Requisito Não Funcional: Usabilidade

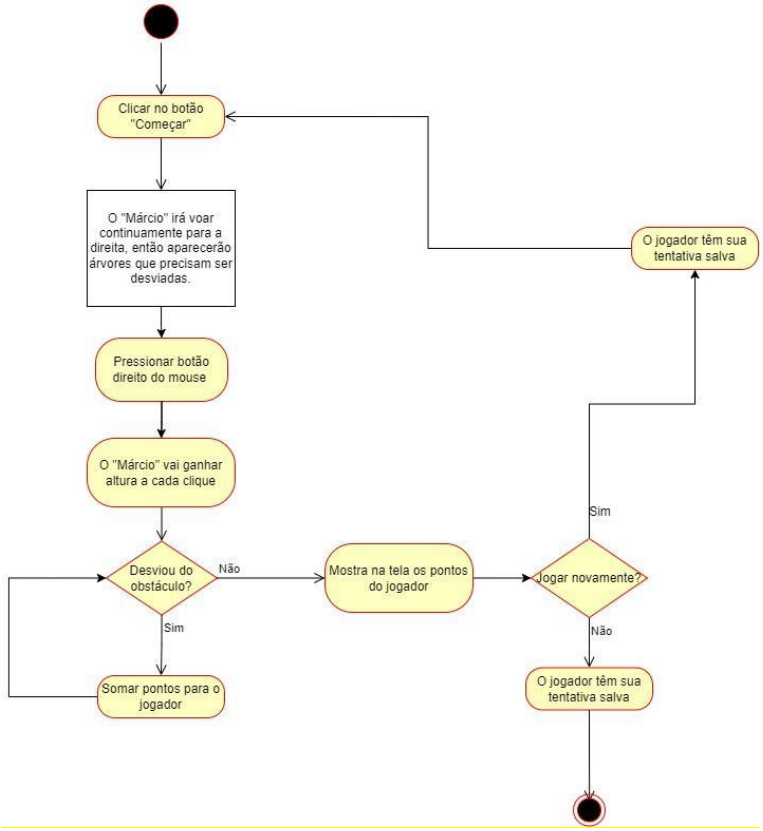
RNF2	**Usabilidade**
Descrição	A interface do usuário deve ser intuitiva e fácil de entender, garantindo que os jogadores possam começar a jogar sem a necessidade de instruções detalhadas.

Fonte: Autores, 2024.

4.3 Diagrama de Casos de Uso



4.4 Diagrama de Atividades UML



4.5 Projeto Visual

4.5.1 Tela Inicial

Modelo utilizado para a tela de inicialização e apresentação do jogo.

4.5.1 Tela de Jogo

Cena retirada de uma partida em andamento no próprio jogo.

4.6 Codificação

4.6.1 Criação Cenário / Personagem

```
#CARREGAR AS IMAGENS ADICIONADAS
IMAGEM_BACKGROUND = pygame.transform.scale(pygame.image.load(os.path.join('imgs', 'bg.png')),
                                           size: (TELA_LARGURA, TELA_ALTURA))
IMAGEM_CANO = pygame.transform.scale2x(pygame.image.load(os.path.join('imgs', 'pipe.png')))
IMAGEM_CHAO = pygame.transform.scale2x(pygame.image.load(os.path.join('imgs', 'base.png')))
IMAGEM_TELA_INICIAL = pygame.transform.scale(pygame.image.load(os.path.join('imgs', 'menu.png')),
                                             size: (TELA_LARGURA, TELA_ALTURA))
IMAGENS_PASSARO = [
    pygame.transform.scale2x(pygame.image.load(os.path.join('imgs', 'bird1.png'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('imgs', 'bird2.png'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('imgs', 'bird3.png'))),
]
IMAGENS_MARCIO = [
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-01.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-02.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-03.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-04.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-05.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-07.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-08.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-09.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-10.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-11.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-12.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-13.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-14.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-15.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-16.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-17.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-18.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-19.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-20.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-21.gif'))),
    pygame.transform.scale2x(pygame.image.load(os.path.join('marcioimg', 'frame-22.gif'))),
```

Codificação de carregamento das imagens adicionadas ao projeto.

4.6.2 Colisão Com Obstáculos

```
def colidir(self, passaro):
    passaro_mask = passaro.get_mask()
    topo_mask = pygame.mask.from_surface(self.CANO_TOPO)
    base_mask = pygame.mask.from_surface(self.CANO_BASE)

    distancia_topo = (self.x - passaro.x, self.pos_topo - round(passaro.y))
    distancia_base = (self.x - passaro.x, self.pos_base - round(passaro.y))

    topo_ponto = passaro_mask.overlap(topo_mask, distancia_topo)
    base_ponto = passaro_mask.overlap(base_mask, distancia_base)

    return base_ponto or topo_ponto
```

Essa codificação representa a colisão do personagem com o obstáculo do cenário.

5. CONSIDERAÇÕES FINAIS

O desenvolvimento do jogo Flappy Bird utilizando técnicas de modelagem UML demonstrou ser uma abordagem eficaz para garantir uma estrutura organizada e compreensível. A aplicação da UML desde as fases iniciais de análise até a implementação permitiu uma visão clara dos requisitos, da arquitetura e das interações do sistema, facilitando a comunicação entre os membros da equipe e assegurando uma implementação eficiente e de alta qualidade.

Através dos diagramas UML, foi possível representar de maneira clara os personagens, cenários, mecânicas de jogo e fluxo de interação, o que contribuiu significativamente para a organização do projeto e a antecipação de possíveis desafios. A utilização da biblioteca Pygame em Python proporcionou uma plataforma robusta e acessível para o desenvolvimento do jogo,

evidenciando a versatilidade e a potência dessa linguagem para projetos de desenvolvimento de jogos.

Além de cumprir os objetivos técnicos e educacionais, este projeto também proporcionou uma excelente oportunidade de aprendizado colaborativo, integrando conceitos teóricos com a prática de desenvolvimento de software. A experiência adquirida com a especificação de software, a gestão de requisitos e a implementação prática contribuiu para o aprimoramento das habilidades dos participantes, preparando-os para desafios futuros no mercado profissional.

Em suma, o uso de UML e Pygame no desenvolvimento do Flappy Bird não só resultou em um produto final funcional e divertido, mas também destacou a importância de uma abordagem estruturada e colaborativa no desenvolvimento de software. Este projeto serviu como um exemplo valioso de como metodologias de engenharia de software podem ser aplicadas de forma eficaz para criar aplicações de alta qualidade.

6. REFERÊNCIAS

ANG, Joan. Como criar um diagrama de atividades [+ exemplos]. 2023. Disponível em: https://pt.venngage.com/blog/diagrama-de-atividades/ . Acesso em: 25 abr. 2024.

COSTA, Matheus Bigogno. O que é software?. 2020. Disponível em: https://canaltech.com.br/software/o-que-e-software/ . Acesso em: 24 abr. 2024.

DEVMEDIA. UML **e os diagramas de casos de uso, sequência e atividade**. 2016. Disponível em: https://www.devmedia.com.br/uml-e-os-diagramas-de-casos-de-uso-sequencia-e-atividade/ . Acesso em: 14 mar. 2024.

FOWLER, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language. ed Addison-Wesley Professional, 2003.

GASKIN, Jennifer. Tudo o que você precisa saber sobre diagrama de caso de uso. 2022. Disponível em: https://pt.venngage.com/blog/diagrama-de-caso-de-uso/ . Acesso em: 25 abr. 2024.

ROSA, Daniel. Linguagens de programação interpretadas x compiladas: qual é a diferença?. 2021. Disponível em: <https://www.freecodecamp.org/portuguese/news/linguagens-de-programacao-interpretadas-x-compiladas-qual-e-a-diferenca/>. Acesso em: 25 abr. 2024.

LUTZ, M. Learning Python. 5 ed. O'Reilly Media, 2013.

SOMMERVILLE, Ian. Engenharia de software. 9 ed. São Paulo: Pearson, 2011.