UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA

DCET - DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL NEVES BRASILEIRO COSTA SILVEIRA

META-JOGO PARA APRENDIZAGEM E EXPERIMENTAÇÃO DE CONTROLES DE MOVIMENTAÇÃO EM JOGOS DE PLATAFORMA 2D COM USO DO MÉTODO PBL E MODELO MDA

Vitória da Conquista- BA

2025

DANIEL NEVES BRASILEIRO COSTA SILVEIRA

META-JOGO PARA APRENDIZAGEM E EXPERIMENTAÇÃO DE CONTROLES DE MOVIMENTAÇÃO EM JOGOS DE PLATAFORMA 2D COM USO DO MÉTODO PBL E MODELO MDA

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Sudoeste da Bahia, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Gidevaldo Novais dos Santos

Vitória da Conquista- BA

2025

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O gênero de plataforma 2D é uma referência icônica dentro da área de jogos. Por conta disso, uma variedade de sistemas, mecânicas e implementações foram criadas para criar jogos com diferentes mensagens e objetivos. Porém, a quantidade de informação sobre o tema pode ser assustadora para programadores que planejam desenvolver um jogo nesses moldes. Portanto, esse artigo apresenta e projeta um jogo que servirá de ferramenta para ensinar e explorar as principais mecânicas de movimentação de personagem encontradas no gênero de jogos de plataforma e avaliará sua performance em teste para esse objetivo.

Palavras-chave: Aprendizagem; Jogo; Método PBL; Pesquisa-aplicação; Plataforma 2D.

ABSTRACT

The 2D platformer genre is an iconic reference within the game industry. Because of this, a variety of systems, mechanics, and implementations have been created to make games with different messages and objectives. However, the amount of information on the subject can be overwhelming for programmers who plan to develop a game in this style. Therefore, this article presents and projects a game that will serve as a tool to teach and explore the main character movement mechanics found in the platformer game genre and will evaluate its performance in testing for this purpose.

Keywords: *2D platformer; Design Research; Game; Learning; PBL method.*

1 INTRODUÇÃO

O cenário de desenvolvimento de jogos 2D passa por um período de crescimento contínuo, incentivando a crescente demanda por jogos que ofereçam não apenas narrativas envolventes, mas também mecânicas de interação com o personagem e o mundo responsivas e cativantes.

Os jogos de plataforma 2D, em particular, ocupam um lugar especial nesse cenário, sendo um dos principais gêneros da área. Contudo, o design desses jogos é complexo e a implementação de mecânicas de movimento responsivas é um desafio que muitos desenvolvedores enfrentam. O controle de movimento de agentes e interação com o ambiente ditam a experiência do jogador com o mundo apresentado. Portanto, a construção de controles que consigam traduzir o objetivo do jogo é fundamental para criar projetos de alta qualidade.

A complexidade do design de jogos de plataforma 2D se reflete na variedade de abordagens adotadas por diferentes desenvolvedores. Cada jogo apresenta uma proposta única e requer mecânicas de movimento construídas para refletir essa visão. No entanto, a falta de ferramentas específicas e acessíveis para auxiliar desenvolvedores na criação e iteração dessas mecânicas tem sido um obstáculo persistente no campo do desenvolvimento de jogos 2D.

É nesse contexto que este trabalho se insere, elaborando uma solução viável e necessária, em uma plataforma de aprendizado e testagem de diferentes variáveis envolvidas no desenvolvimento de mecânicas de movimento para jogos 2D. Essa plataforma será projetada para ser acessível a desenvolvedores de todos os níveis de experiência, desde iniciantes até veteranos da indústria.

É notório que a pesquisa necessária para compreender e coletar as diferentes mecânicas de movimento resulta em uma coleção de técnicas dispersas e, por vezes, conflitantes. Por este motivo, é importante apresentar um ambiente que reúna as principais mecânicas de movimento adotadas pelos jogos de plataforma 2D e permitir que o jogador interaja com as variáveis envolvidas para maior entendimento do *design* de cada movimento.

Esta plataforma interativa desafia o usuário a completar fases com foco em atributos específicos de cada componente e assim, à medida que avança pelo jogo, construir um personagem que consegue executar ações complexas, compreendendo cada aspecto que o faz comportar-se da maneira que foi configurado pelo jogador. Diante disso, é necessário fazer

uma pergunta: convém desenvolver uma ferramenta que permite o jogador explorar as diferentes mecânicas que constroi um personagem em um jogo de plataforma 2D?

1.1 OBJETIVO GERAL E ESPECÍFICO

1.1.1 Objetivo geral

Desenvolver um meta-jogo para auxiliar no desenvolvimento de jogos em plataforma 2D, com foco nas mecânicas de movimento do personagem.

1.1.2 Objetivos específicos

- Projetar e implementar uma interface intuitiva para a manipulação em tempo real das variáveis de movimentação de um personagem em um ambiente de plataforma 2D.
- Elaborar fases que apresentam conceitos e desafiam o jogador na resolução do problema à medida que avança no jogo.
- Testar e avaliar a eficácia do meta-jogo em facilitar o desenvolvimento de jogos 2D com mecânicas de movimento aprimoradas.

1.2 METODOLOGIA

Nesta seção, serão discutidas as abordagens metodológicas utilizadas para desenvolver e avaliar o produto resultante do projeto.

1.2.1 Método PBL

Neste projeto, o jogador terá acesso a uma variedade de controladores de variáveis para modificar os valores de diferentes aspectos do movimento do personagem que controla. O modo de interagir com o mundo são os controles de movimento e ação disponíveis a qualquer momento e as variáveis ajustáveis para modificar o protagonista.

Segundo William Watson (2012), essa abordagem estimula a exploração e a descoberta, levando o jogador a entender melhor os sistemas pelos quais o jogo funciona e conseguir fazer melhores associações futuras para resolver problemas mais complexos no futuro.

Segundo Savery (1996), o método *Problem Based Learning* (PBL) – ou aprendizado baseado em problemas – surgiu nos anos 50, inicialmente como metodologia pedagógica para

o ensino médico, adaptando-se posteriormente a diversas áreas do conhecimento. O modelo de Barrows (1986) destaca um processo de aprendizagem: os alunos, sem prévio conhecimento, identificam e analisam o problema, buscam soluções, estudam para resolvê-lo e, por fim, aplicam o aprendizado.

No contexto dos jogos, o jogador enfrenta desafios desconhecidos, passando por etapas de pesquisa, avaliação e aplicação de hipóteses para resolver os problemas apresentados. Com isso em mente, o jogador pode explorar diferentes soluções para resolver as fases apresentadas e superar os desafios. Nessa abordagem, onde não há um tutorial definido para ensinar ao jogador o que precisa ser feito para resolver um problema, descrita por L. Adães *et al.* em artigo para a Universidade do Porto (2015). Nesse artigo, os autores associam o processo de assemelhamento e resolução de desafios em jogos ao processo de PBL, onde o jogador é apresentado a um desafio com conhecimento restrito às ações que pode executar. Cabe, então, ao jogador entender o problema apresentado e utilizar as ferramentas que estão disponíveis (as ações que pode executar) para encontrar uma solução para o problema.

1.2.2 Modelo MDA

Em sua revista "*Understanding Gamification*", Bohyun Kim (2015) explica a aplicação do modelo *Mechanics, Dynamics and Aesthetics* (MDA) – ou Mecânicas Dinâmicas e Estéticas – no *design* de jogos citando o trabalho feito por Hunicke *et al.* (2004) para apresentar o modelo na construção da identidade de um jogo baseado em regras, sistemas e resposta emocional.

O conjunto de regras que imperam sobre o jogo são chamadas de mecânica e determinam o resultado de cada *input* do jogador. As dinâmicas do jogo são uma coleção de mecânicas que juntas invocam certa sensação no jogador, o sistema de dinâmicas do jogo deve corresponder com a estética (ou emoções) que se espera atingir com cada aspecto do jogo.

O modelo MDA, explorado por Hunicke *et al.* (2004), estrutura o *design* e entendimento de jogos ao conectar mecânicas à estética desejada. Define três conceitos: mecânicas (componentes do jogo), dinâmicas (comportamento em tempo real das mecânicas) e estética (resposta emocional do jogador). Para Hunicke, encarar o jogo como um artefato de

estudo é essencial, considerando que a principal atividade de um jogo é a interação do jogador. Essa visão orienta o desenvolvimento e análise, garantindo que as mecânicas do jogo sustentem a estética desejada, sem comprometer a interação do jogador com o cenário.

Por isso, é imperativo a construção e estudo de um conjunto de controles que atendam às necessidades estéticas de cada jogo de plataforma. Assim, cada jogo se difere em ambiente, emoções, dificuldade, mensagem, tom e até história (recursos estéticos) a partir dos controles e mecânicas definidos.

1.2.3 Metodologia de desenvolvimento

Por ser um jogo de plataforma 2D com grande influência de jogos clássicos do século XX, a arte do jogo irá adotar um estilo de desenho pixelado, com elementos animados *frame* a *frame* em baixa definição. Alguns elementos como partículas e elementos de luz podem sair do padrão, mas irão compor uma peça única e harmoniosa.

A criação desses elementos será feita por um *software* a parte da Godot, especializada no desenho e animação de peças pixeladas: o Aseprite. Nele, é possível desenhar e animar com eficiência e importar para o Godot os arquivos para implementação no jogo.

1.2.4 Metodologia de pesquisa

A pesquisa-aplicação em educação, como descrita por Plomp *et al.* (2018), é uma abordagem com objetivo de desenvolver soluções científicas aplicáveis a problemas complexos no contexto educacional, além de validar teorias relacionadas a processos e ambientes de aprendizagem. No caso deste estudo, a questão central foca no desenvolvimento de uma ferramenta interativa para que desenvolvedores de jogos, amadores ou experientes, possam explorar variáveis de controle de jogos de plataforma 2D e aprimorar os movimentos dos protagonistas.

A pesquisa-aplicação é dividida em estudos que visam o desenvolvimento e validação de protótipos, destacando-se os estudos de desenvolvimento e validação. No desenvolvimento, intervêm pesquisadores e colaboradores que, a partir de princípios de design, criam soluções inovadoras para o contexto em questão. Já a validação pode ocorrer ao longo do processo, por meio de avaliações formativas e somativas para garantir, entre outros, a consistência e praticidade do produto final. Neste projeto, o foco está na avaliação desses

dois critérios, por meio de um formulário com um grupo-alvo após teste do produto. O grupo utilizará a ferramenta desenvolvida e, posteriormente, responderá a questionários para quantificar os aspectos da intervenção, utilizando uma escala de 1 (discordo totalmente) a 5 (concordo totalmente), garantindo uma análise qualitativa dos resultados obtidos.

2 FUNDAMENTOS TEÓRICOS E METODOLÓGICOS

Nesta seção, serão analisadas as referências teóricas e conceitos utilizados para fundamentar o *design* do jogo "No Controle" e as metodologias de pesquisa e ensino abordadas apresentados neste artigo.

2.1 FUNDAMENTOS TEÓRICOS

Essa seção irá analisar os conceitos necessários para entender o problema apresentado e contextualizar tópicos importantes para o desenvolvimento do jogo apresentado. Assim, os conceitos apresentados e os jogos estudados na seção são usados para desenhar as mecânicas desenvolvidas.

2.1.1 Jogos de plataforma 2D

Segundo ID Tech (2012), jogos de plataforma 2D são caracterizados por uma câmera afastada do personagem principal, geralmente lateral ao seu movimento (definindo o jogo como um *side scroller*). O personagem navega por um ambiente povoado de blocos e plataformas de onde ele pode pular e se mover para alcançar novos patamares.

É comum também os jogos apresentarem outras dinâmicas, como um sistema um ataques do personagem e inimigos espalhados pelas fases. Outros obstáculos também são recorrentes em jogos do estilo, como espinhos e seções meticulosas de movimentação. O mundo em que o personagem está inserido também pode apresentar movimento e ser interativo, como em plataformas móveis, botões a serem pressionados e uma combinação de portas trancadas e chaves escondidas.

A combinação de tantos sistemas constroi um mundo onde o jogador pode explorar e interagir por meio das mecânicas de movimento definidas pelo jogo. Ao passo que, por exemplo, um jogo como Mega man (1987) permite que o jogador destrua objetos no cenário com uma arma, outros jogos, como Celeste (2018), disponibilizam outras formas de conectar

com o mundo, com o uso de movimento rápido (*dash*) ou então, com um botão para interação com o cenário.

2.1.2 Movimentação do personagem em jogos de plataforma 2D

Segundo Hunicke *et al.* (2004), é imperativo para um jogo que a movimentação seja compatível com a proposta que o jogo evoca, sendo a principal mecânica do jogo. O modo como o personagem se movimenta pulando, correndo ou se esquivando ilustra ao jogador os desafios e dinâmicas que serão abordados no jogo.

Com a evolução dos jogos de plataforma durante os anos, essas mecânicas foram evoluindo, tornando a experiência do jogador mais responsiva e agradável. Controles são a interface mais significativa que o jogador tem para se expressar e interagir com o mundo construído pelo jogo, assim, quanto mais responsivos forem os controles do jogo, menos frustrado o jogador ficará e mais imerso no jogo.

2.1.3 Mecânicas comuns e implementações

- **Movimentação Lateral**

Como descrito por Andrade (2020), a movimentação lateral nos jogos de plataforma 2D, como os *side-scrollers*, é essencial para estabelecer a dinâmica central do jogo. Esse tipo de movimento não apenas define a progressão do jogador, mas também desempenha um papel crucial na atmosfera e na experiência do usuário. A forma como o personagem se desloca lateralmente pode transmitir uma ampla gama de sensações e emoções ao jogador: com a sensação de poder, velocidade e fluidez até evocar sentimentos de peso, rigidez ou instabilidade.

Existem várias abordagens para implementar a movimentação horizontal nos jogos de plataforma 2D. Uma das estratégias mais utilizadas envolve o incremento direto da posição do personagem no código. Essa solução, simples em execução, oferece um controle preciso sobre o movimento do personagem, exigindo apenas a atualização da posição do modelo no cenário.

Outra alternativa comum é a utilização de velocidade, como explorado no vídeo de Dawnosaur (2021), essa abordagem depende de um motor de física robusto para calcular colisões e vetores de velocidade do modelo. Essa abordagem acrescenta um nível de realismo ao movimento, tornando-o mais fluido e natural, especialmente durante interações físicas mais complexas. Além disso, a aplicação de forças nos modelos de personagem e cenário pelo

motor de simulação de física é uma terceira opção. Essa abordagem delega ao motor o cálculo detalhado das forças aplicadas, oferecendo uma representação mais realista do movimento.

- **Pulo**

A mecânica do pulo em jogos de plataforma é possivelmente a principal mecânica do gênero; ela é a chave para a maioria dos desafios clássicos enfrentados pelos jogadores. A evolução dessa mecânica trouxe mais responsividade dos controles e mais espaço para erros do jogador para entrada dos comandos.

Como explorado por Wagar (2015), o ajuste das variáveis do pulo podem mudar completamente o objetivo do jogo, um pulo com sensação mais flutuante pode passar a impressão ao jogador de um personagem mais impreciso, com desafios de plataforma menos punitivos. Já um pulo mais responsivo indica que o jogo tem seções de plataforma com pulos precisos e perigosos.

- **Ataque**

Diversos jogos dão ao jogador a ação de atacar com armas ou ferramentas dispostas ao personagem, geralmente com um toque de um botão de ação. Essa mecânica é tão comum em jogos, que subgêneros de plataforma foram criados graças à introdução da mecânica. Mega man é um exemplo de um plataforma 2D onde o protagonista tem um canhão que executa ataques a distância para derrotar inimigos e destruir objetos.

Há também os ataques corpo a corpo, podendo ser usados os punhos, espadas, machados, etc. Jogos com essa mecânica tendem a desviar para um sub gênero de ação, com mecânicas complexas e desafios de combate, um ótimo exemplo que apresenta isso como sua principal mecânica é Hollow Knight.

- **Movimentos Especiais**

Além de andar e pular, diversos jogos oferecem opções de movimento mais complexas ao jogador. Assim, o protagonista ganha mais mobilidade para deslocar em mapas mais complexos, maiores e mais verticais.

Um dos primeiros movimentos explorados foi o *dash* ou então *slide* (deslize) ou rolamento. Esse tipo de movimento apresenta implementações variadas, cada um com um propósito no jogo: o *dash* é um movimento de curto alcance em alta velocidade para alguma

direção, podendo ser executado no ar ou em terra; o deslize e rolamento geralmente são implementados para deslocar o protagonista para frente apenas quando o personagem está no chão e tornam o personagem invencível por alguns milésimos.

No vídeo feito por Dawnosaur (2022) é mostrado também a escala na parede (*wall climb* ou *wall jump*) com diferentes implementações a depender do objetivo do jogo com a mecânica. Jogos podem, como Celeste (2018), preferir que o jogador não abuse das escaladas a fim de incentivar a solução das fases de forma mais interessante. Porém, outros jogos como Hollow Knight e Mega man utilizam da mecânica como um meio de exploração de novas áreas do mapa e não limitam o uso da mecânica.

Ainda no vídeo feito por Dawnosur, é descrita outra mecânica muito comum: o pulo duplo, que pode ser mais explorada na seção de pulo, que permite o jogador executar um pulo extra para alcançar novas alturas ou percorrer mais distância antes de alcançar novamente o chão. Nota-se que há sempre novas opções de movimentação de jogos surgindo e sendo exploradas a todo momento. Portanto, o papel do projeto não é abordar todas as possibilidades nem todas as implementações existentes de uma mecânica.

2.1.4 Jogos notáveis

- **Super Mario (1985)**

Um jogo clássico que consolidou mecânicas importantes usadas até hoje na indústria. A série Mario conta com diversos títulos, em cada um deles, Mario e outros protagonistas apresentam diversas habilidades à disposição do jogador.

Nos jogos do Mario, o jogador pode andar, correr, pular, girar para evitar obstáculos, usar armas, habilidades e transformações que mudam a jogabilidade por um período de tempo. É com base nas principais características desse gênero que os controles do projeto deste artigo serão construídos.

- **Mega Man (1987)**

Mega man também é uma série de jogos com muitos títulos e muitos personagens jogáveis. Nesses jogos, são apresentadas diversas mecânicas, mas a que se destaca é o canhão de plasma (*Buster Gun*) que carrega em seu braço esquerdo. Com um botão de ação, o

jogador pode atirar com o canhão indefinidamente ou segurar para carregar um disparo poderoso.

O traje e a arma do Mega Man podem ser modificados, alterando os padrões de disparo e servem diferentes funções como derrotar inimigos e destruir o cenário. O jogador também dispõe de outras habilidades, como uma esquiva ou um deslize para movimentar-se mais rápido ou desviar de ataques inimigos; agarrar-se e pular de paredes, entre outras.

- **Sonic *the Hedgehog* (1991)**

Os jogos do Sonic contam com um estilo de *gameplay* (jogabilidade) diferente de seus antecessores, focando em uma movimentação rápida com muita inércia. Sendo assim, o personagem demora para ganhar velocidade uma vez parado, mas pode atingir altas velocidades e aproveitar o momentum acumulado para fazer manobras e explorar as fases.

- **Castlevania (1986)**

Jogo que foca em exploração e combate, com o personagem obtendo novas habilidades de movimento e formas de ataque com novas armas e feitiços. Esse jogo foi tão influente que, junto com Metroid (1986), fundou um novo gênero de jogos com foco em exploração e desbloquear novas habilidades. O gênero Metroidvania inspira diversos jogos, mesmo fora do gênero de *side scrollers* 2D.

- **DuckTales (1989)**

Por mais que seja um jogo simples na maioria das mecânicas de movimento, ele apresenta um aspecto na jogabilidade que uma infinidade de jogos posteriores irão utilizar. Pois, quando o jogador pressiona um botão enquanto está no ar, o Tio Patinhas – protagonista do jogo – estende sua bengala para baixo, permitindo-o quicar em objetos perigosos e inimigos. Esse aspecto na jogabilidade ficou conhecido como *Pogo Jump*, em referência ao brinquedo pula pula, e implementações variadas dessa mecânica se popularizaram em jogos de plataforma.

- ***Hollow Knight* (2017)**

Hollow Knight é o jogo com maior número de mecânicas relacionadas à movimentação em plataformas nesta lista. Nele, inicialmente, o jogador pode andar, pular e atacar com um curto ataque de espada. Porém, à medida que o jogo avança e novas

habilidades são desbloqueadas (herança da natureza Metroidvania), o jogador também pode esquivar (*dash*), escalar paredes, executar um pulo duplo, disparar feitiços de longo alcance e muito mais.

Com isso, o jogador ganha mais opções de movimento, facilitando a travessia no mapa e o auxiliando no combate. Por sua vez, o combate é também uma dinâmica essencial para o jogo, com uma coletânea de mecânicas que constroem encontros ágeis e arriscados com inimigos. Entre essas mecânicas estão: o *dash*, o ataque rápido e ataques carregados com a espada, feitiços e o *pogo jump* para evitar ataques.

- ***Super Meat Boy* (2010)**

Jogo desafiador de plataforma onde pulos precisos são muito importantes para passar das fases recheadas de armadilhas, exigindo tempo de reação rápido do jogador. Nesse jogo, o jogador toma controle de um personagem rápido, com um arco de pulo alto e bastante horizontal. O jogador também oferece ao jogador controle preciso dos movimentos do personagem enquanto está no ar, permitindo-o fazer manobras para evitar obstáculos e passar de fase.

- **Celeste (2018)**

Celeste é um jogo carregado de história e técnica. Nele, o jogador acompanha Madeline em uma escalada traiçoeira em uma montanha misteriosa e cheia de perigos. O jogador dispõe de nada além de mecânicas de jogos de plataforma bem construídas para desbravar esse mundo e chegar ao topo.

O pulo de Madeline é baixo e não cobre muita distância, porém, ela conta com uma habilidade especial: um *dash* que pode ser direcionado em 8 direções; além disso, ela também consegue escalar paredes por um curto período de tempo. O jogo conta com um polimento nos controles de pulo e *dash* para passar maior sensação de controle para o jogador. Celeste também utiliza forças para mover a personagem, possibilitando executar movimentos que conservam o momentum de pulos e dashes para cobrir distâncias maiores. As técnicas adotadas pelo jogo serão exploradas posteriormente no artigo.

- ***Platformer Toolkit* (2023)**

Um jogo desenvolvido por Mark Brown (2023) na plataforma Unity que apresenta um ambiente virtual para modificar as variáveis do personagem em um jogo de plataforma 2D. O projeto foca na exploração dos controles de movimento lateral, pulo e câmera e não tem foco na solução de problemas nem apresenta um desafio no qual o jogador deva se preocupar.

Por fim, se trata de uma experiência curta, apresentando controles básicos de movimentação e pulo, sem componentes mais complexos como controle de *dash*, ataque e movimentação por escala nas paredes.

2.2 FUNDAMENTOS METODOLÓGICOS

As referências para desenvolvimento desta aplicação estão no método PBL, descrito por Barrows (1986) e explorado por T. Duffy (1996, 2010) em outras áreas e também William Watson (2012) e L. Adães *et al.* (2015) já no contexto de desenvolvimento de jogos.

O modelo MDA é baseado no trabalho feito por Hunicke e LeBlanc (2004) e o capítulo do livro escrito por B. Kim (2015) sobre o assunto. O artigo de P. Leite (2013) também aborda conceitos semelhantes aos apresentados no método MDA.

O processo de avaliação do produto será baseado na abordagem de pesquisa-aplicação em educação, explorado por Plomp *et al.* (2018) e contará com escalas de autorrelato baseadas na escala de Likert, com descrevem Aguiar *et al.* (2011) no seu trabalho.

2.2.1 Método PBL

Segundo J. Savery e T. Duffy (1996), o método *Problem Based Learning* (PBL) foi proposto pela primeira vez nos anos 50, com aplicação principal no ensino de estudantes de medicina. Porém, com o passar do tempo, o modelo foi se adaptando ao ensino institucional de diferentes áreas do conhecimento.

A partir do modelo de Barrows (1986) é possível identificar um processo para resolução do problema e aprendizagem do discente. No primeiro momento, o aluno não possui conhecimento prévio do problema que lhe é apresentado. Então, é papel do discente buscar por fontes e meios para identificar o problema.

Depois da identificação do problema, os alunos podem então discutir suas ideias, hipóteses e avaliar o que precisa ser estudado para resolver o problema e o plano de ação para

o futuro. Ao fim dos trabalhos de pesquisa, com o conhecimento novo adquirido para resolução do problema, os alunos podem então aplicar o que foi aprendido para resolver o problema apresentado e avaliar o processo.

No contexto dos jogos, as etapas do processo são executadas a cada desafio apresentado ao jogador. Como descrito por L. Adães *et al.* (2015), o jogador é apresentado a uma *Fuzzy Situation*, sendo por vezes um desafio desconhecido ao usuário.

> Neste cenário são apresentados vários problemas a resolver (*Fact Finding*). O utilizador, ao longo do cenário, efectua uma análise e decide que ações tomar para concluir o seu objectivo, por exemplo: movimentação pelas diversas plataformas (*Idea Finding*). Numa instância final, o utilizador implementa a sua solução, jogando e concluindo o objectivo proposto (*Solution Finding*). (Adães *et al.*, 2015, p. 1)

2.2.2 Modelo MDA

O modelo *Dynamics and Aesthetics* (MDA) é uma metodologia de *design* de jogos descrita por Hunicke *et al.* (2004) que define o processo de construção da percepção do jogo a partir da criação de mecânicas que suportem a estética desejada. Com esse método, Hunicke condiciona a experiência do usuário, proporcionada pela estética às mecânicas nas quais o jogo se baseia.

Para descrever o impacto de decisões nas diferentes camadas de desenvolvimento e percepção de jogo, o modelo MDA separa a construção de um jogo em 3 camadas: Mecânicas, Dinâmicas e Estéticas.

- **Mecânicas:** descreve os componentes particulares de um jogo, a nível de representação de informações e algoritmos;
- **Dinâmicas:** descreve o comportamento em tempo de execução das mecânicas que atuam sobre as entradas do jogador e todas as outras interações resultantes delas;
- **Estética:** descreve a resposta emocional desejada despertada no jogador quando ele interage com o jogo.

A “direção” de interação com essas camadas é referenciada a partir do consumidor ou desenvolvedor. Segundo Hunicke *et al.* (2004), na perspectiva do desenvolvedor, mecânicas são construídas e dão origem ao sistema de dinâmicas, que, por sua vez, influenciam

experiências estéticas específicas. Porém, na perspectiva do jogador, o primeiro elemento que tem contato é a estética, que atribui um tom ao jogo e manifesta dinâmicas perceptíveis e, eventualmente, mecânicas a serem operadas e manipuladas.

Hunicke completa que é essencial para o desenvolvimento por essa abordagem que o jogo seja encarado como um artefato. Implicando que, dada a natureza imprevisível de um jogo, o conteúdo real dele reside nas interações do jogador com o que lhe é apresentado e não com o conteúdo exposto em si.

Pensar no jogo como um artefato ajuda a guiar o modelo para construir elementos baseados na interação com o jogo. Assim, as mecânicas do jogo devem sustentar a estética desejada, não atrapalhar a construção do cenário para o jogador.

2.2.3 Avaliação formativa em Pesquisa-Aplicação

Como Plomp *et al.* (2018) descreve em seu livro, a pesquisa-aplicação em educação é

> uma abordagem apropriada para desenvolver soluções a partir de pesquisa científica para problemas complexos no contexto da prática educacional, ou ainda desenvolver ou validar teorias sobre processos de aprendizagem, ambientes de aprendizagem e assemelhados. (Plomp *et al.*, 2018, p. 25)

A abordagem de pesquisa-aplicação é também sensível a contexto, sendo empregada em um grupo ou ambiente de aprendizagem para validar e desenvolver teorias e artefatos para resolver os problemas estudados. Sendo assim, o estudo e iterações da pesquisa-aplicação são guiados por uma pergunta central, generalizada por Plomp *et al.* (2018):

Quais são as características de uma <intervenção X> com o resultado/propósito Y no contexto Z?

No caso do objeto de pesquisa deste artigo, a pergunta central que guiará a pesquisa e validação do projeto realizado é: Quais são as características de uma ferramenta interativa para aprendizagem e experimentação de variáveis de controle de jogos de plataforma 2D para desenvolvedores de jogos, experientes ou amadores, explorarem as capacidades de movimentos de seus protagonistas?

A abordagem de pesquisa-aplicação é dividida em vários tipos de estudos para criação e aprimoramento de protótipos e artefatos importantes para construção de uma teoria válida

para solução aplicada para o problema. Entre esses estudos, podem-se destacar os estudos de de desenvolvimento, onde, segundo Plomp, pesquisadores e colaboradores desenvolvem intervenções funcionais para o contexto abordado, refletindo e moldando princípios de *design* para intervenções inovadoras.

Por sua vez, o estudo de validação tem foco no "desenvolvimento de ambientes de aprendizagem ou trajetórias com o fito de desenvolver ou validar teorias acerca do processo de aprendizagem e sobre como os ambientes de aprendizagem podem ser projetados." Plomp *et al.* (2018). O processo de validação pode ocorrer várias vezes durante o processo de desenvolvimento, a fim de validar um estado de protótipo do objeto desenvolvido, denominada por avaliação formativa; ou pode ser implementada no momento final de um estudo de pesquisa-aplicação, sendo aplicada a fase de avaliação somativa do produto, que, segundo Plomp *et al.* (2018).

> [...] as iterações iniciais na fase de desenvolvimento, a avaliação formativa dos protótipos deve focar na consistência e praticidade, ao mesmo tempo em que o critério da efetividade só ganhará importância crescente nas iterações posteriores. (Plomp *et al.*, 2018, p. 46)

Os critérios de qualidade podem ser avaliados entre relevância (validade do conteúdo), consistência (validade da construção), praticidade (utilizabilidade do projeto) e efetividade (se os resultados correspondem ao planejado). Como destacado por Plomp *et al.* (2018), cada critério de avaliação recebe foco diferente a depender da fase do projeto.

O foco dos critérios de avaliação dos protótipos que, no começo da pesquisa, se mantém na relevância do trabalho inserido na literatura existente; durante o desenvolvimento, avalia-se se o projeto se mantinha relevante, consistente e, por vezes, prático; nas fases finais do projeto de pesquisa, quando o produto está concluído, o foco se volta para avaliar a praticidade e efetividade real do projeto resultante, agora avaliado por um grupo do público-alvo.

Este artigo irá focar na validação de consistência e praticidade do projeto. Para fazer isso, é necessário desenvolver um sistema de avaliação do produto realizado com um grupo de pessoas selecionado para coleta de dados. Entre as estratégias de pesquisa apresentadas por Plomp *et al.* (2018), a estratégia escolhida para executar a pesquisa de qualidade do produto foi o teste.

Durante o teste, o grupo alvo selecionado usa o produto na prática e pode ser inquirido ou testado sobre o que foi feito durante a intervenção. Podem ser admitidos questionários, entrevistas e fazer o teste em um grupo sob observação para avaliar a praticidade da intervenção. Para avaliar a consistência, é possível realizar uma triagem no público-alvo. Para os fins deste projeto, o grupo selecionado para teste irá utilizar o produto e, posteriormente, responder um questionário onde irão quantificar aspectos da intervenção que pretendem ser avaliados. Cada uma das questões contará com valores entre 1 (para discordo totalmente) e 5 (para concordo totalmente), os valores marcados nesta escala irão qualificar as características de cada aspecto destacado.

2.2.4 Plataforma Godot

O motor de desenvolvimento Godot (2007) é um projeto *open source*, gratuito, baseado em uma construção de jogos por árvores de elementos em uma cena. Em cada uma dessas árvores, os *nodes* (nós) podem ser de componentes, assumindo responsabilidades simples na cena. Como por exemplo, nodes de exibição de imagem, de gerenciamento colisão, podem guardar um script personalizado pelo desenvolvedor ou podem também ser uma cena completa.

A composição de cenas na árvore de elementos da cena do Godot o torna uma ferramenta poderosa. Podendo construir, por exemplo, uma cena que recebe nodes para construir toda a funcionalidade de um *player* (jogador) e uma segunda cena para o cenário, onde se tem uma cena para o jogador e outras cenas para moedas espalhadas pelo cenário, inimigos, etc.

A comunicação entre esses nós é feita por *callbacks*, quando um elemento envia informações para um ou mais de seus nodes filhos. Há também a comunicação por sinais (*signals*) que é usada para alcançar os nodes acima da árvore.

2.2.4.1 Principais características

- **Flexibilidade de Plataforma:** O Godot oferece suporte multiplataforma para o desenvolvimento de jogos, abrangendo sistemas operacionais como Windows, macOS, Linux, e dispositivos móveis iOS e Android, além de possibilitar exportação para alguns consoles de videogame.

- **Editor Visual Intuitivo:** O editor Godot possui uma interface intuitiva que permite a criação de jogos, oferecendo suporte a linguagens como GDScript (uma linguagem de script *multi-thread* especializada no desenvolvimento de jogos) e também C#, C++, entre outras, para os desenvolvedores que optam por um código mais eficiente.
- **Motores 2D e 3D Consolidados:** Inicialmente reconhecido pelo robusto motor 2D, o Godot evoluiu para oferecer um conjunto completo de ferramentas para criação de jogos em 3D, incluindo recursos avançados como iluminação, shaders e física realista.
- **Sistema de Nodes e Física Integrada:** O Godot utiliza um sistema de nodes para organizar a lógica do jogo, oferecendo flexibilidade na construção das interações e elementos do jogo. Além disso, possui um sistema de física integrado para simulação de movimentos e colisões realistas.
- **Comunidade Ativa e Suporte:** A comunidade engajada fornece recursos, tutoriais e suporte extensivo, tornando o Godot uma opção atrativa para desenvolvedores iniciantes e experientes.

3 METODOLOGIA

O método de aprendizagem PBL, apresentado primeiramente por Barrows (1986), foi desenvolvido como modelo para processo de aprendizagem para estudantes de medicina, como descrito por Adães *et al.* (2015):

> *Problem Based Learning* surgiu inicialmente no campo da medicina, sendo que professores sentiam que os alunos beneficiariam mais experimentando situações reais do que através da aprendizagem de factos teórico. Existem 3 componentes principais no PBL (...): O *Fact Finding*, *Idea Finding* e *Solution Finding*. (Adães *et al.*, 2015, p. 1)

Os componentes do PBL apresentados são desenvolvidos para que o aluno possa ser exposto a um problema, pesquisar uma solução e, por fim, implementar a solução encontrada. Por sua definição flexível e construção didática focada na atividade prática e experimental, o método PBL conseguiu sair do seu contexto inicial, podendo ser modelado para outras áreas abordando os princípios de pesquisa e aprendizagem não guiada.

Adães *et al.* (2015) apresentam a implementação do método PBL no contexto de videojogos. Nesse contexto, o usuário – ou jogador – dispõe de controles para superar um desafio apresentado (*fact finding*); então ele pode analisar o cenário atual e decidir qual

conjunto de inputs de controle e atividades podem ser efetuados para solucionar o problema (*idea finding*) e, por fim, o jogador executa a solução encontrada na fase de análise e resolve o problema (*solution finding*), sem a necessidade de um guia explícito de como resolver o desafio.

No projeto apresentado neste artigo, uma *Fuzzy Situation* (situação nebulosa) é apresentada em cada fase que o jogador encontra, essa situação pode ser encontrada tanto nas descrições de cada fase, descrita no menu de seleção de fases; quanto na próxima construção de cada um dos níveis, onde são construídos cenários específicos para testar ou explorar um aspecto específico da movimentação da personagem.

O jogo é dividido em 5 seções com o total de 27 fases. Dessas 27 fases, 5 níveis são dispostos no fim de cada seção e são caracterizadas como "fases de desafio". Normalmente, uma fase comum no jogo apresenta uma situação problema, com uma breve descrição do tema na descrição da fase e trabalha um único aspecto alvo daquela aula. Nesse caso, novos *sliders* ou botões são apresentados ao jogador, permitindo modificar as variáveis pertinentes para conclusão daquela aula.

Fases de desafio são níveis mais longos dispostos no fim de cada seção que compilam todo conhecimento apresentado até então ao jogador. Diferente de fases normais, fases de desafio disponibilizam todos os recursos e variáveis para o jogador modificar e, por fim, conquistar o desafio. Essas fases têm como objetivo testar os conhecimentos estudados até então, como também testar a inventividade do jogador para criar um personagem capaz de concluir a fase e encontrar novas soluções a problemas mais complexos.

Para construir um jogo com bases teóricas, foi usado o *framework* MDA, descrito por Hunicke *et al.* (2004), como uma abordagem formal para entender e desenvolver jogos. Essa abordagem objetiva unir o *game design*, críticas, desenvolvimento e pesquisa técnica acerca do jogo.

Para utilizar este *framework* é importante entender que toda decisão – criativa ou não – para construção de um artefato deve ser baseada em uma metodologia de *design*. Essa afirmativa, por mais que seja impactante, é refletida no desenvolvimento de jogos, já que, como apontam Hunicke *et al.* (2004), "decisões aparentemente inconsequentes sobre dados,

representação, algoritmos, ferramentas, vocabulário e metodologia vão se acumulando, moldando o gameplay final" (tradução nossa).

Para explorar as características do jogo apresentado neste projeto, é preferível reconhecer as camadas do MDA pela perspectiva do jogador. Assim sendo, o primeiro aspecto que se tem contato é a estética. Para analisar as características estéticas de um jogo, não basta dizer que ele é "divertido" ou "prazeroso", há significantes definidos que nascem das dinâmicas compostas pelo jogo.

No caso do jogo apresentado, "No Controle" se trata de um jogo desafiador, exploratório e expressivo. O aspecto desafiador deriva da natureza e construção de cada fase do jogo, que define um objetivo claro para ser alcançado e obstáculos a serem superados para sua conclusão; exploratório é uma característica que parte não da exploração de um mapa extenso ou de personagens, mas na descoberta e testagem de combinações das habilidades dispostas em cada seção; e, por fim, é um jogo expressivo pois permite que o jogador exerça sua criatividade e crie, projete e imagine um personagem com as habilidades que comunica o que deseja, sendo cada experiência única e um reflexo do criador.

As dinâmicas que criam as estéticas citadas são, por exemplo:

- **Desafiador:** os obstáculos e desenho das fases penalizam o jogador que falha na tarefa apresentada; a curva de progressão de cada seção acompanha o conhecimento adquirido pelo jogador; as manobras exigem concentração e *inputs* precisos etc;
- **Exploratório:** cada fase apresenta uma nova forma de interagir com o ambiente apresentado; testar e entender cada variável apresentada é recompensado ao passo que jogadores que conhecem as mecânicas mais a fundo para serem mais eficientes;
- **Expressivo:** ao criar-se um personagem finalizado, modificando como pretende, o jogador é livre para expressar seus desejos e manipular os movimentos do personagem a sua vontade, possibilitando superar um desafio com soluções inovadoras.

As dinâmicas são resultantes de combinações das mecânicas implementadas no jogo. No modelo MDA, as mecânicas são o conjunto de regras, interações e limitações fundamentais para o funcionamento do jogo. Enquanto as mecânicas são comportamentos observados "em tempo de execução", mecânicas são trabalhadas anteriormente para construir os comportamentos adequados para cada jogo.

No caso do projeto apresentado, uma das principais mecânicas é a customização de habilidades da personagem, possibilitando modificar todos os aspectos de movimentação dispostos para o jogador. Outras mecânicas são os próprios controles do jogador, apresentados em cada seção, como pular, correr, desviar, atacar etc. Cada um desses elementos constroi um personagem e possibilitam a expressão do jogador para solucionar desafios propostos.

À medida que o jogo avança, novas mecânicas são introduzidas: *checkpoints* que salvam o progresso do jogador em uma fase; superfícies que, ao serem tocadas, reiniciam o progresso do jogador ao último *checkpoint* encontrado; uma barra de vida que informa visualmente quantas tentativas restam até que a fase seja completamente reiniciada; inimigos que podem ser derrotados ou devem ser evitados; alvos espalhados em certas fase que abrem uma porta trancada se todos forem destruídos pelo jogador *etc*.

As mecânicas de movimento que o personagem ganha acesso também têm importância para ditar o tipo de desafio que ele enfrentará. É notável que, ao fim do jogo, o jogador tem acesso a um longo conjunto de habilidades para customizar: andar, correr, pular, deslizar e escalar paredes, *dash*, ataque corpo a corpo e à distância e outras habilidades associadas como o pulo duplo e o *pogo*.

Para conseguir analisar a praticidade de se usar um sistema como o jogo apresentado no artigo como ferramenta para aprendizagem e exploração de todas essas mecânicas e dinâmicas, foi-se usado a abordagem pesquisa-aplicação, descrita por Plomp *et al.* (2018).

Para fazer uma avaliação formativa do protótipo desenvolvido, um formulário foi redigido para coletar dados de praticidade e consistência do projeto. Assim, com a análise dos dados, será possível determinar os próximos passos para construção do projeto e os trabalhos futuros para assegurar que “No Controle” seja uma alternativa viável para exploração e aprendizagem do conteúdo proposto.

O formulário redigido via Google Forms (2024) e consta com 23 questões, entre elas: 1 pergunta de múltipla escolha, 2 perguntas de alternativas únicas, 5 perguntas discursivas e 15 perguntas utilizando a escala de Likert, como descrito por Aguiar *et al.* (2011), para avaliar aspectos de praticidade e consistência do projeto, como também usabilidade de interface e facilidade de uso.

4 NO CONTROLE

"No Controle" se trata de um jogo de quebra cabeças e plataforma com múltiplas fases que exploram diferentes formas de interação com o ambiente por meio da movimentação do personagem. Nesta seção, serão exploradas algumas decisões tomadas para construção de um produto que pudesse abranger estes conceitos e resultar em um jogo desafiador e didático.

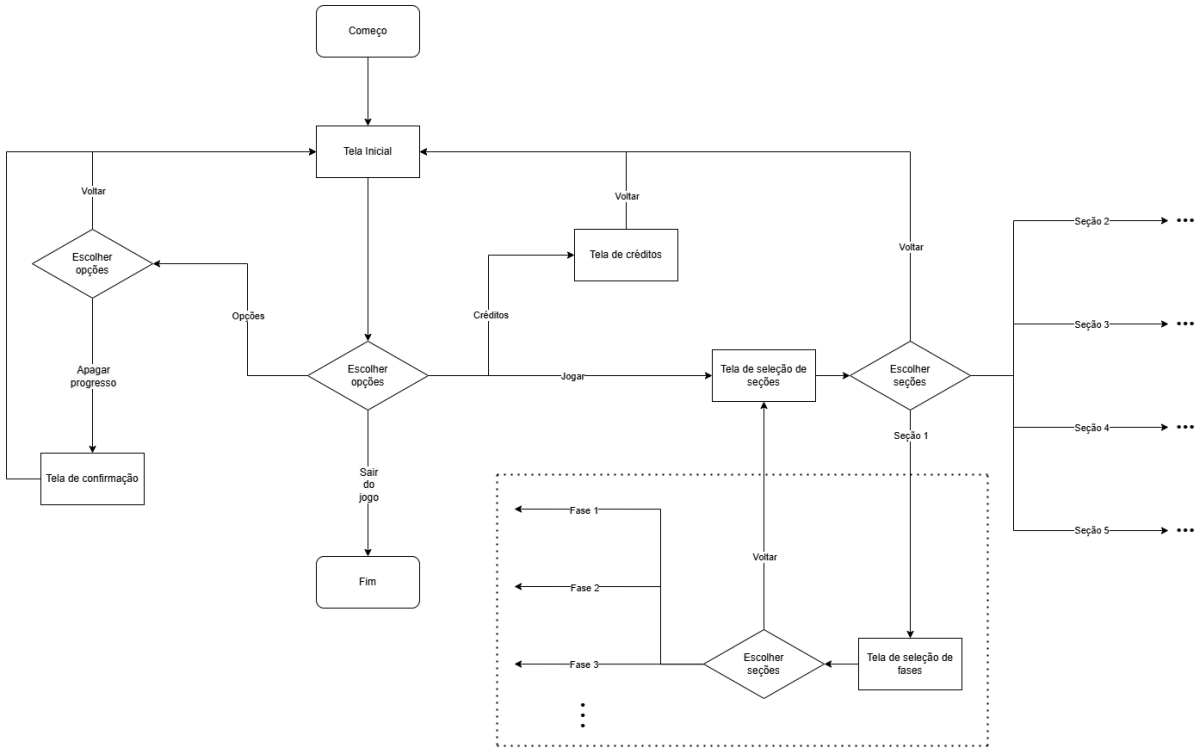
4.1 DESCRIÇÃO DAS INTERFACES DO PRODUTO

Por conta do número de fases e assuntos a serem tratados durante o jogo, foi necessário dividir os temas em diferentes seções, cada uma delas responsável em abordar uma característica do movimento do personagem a fim de introduzir e guiar o jogador por diversos conceitos.

4.1.1 Fluxo de interfaces de usuário geral

Para acessar as fases, o jogador, a partir do menu principal, precisa primeiro escolher uma das seções disponíveis no menu de seleção de seções. Uma vez escolhida uma seção, será direcionado a um menu de seleção de fases correspondente. Cada um desses menus apresenta uma lista única de fases que podem ser selecionadas e, uma vez que o jogador complete todas as fases de uma seção, a próxima seção é disponibilizada. A Figura 1 ilustra o fluxo de interação do usuário com essa estrutura de interfaces:

Figura 1 - Diagrama de fluxo do usuário para seleção de fases

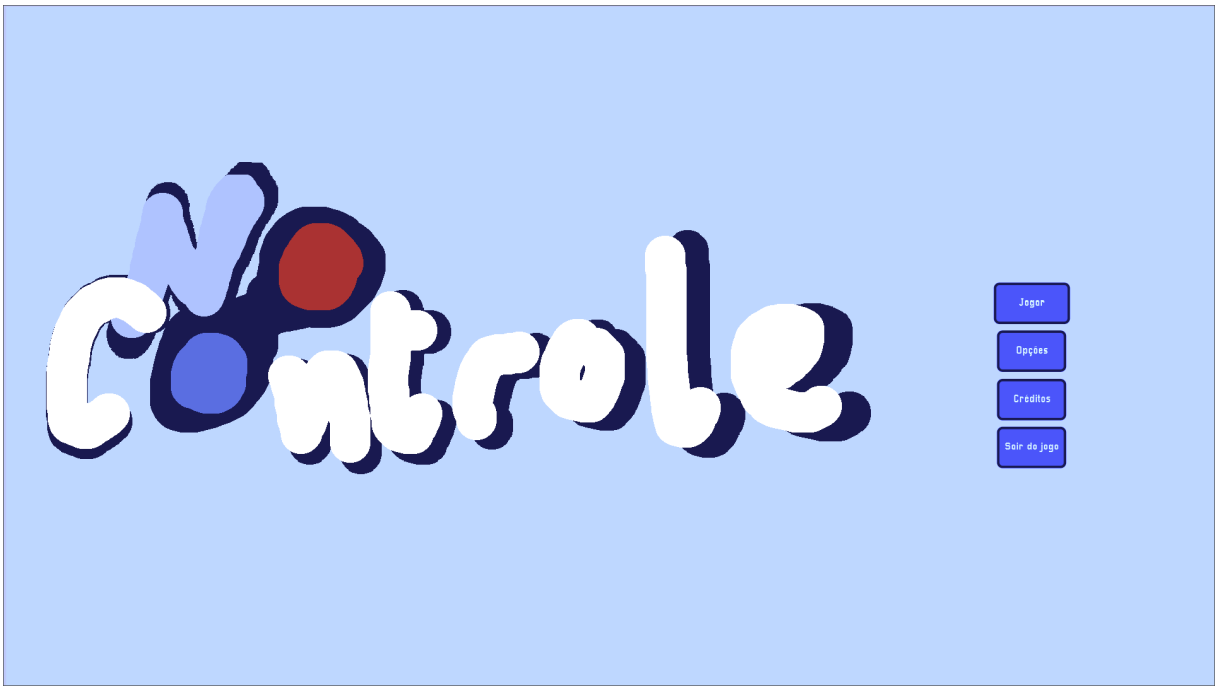


Fonte: Autor, 2024.

No diagrama acima, algumas zonas foram omitidas por conta do caráter modular do menu de seleção de fases, onde cada um dispõe de uma lista de fases distinta – e, portanto, uma quantidade variada de botões – como também o modelo destacado entre o retângulo tracejado é repetido em todas as outras seções.

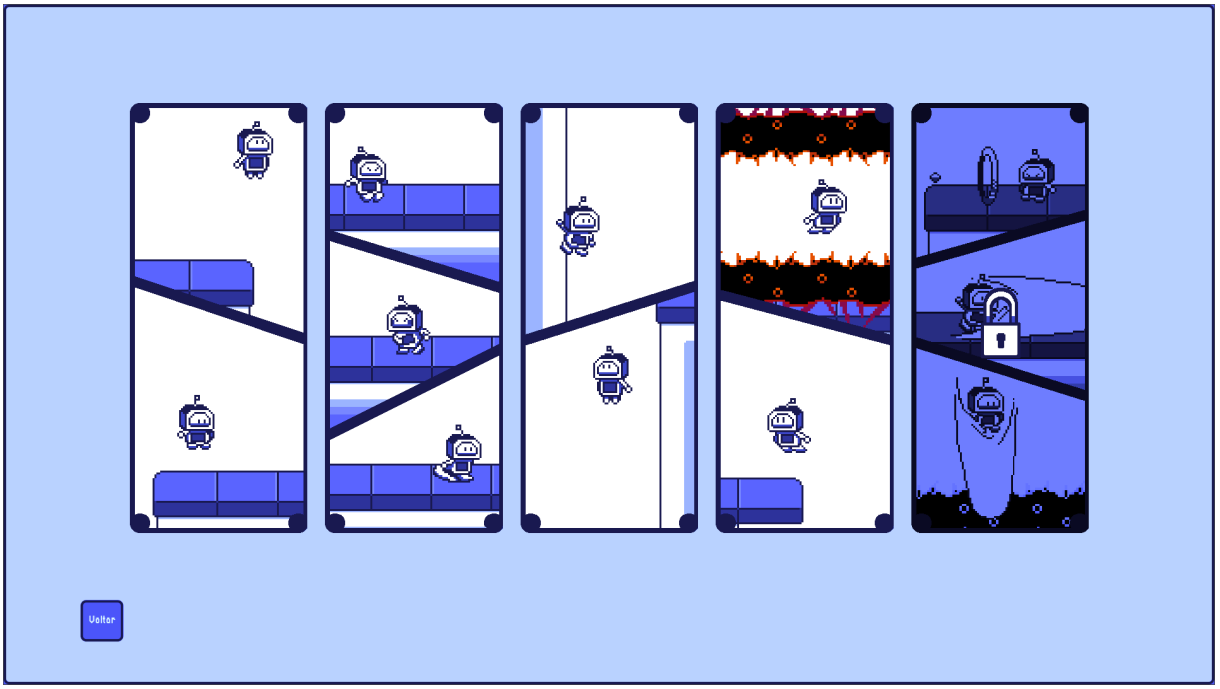
Abaixo, são listados exemplos de telas de interface do jogo. A Figura 2 se trata da tela de menu principal, seguida pela Figura 3 que ilustra um dos possíveis estados para tela de seleção de seções e, por fim, dois exemplos para telas de seleção de fases em diferentes seções, nas Figuras 4 e 5.

Figura 2 - Tela de menu principal



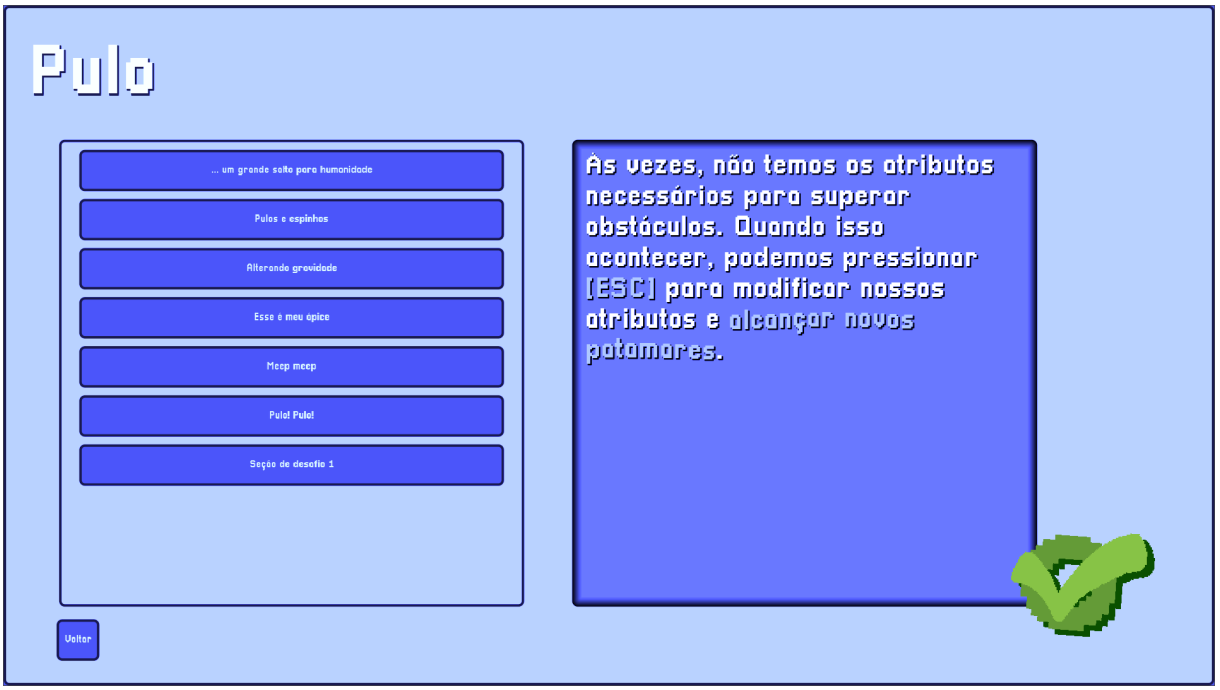
Fonte: Autor, 2024.

Figura 3 - Possível estado para tela de seleção de seções



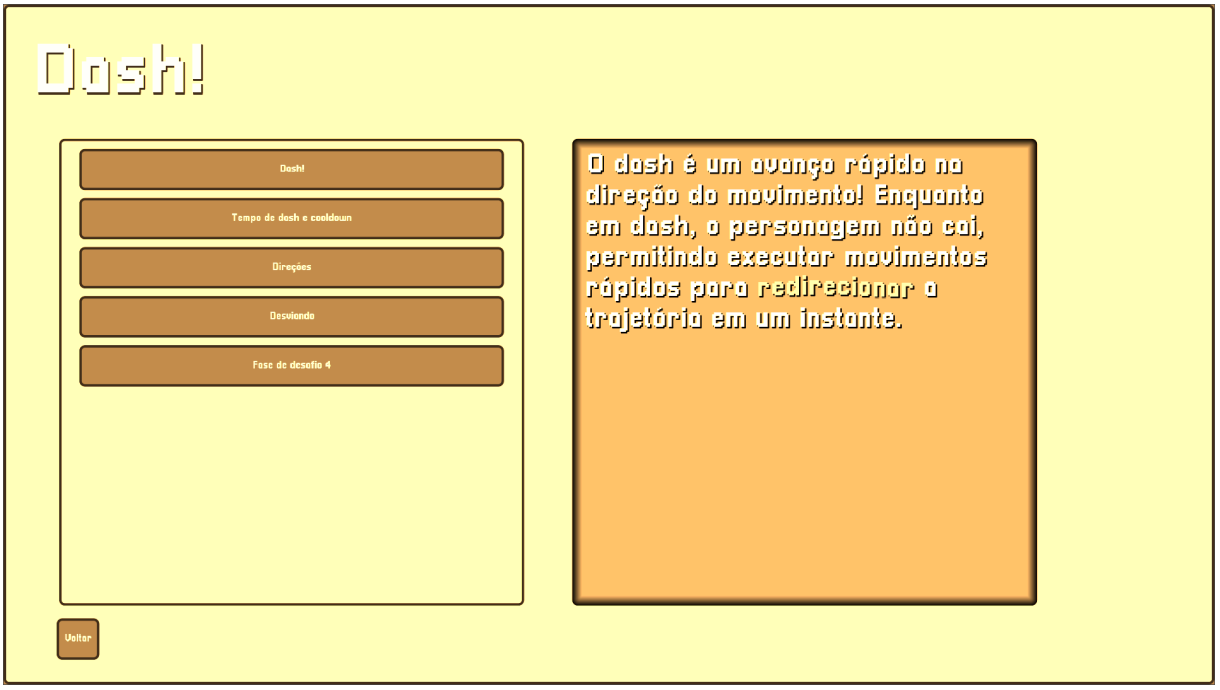
Fonte: Autor, 2024.

Figura 4 - Tela de seleção de fases da seção 1



Fonte: Autor, 2024.

Figura 5 - Tela de seleção de fases da seção 4

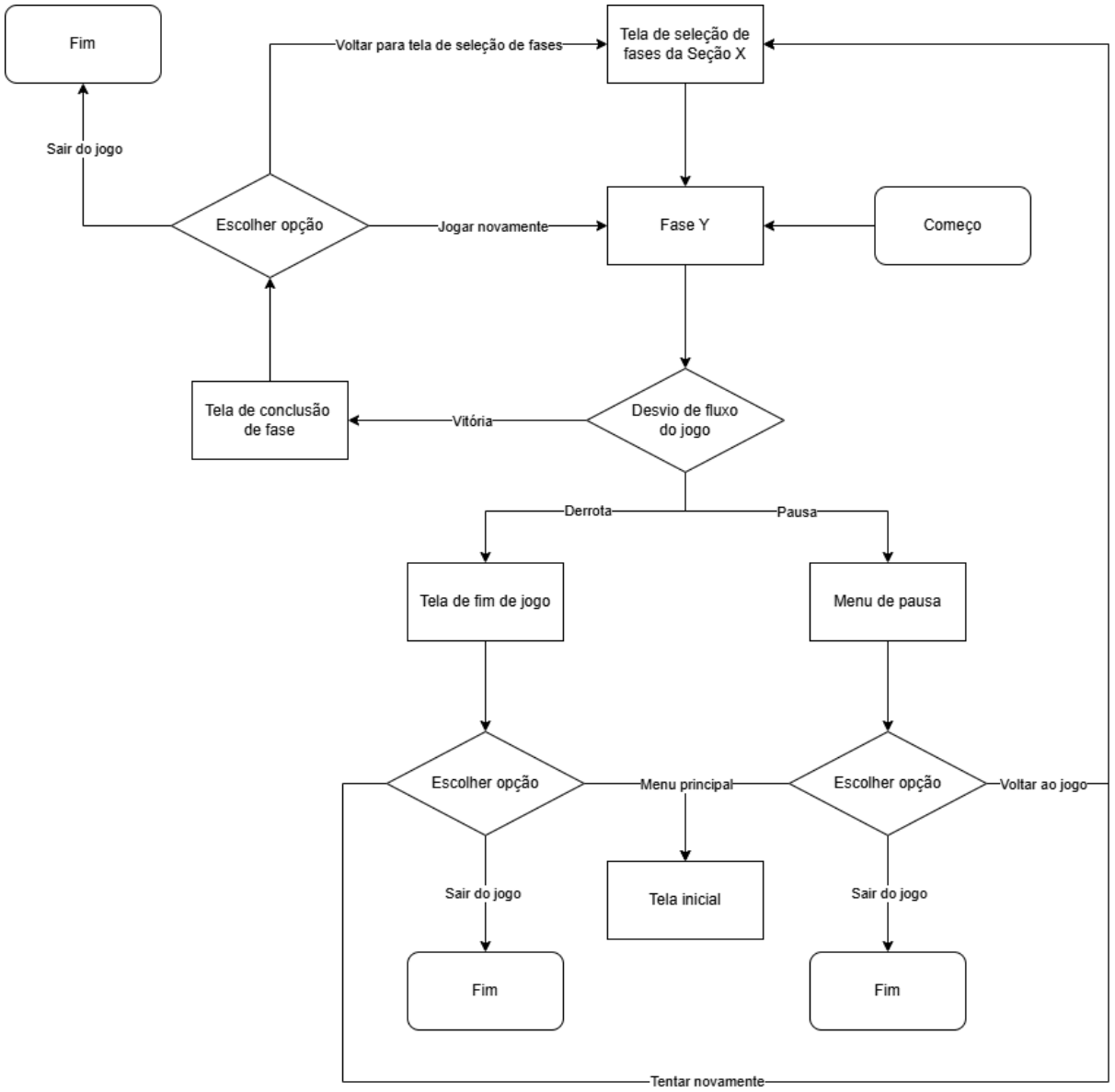


Fonte: Autor, 2024.

4.1.2 Fluxo de interfaces de usuário secundário

Durante a execução de uma fase, o jogador está sujeito a três fluxos de acesso às interfaces do sistema – Ilustrado na Figura 6 abaixo. O primeiro caminho é encontrado quando o jogador acessa o menu de pausa, tendo opções de retornar para fase, ao menu principal ou sair do jogo; um caminho semelhante se encontra em caso de derrota na fase. Já em caso de sucesso e conclusão de uma fase, o jogador poderá optar em jogar novamente o mesmo nível, voltar para seleção de outras fases na mesma seção ou sair do jogo.

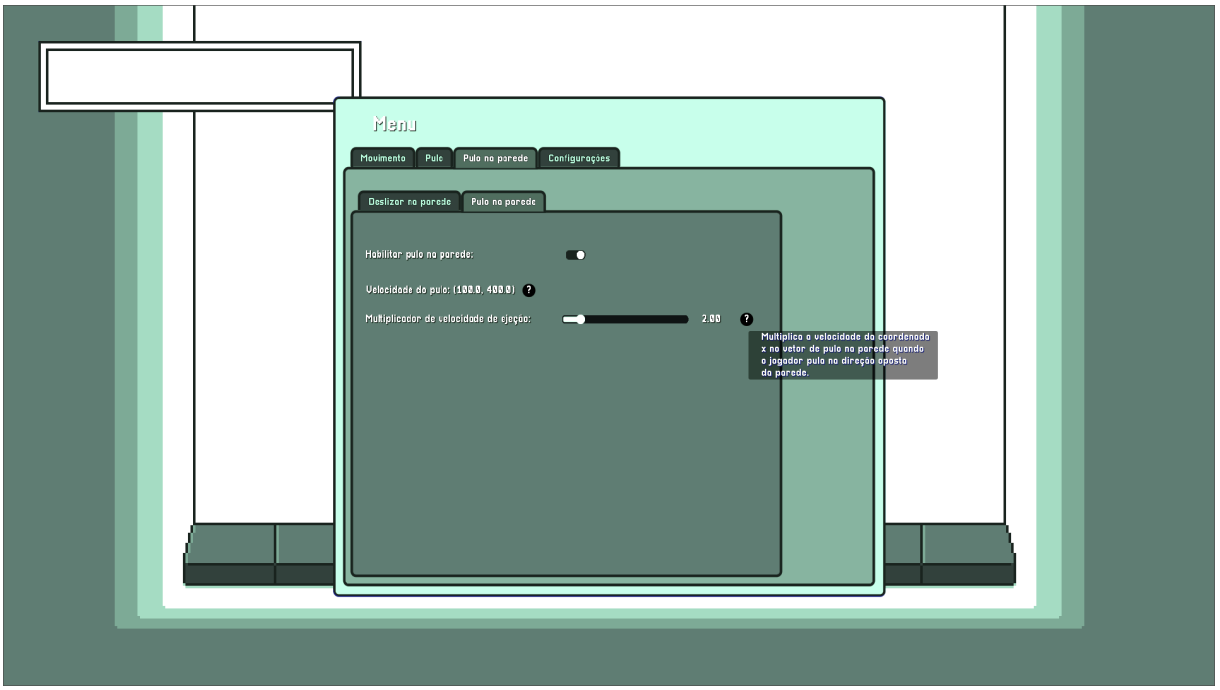
Figura 6 - Diagrama de fluxo do usuário a partir da execução de uma fase



Fonte: Autor, 2024.

O acesso da principal mecânica apresentada pelo jogo – a alteração de variáveis e atributos de personagem em tempo de execução – também é realizada por meio de uma interface. A interface de acesso para mudança dessas variáveis é feita no menu de pausa, que foi adaptado para cada fase, disponibilizando recursos distintos necessários para alteração do personagem e conclusão do objetivo apresentado. Como ilustrado na Figura 7 abaixo.

Figura 7 - Tela de menu de pausa da terceira fase da seção 3

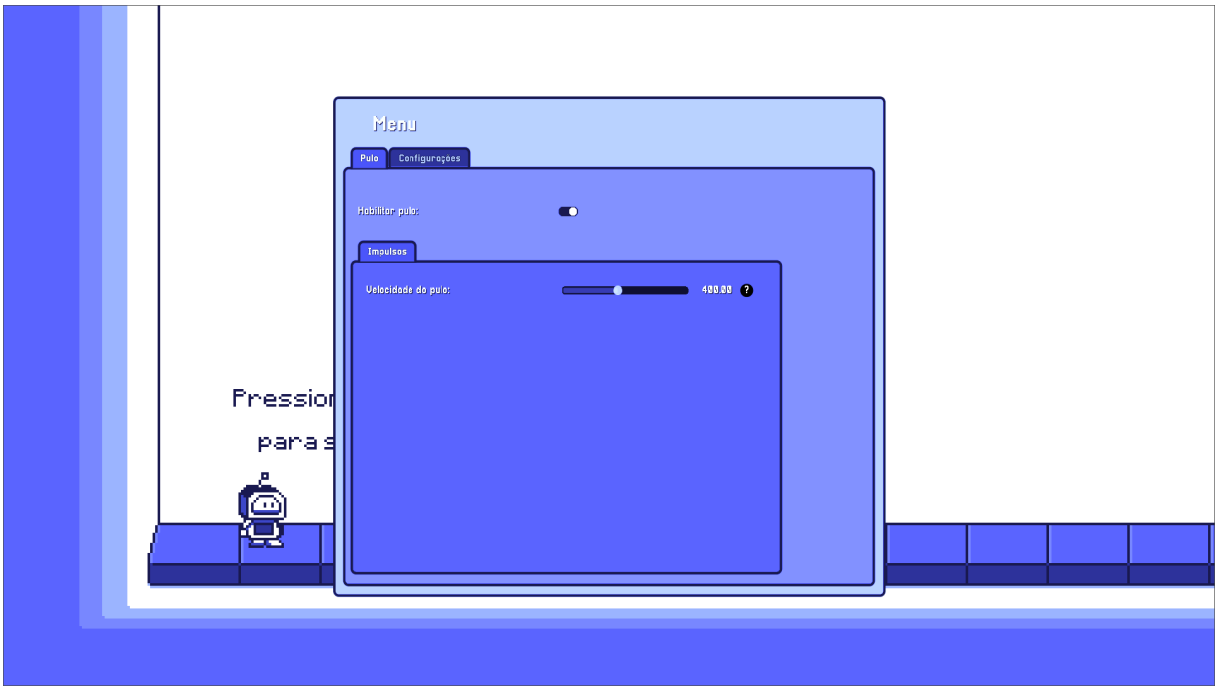


Fonte: Autor, 2024.

O menu é dividido em diversas partes, categorizadas pela funcionalidade na qual é responsável em modificar. Assim, o jogador pode navegar pelas abas e encontrar as variáveis de interesse para modificação. Cada variável apresenta uma *tooltip* que explica qual sua função para a movimentação do personagem, indicado no símbolo de interrogação à direita.

O número de abas e variáveis disponíveis no menu de pausa aumenta à medida que o jogador é introduzido a novos conceitos, tornando-se mais complexa a cada fase. Abaixo, as Figuras 8, 9, 10 e 11 representam exemplos da tela de menu de pausa em diferentes etapas de progressão do jogo:

Figura 8 - Tela de menu de pausa da primeira fase da seção 1



Fonte: Autor, 2024.

Figura 9 - Tela de menu de pausa da última fase da seção 1

Fonte: Autor, 2024.

Figura 10 - Tela de menu de pausa da quarta fase da seção 3

Fonte: Autor, 2024.

Figura 11 - Tela de menu de pausa da última fase da seção 5

Fonte: Autor, 2024.

Respectivamente, são exemplos das interfaces de vitória e derrota, nas Figuras 12 e 13:

Figura 12 - Tela de conclusão de fase

Fonte: Autor, 2024.

Figura 13 - Tela de derrota

Fonte: Autor, 2024.

4.2 CONSIDERAÇÕES SOBRE O PROJETO

A criação de um jogo é um processo complicado que envolve muitas partes na construção de um bloco coeso de instruções e objetos que se comunicam entre si e também possam operar independentemente, dada circunstâncias. Por conta disso, esta seção privar-se-á de comentar tópicos como desenvolvimento e controle de animações; construções de *tilemaps*; gerenciamento de sinais; processamento de recursos; fluxo de jogabilidade; criação de *HUD* (*Heads-up display*) e *UIs* (*User Interfaces*); elaboração do sistema de dano de entidades e renascimento do jogador; explicar em detalhes códigos e padrões usados para escrever comportamentos paralelos à discussão, como as diferentes formas de se programar o movimento do personagem, *etc*.

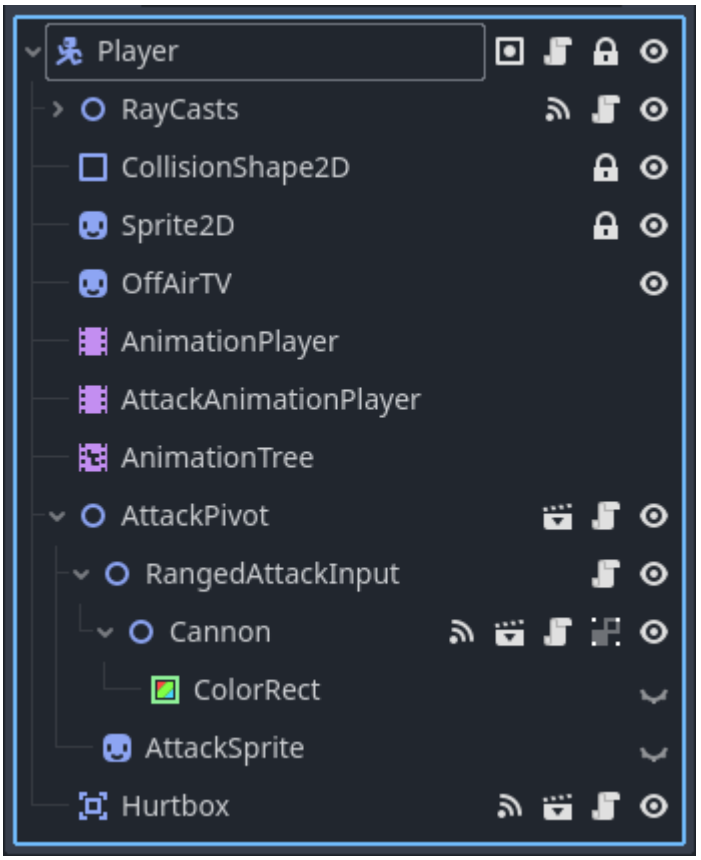
Assim, a discussão desta seção será restrita a falar sobre construção de personagem protagonista de "No Controle" com uso de composição para personalização de seus movimentos em tempo real; como gerar uma interface de menu de pausa customizada para cada uma fases com base nas suas exigências e como armazenar e resgatar os diferentes atributos para modificação durante o jogo. Todas as informações de funcionamento da *engine* podem ser encontradas em sua documentação (Godot, 2023).

4.2.1 Construção do protagonista por composição de elementos

Toda cena em Godot é construída como uma árvore onde cada nó adota uma função definida. Por exemplo, a Figura 14 abaixo representa a cena de um jogador como um *CharacterBody*, admitindo funcionalidades inerentes deste tipo de nó. Como filhos dele há nós para detectar colisão, no *CollisionShape2D*, ou para renderizar *sprites* ou animações – nos nós *Sprite2D*, *OffAirTV*, *AnimationPlayer* e *AttackAnimationPlayer*.

Todos os casos listados acima apresentam funcionalidades predefinidas pela *engine* da Godot. Porém, é possível programar um *script* personalizado para cada nó, indicado pelo ícone de pergaminho à direita da identificação do nó – como visto em *Player*, *AttackPivot* e *Hurtbox*, por exemplo. É possível então, ao se acoplar um *script* personalizado, modificar o comportamento de um nó via código.

Figura 14 - Árvore de hierarquia da cena de *Player*



Fonte: Autor, 2024.

O *script* do jogador funciona como uma central de processamento de estados, verificando apenas se o personagem está morto, foi atingido, está no ar etc. Esses sinais coletados no *script Player* são, então, repassados para processamento nas diferentes áreas de movimentação do personagem.

Como o personagem precisa se modificar em cada fase, suas funcionalidades de movimentação são construídas assim que ele é criado, a depender de quais movimentos serão usados em uma fase. Para isso, o jogador é composto de scripts para cada movimento, que são instanciados separadamente se, e somente se, há um correspondente do movimento em questão sendo trabalhado na fase. Como mostrado na Figura 15:

Figura 15 - Composição de comportamentos no *script* de jogador

```gdscript
# Player.gd
#region Moveset
var movement : Movement
var jump : Jump
var dash : Dash
var wall_slide : WallSlide
var wall_jump : WallJump
var wall_climb : WallClimb
#endregion

var is_hurt : bool = false
var is_dead : bool = false
var stats : PlayerStats :
    set(_stats):
        stats = _stats
        if stats.MOVEMENT_STATS: movement = Movement.new(self)
        if stats.JUMP_STATS: jump = Jump.new(self)
        if stats.DASH_STATS: dash = Dash.new(self)
        if stats.WALL_SLIDE_STATS: wall_slide = WallSlide.new(self)
        if stats.WALL_JUMP_STATS: wall_jump = WallJump.new(self)
        if stats.WALL_SLIDE_STATS: wall_climb = WallClimb.new(self)
        if stats.MELEE_ATTACK_STATS: attack_pivot.new(self)
        if stats.RANGED_ATTACK_STATS: ranged_attack_input.new(self, attack_pivot)
```

Fonte: Autor, 2024.

Para obter os dados do jogador, a fase, uma vez iniciada, copia os dados salvos atuais para garantir consistência de estados iniciais e os repassa para o seu jogador e o menu de pausa. Ilustrado da Figura 16 abaixo:

Figura 16 - *Script* de fase injetando os dados iniciais em suas dependências

```gdscript
# Level.gd
func _ready() -> void:
    var current_stats = SavedResources.current_stats.duplicate(true)
    $Player.stats = current_stats
    $PauseMenu.stats = current_stats
```

Fonte: Autor, 2024.

O código definido em *Player* tem o papel de concentrar as informações atuais do jogador e distribuí-las entre os *scripts* compostos. Dessa forma, o código do jogador consegue delegar as funções de cálculo de posição e física do corpo aos componentes que foram instanciados com o início da fase e apenas selecionar quais movimentos serão executados em cada momento. Esse processo está ilustrado na Figura 17 abaixo.

Figura 17 - O código do jogador apenas determina qual movimento será executado

```gdscript
# Player.gd
func _physics_process(delta):
    if is_dead: return
    if not stats: return
    movement.handle_movement(delta)

    var special_moveset = (is_dashing or is_wall_sliding or wall_jumping_timer > 0.0)
    if jump and not special_moveset:
        jump.handle_jump(delta)
    elif not special_moveset:
        velocity.y += gravity * delta

    if dash and not is_hurt: dash.handle_dash(delta)
    if wall_slide and not is_hurt: wall_slide.handle_wall_slide(delta)
    if wall_jump and not is_hurt: wall_jump.handle_wall_jump(delta)
    if wall_climb and not is_hurt: wall_climb.handle_wall_climb(delta)

    move_and_slide()
```

Fonte: Autor, 2024.

4.2.2 Recursos em Godot

Recursos são unidades de armazenamento de dados gerais do Godot. Um recurso pode armazenar qualquer tipo de dado, desde o mais primitivo, como números ou valores lógicos, até imagens ou mesmo aninhando mais recursos internamente. Esses valores podem ser alterados posteriormente na página Inspetor, no editor Godot.

O programador pode também criar recursos customizados usando uma classe filha à classe *Resource*. Assim, basta editar o arquivo como normalmente faria em qualquer outra classe. O diferencial dos recursos é que podem ser instanciados várias coleções de dados à partir do mesmo *template*, armazenando alterações em memória e modificando todas as

entidades que fazem relação com o recurso alterado simultaneamente. Um exemplo simples de um recurso customizado está ilustrado na Figura 18:

Figura 18 - Classe para criação de um recurso customizado para dados de fases

```
# LevelResource.gd
class_name LevelResource
extends Resource

@export var title : String
@export_multiline var description : String
@export var level_scene : PackedScene
@export var stats : PlayerStats
@export var done : bool
```

Fonte: Autor, 2024.

4.2.3 Configuração de fases

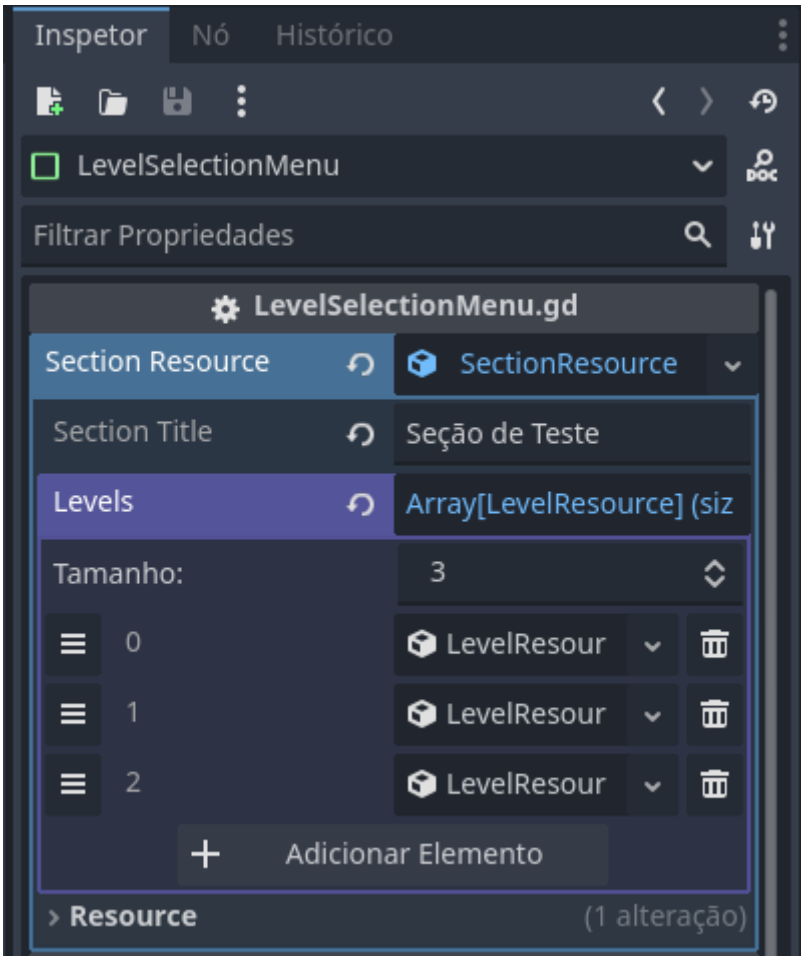
Para editar as diferentes fases, foi usado um recurso chamado *LevelResource*, que tem papel único de armazenar os dados específicos de cada nível do jogo. Cada seção – que também dispõe de um recurso, *SectionResource* – guarda um título e uma lista de recursos de fases. Deste modo, o menu de seção pode ser populado de informação automaticamente de forma mais efetiva e obtenção da cena do nível é mais confiável. As Figuras 19 e 20 mostram a organização para armazenamento de fases em uma seção.

Figura 19 - Definição da classe de *SectionResource*

```
# SectionResource.gd
class_name SectionResource
extends Resource

@export var section_title : String
@export var levels : Array[LevelResource]
```

Fonte: Autor, 2024.

Figura 20 - Exemplo de um recurso de seção criado a partir de *SectionResource*



Fonte: Autor, 2024.

O menu de seleções de fases de cada seção popula-se com base na lista de recursos, carregando em memória se o jogador já tiver concluído a fase ou encontrando o recurso correspondente àquela fase nos arquivos do jogo. Desta forma é possível manter o progresso jogador persistente, desbloquear novas seções e exibir essas informações na interface. A Figura 21 abaixo apresenta o código utilizado na seleção de fases para executar essa tarefa:

Figura 21 - Código para construção da lista de níveis no menu de seção

```gdscript
# LevelSelectionMenu.gd
func _add_level(level : LevelResource):
    var level_btn = LEVEL_BUTTON.instantiate()
    var user_path = format_user_path(level.resource_path)
    if not FileAccess.file_exists(user_path):
        level_btn.level_resource = ResourceLoader.load(level.resource_path)
    else:
        level_btn.level_resource = ResourceLoader.load(user_path)
    level_list.add_child(level_btn)
```

Fonte: Autor, 2024.

4.2.4 Construção e configuração do menu de pausa

Abaixo, na Figura 22, o código do menu de pausa está totalmente descrito. Ele contém uma variável *MenuComponentCreator* que é responsável em construir um menu de pausa único a cada fase. Para isso, uma vez que o *PauseMenu* recebe as informações do jogador na inicialização de uma nova fase (vide Figura 16), o criador de componentes do menu constroi as abas, botões e outros componentes definidos em seu código.

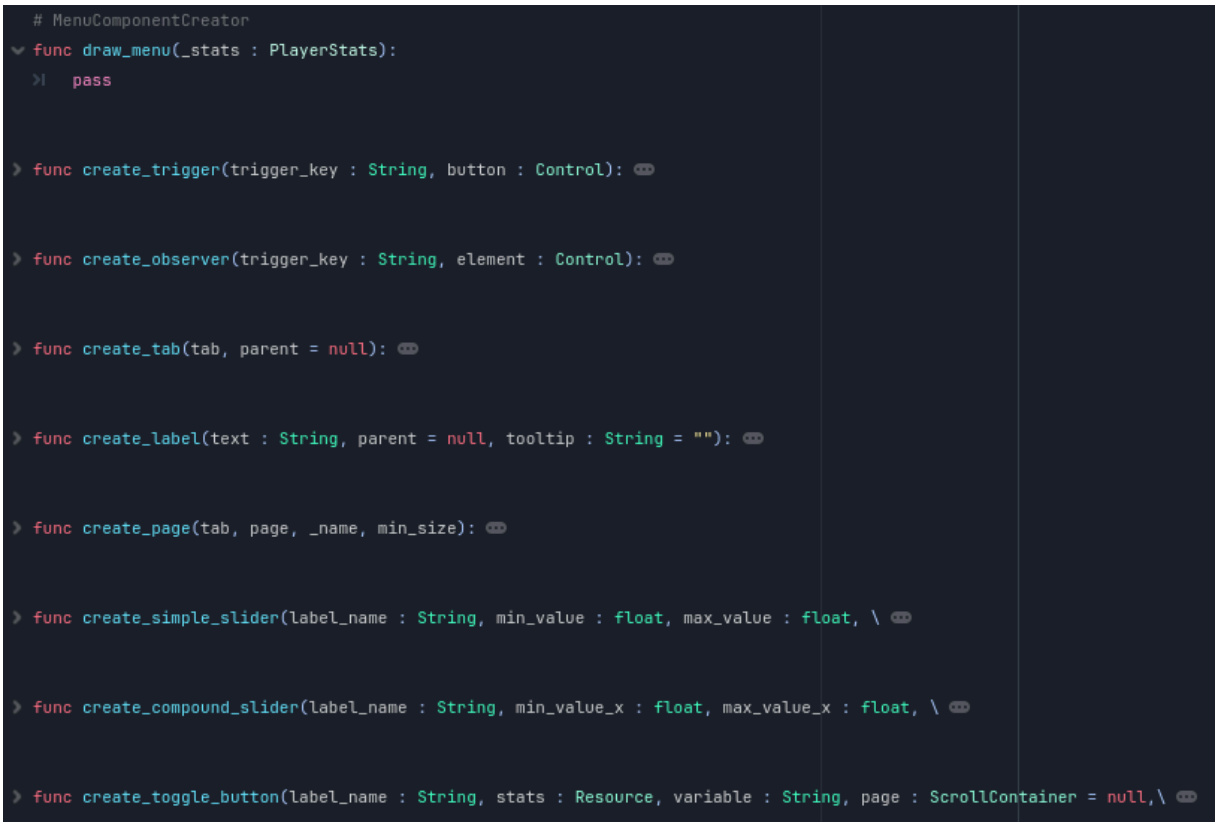
Figura 22 - Código de *PauseMenu.gd*

```
# PauseMenu.gd
extends CanvasLayer

@onready var component_creator : MenuComponentCreator = %ComponentCreator
var stats : PlayerStats :
    set(_stats):
        component_creator.draw_menu(_stats)

func _ready() -> void:
    hide()


func _unhandled_input(event):
    if event.is_action_pressed("pause"):
        toggle_pause()


func _on_resume_button_pressed():
    toggle_pause()


func toggle_pause():
    visible = not visible
    get_tree().paused = not get_tree().paused
```

Fonte: Autor, 2024.

Por sua vez, o *MenuComponentCreator* se trata de uma classe mãe para criação de elementos, responsável em reunir todos os métodos que configuram e posicionam cada um dos elementos solicitados por suas classes filhas. A Figura 23 abaixo apresenta os diferentes métodos construtores de elementos.

Figura 23 - Métodos de fabricação de componentes em *MenuComponentCreator.gd*



Fonte: Autor, 2024.

Para construir os diferentes elementos de interface, diferentes cenas foram criadas para agrupar comportamentos, já que cada componente dispõe de uma lista diferente de elementos com métodos de entrada e saída variados. Cada elemento apresenta um *script* que controla comportamento comum, como exibição das *tooltips*, e também específicos, como o processamento dos dados coletados e aplicação de restrições.

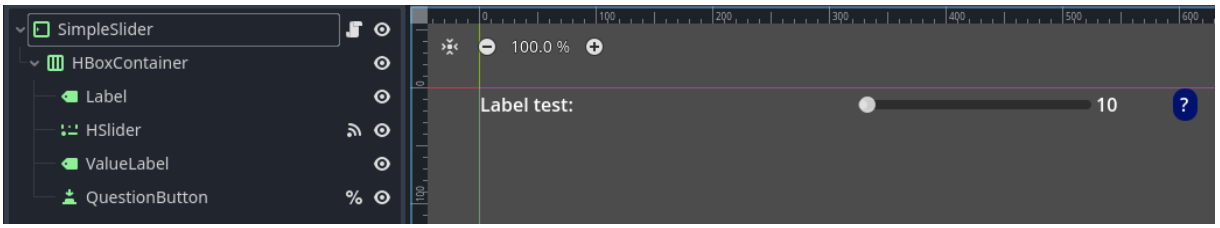
Abaixo, a Figura 24 apresenta a cena dos elementos sendo carregadas em *MenuComponentCreator.gd* e a Figura 25 mostra um exemplo de uma das cenas desses componentes ainda não configurada.

Figura 24 - Carregamento das cenas dos componentes de interface

```
# MenuComponentCreator
class_name MenuComponentCreator
extends VBoxContainer

const CHECK_BUTTON_CONTROL = preload("res://scenes/menus/components/check_button_control.tscn")
const COMPOUND_SLIDER_CONTROL = preload("res://scenes/menus/components/compound_slider_control.tscn")
const PAGE_COMPONENT = preload("res://scenes/menus/components/page_component.tscn")
const SIMPLE_SLIDER_CONTROL = preload("res://scenes/menus/components/simple_slider_control.tscn")
const TAB_COMPONENT = preload("res://scenes/menus/components/tab_component.tscn")
const LABEL_COMPONENT = preload("res://scenes/menus/components/label_component.tscn")
const SETTINGS_SCENE = preload("res://scenes/menus/components/settings_scene.tscn")
```

Fonte: Autor, 2024.

Figura 25 - Cena de um *slider* simples usado na interface de menu de pausa



Fonte: Autor, 2024.

Para criação de um menu de pausa customizado, é necessário criar uma classe filha de *MenuComponentCreator*. Essa classe sobrescreve o comportamento do método *draw_menu* definido em sua classe mãe. Nele, são descritos os comandos em ordem para inserção dos elementos de interface e criação de gatilhos de visibilidade para alguns elementos. Abaixo, na Figuras 26 e 27, há um exemplo do menu de pausa da terceira fase da seção de pulo, neste exemplo é possível ver a configuração de abas e a ligação de botões com o comportamento de variáveis descritas em recursos *JUMP_STATS*.

Figura 26 - Código para construção do menu de pausa da fase 3 da seção 2

```gdscript
# jump_3_menu.gd
extends MenuComponentCreator

func draw_menu(stats : PlayerStats):
	var selection_tab = TAB_COMPONENT.instantiate()

	create_tab(selection_tab)

#region Jump Page
	var jump_page = PAGE_COMPONENT.instantiate()
	create_page(selection_tab, jump_page, "Pulo", LARGE_PAGE)

	create_toggle_button("Habilitar pulo", stats.JUMP_STATS, "enable_jump", jump_page)

	var jump_set_tab = TAB_COMPONENT.instantiate()
	create_tab(jump_set_tab, jump_page)

#region Impulses
	var impulses_page = PAGE_COMPONENT.instantiate()
	create_page(jump_set_tab, impulses_page, "Impulsos", MEDIUM_PAGE)

	create_label("Velocidade do pulo: %.02f" % stats.JUMP_STATS.jump_velocity, impulses_page,
		"Influencia na altura máxima que um pulo poderá alcançar")
#endregion

#region Falling
	var falling_page = PAGE_COMPONENT.instantiate()
	create_page(jump_set_tab, falling_page, "Queda", MEDIUM_PAGE)

	create_label("Modificadores de gravidade", falling_page)
	create_simple_slider("Multiplicador de gravidade na queda", 1, 4, 0.1, stats.JUMP_STATS, "gravity_fall_multiplier", falling
		"Altera o comportamento da gravidade em um personagem em queda, recomendado para que não passe a sensação que está flut
	create_simple_slider("Multiplicador para corte do pulo", 0, 1, 0.01, stats.JUMP_STATS, "jump_cut_multiplier", falling_page,
		"Afeta a altura do pulo ao soltar o botão de pulo durante a subida. Quando o multiplicador está em 0, a velocidade do p
#endregion
#endregion

	var settings = SETTINGS_SCENE.instantiate()
	create_page(selection_tab, settings, "Configurações", LARGE_PAGE)
```

Fonte: Autor, 2024.

Figura 27 - Resultado dentro do jogo da interface definida em *jump_3_menu.gd*

Fonte: Autor, 2024.

4.2.5 Armazenamento e resgate de dados de configuração de personagem

Recursos foram utilizados para armazenar os parâmetros iniciais diferentes de cada fase para todos os tipos de movimento. Na Figura 26 é possível identificar um recurso chamado *JUMP_STATS*, nele são armazenados os dados pertinentes para controle do pulo do personagem. Naquele exemplo, os dados são referentes à configuração inicial na fase 3 da primeira seção. Deste modo, todos os outros aspectos de movimentação do personagem abordados em “No Controle” apresentam um recurso para armazenar seus dados.

Abaixo, um exemplo de construção de um desses recursos, primeiro em classe, descrito na Figura 28 e, em seguida, um exemplo de configuração de um *WallJumpResource* na página Inspetor da Godot, na Figura 29.

Figura 28 - Descrição do recurso de pulo na parede, herdando a classe *Resource*

```gdscript
# WallJumpResource.gd
class_name WallJumpResource
extends Resource

@export_group("Wall Jump")
@export var enable_wall_jump: bool = false

@export var wall_jump_vector: Vector2 = Vector2(100, 400)
@export var wall_jumping_grace_time: float = 0.1
@export var jump_off_wall_multiplier: float = 1
```

Fonte: Autor, 2024.

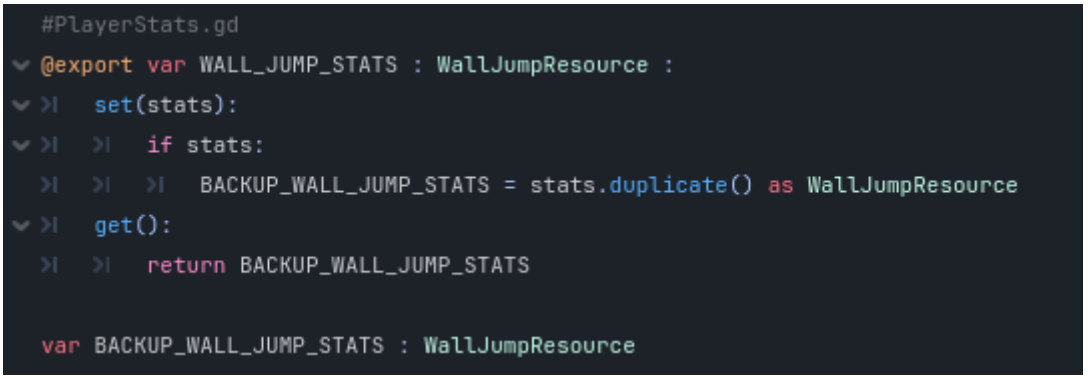
Figura 29 - Exemplo de um recurso instanciado a partir de *WallJumpResource*

Fonte: Autor, 2024.

Para garantir a consistência dos estados iniciais de cada fase, é realizado um backup na classe do recurso de *PlayerStats*. O procedimento encapsula o valor enviado para configuração de suas variáveis e cria uma cópia do parâmetro recebido. Assim, no momento que um elemento externo solicita os dados definidos em *PlayerStats* para modificação durante uma fase, a versão original definida no recurso da fase está segura e inalterada. Todas as

modificações realizadas nesta cópia são perdidas uma vez que os dados do jogador são lidos ao iniciar uma nova fase. Abaixo, na Figura 30, o processo de encapsulamento das variáveis está sendo exemplificado com *WallJumpResource*, porém, o mesmo processo é realizado com todos os outros recursos definidos em *PlayerStats.gd*.

Figura 30 - Exemplo de encapsulamento de recursos feito em *PlayerStats.gd*



Fonte: Autor, 2024.

Com isso, ao fim de cada nível, o progresso da fase pode ser salvo sem alterar seus parâmetros iniciais. Para executar esse processo, um recurso referente à fase concluída é gravado nos arquivos da máquina do jogador. O caminho padrão usado pela godot para isso é "user://" que, segundo a documentação da Godot (Godot, 2023), é um prefixo que aponta para diretórios específicos em cada sistema operacional. No Windows, "user://" redireciona para "%APPDATA%\Godot\app_userdata\[project_name]" e, nas distribuições Linux, o diretório se refere à "~/.local/share/godot/app_userdata/[project_name]".

Uma vez com o caminho para armazenamento definido, uma pasta é criada para separar o jogo em seções e então grava o recurso da fase atual. O recurso salvo em memória será usado para conferir quantas fases o jogador concluiu em cada seção e, por fim, habilitar novas seções para o jogador explorar.

O código que permite salvar a fase concluída está descrito na Figura 31.

Figura 31 - Código para salvar progresso do jogador

```gdscript
# WinMenu.gd
func save_progress():
    if not FileAccess.file_exists(current_level_path):
        var dir_path = "user://levels/%s" % get_section()
        if not DirAccess.dir_exists_absolute(dir_path):
            DirAccess.make_dir_recursive_absolute(dir_path)
        FileAccess.open(current_level_path, FileAccess.READ_WRITE)
    ResourceSaver.save(SavedResources.current_level, current_level_path)
```

Fonte: Autor, 2024.

5 RESULTADOS E ANÁLISE

A pesquisa foi realizada via Google Forms (2024) durante o período de uma semana – entre os dias 28 de outubro de 2024 e 4 de novembro de 2024 – e obteve um total de 24 respostas. O formulário constou com um total de 23 perguntas, sendo, em sua maioria, perguntas em escala de Likert, que serão utilizadas para análise objetiva em conjunto com questões discursivas e de múltiplas escolhas para definir um contexto e também entregar o *feedback* necessário para análise de certos fenômenos observados na pesquisa.

Para esta análise, é importante segmentar a discussão de temas para melhor compreensão dos dados a seguir e também para fundamentar os temas e resultados: o primeiro tema abordado será uma análise rápida do público respondente do formulário, essa análise servirá de contexto para discussões futuras; o segundo tema será a praticidade do sistema, um dos principais objetivos do estudo, e abordará questões como facilidade e experiência com o uso das interfaces e mecânicas apresentadas para o jogador. Por fim, a consistência do produto será analisada observando os dados sobre progressão de fases e desafios e implementação de certos pontos teóricos no produto.

A escala de Likert utilizada para coletar os dados nesta pesquisa foi descrita por Aguiar *et al.* (2011) e sua estrutura é utilizada para quantificar um relato, muitas vezes qualitativo, de um aspecto da pesquisa:

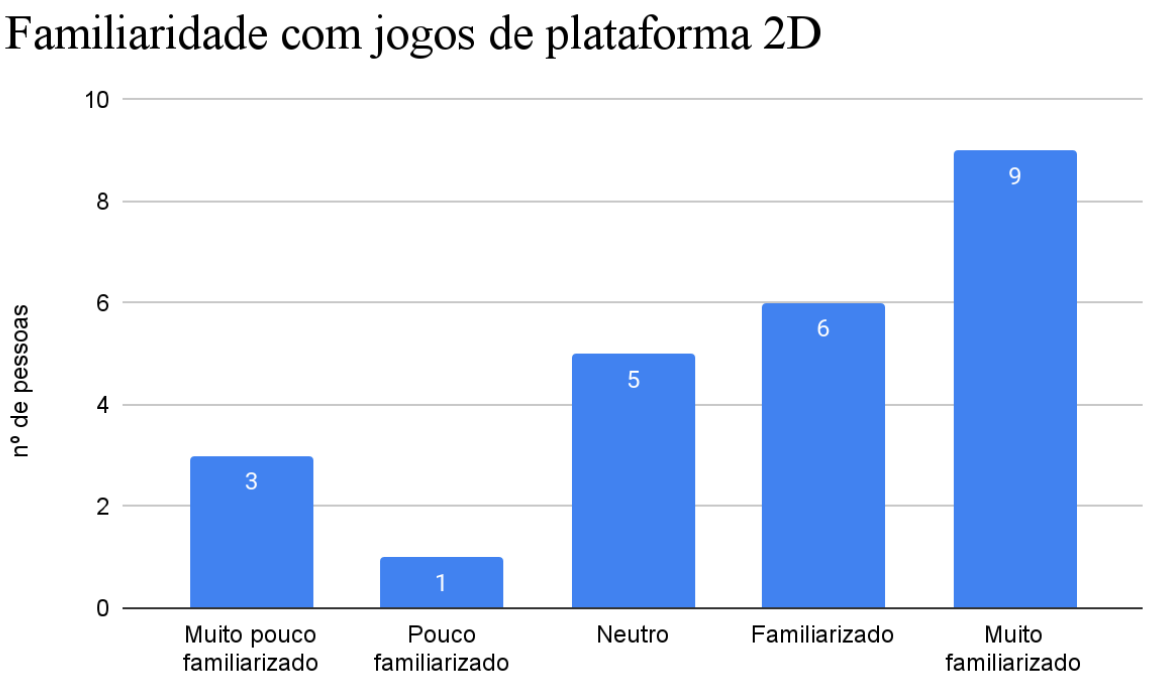
> Para analisar os resultados coletados por uma escala Likert, atribui-se valores para cada um dos itens, começando em zero para o item neutro e aumentando

> ou diminuindo em 1 para cada item acima ou abaixo, respectivamente [...] (Aguiar *et al.*, 2011, p. 2)

Assim, em uma escala de autorrelato baseada na metodologia de Likert com 5 opções, há 3 campos a serem analisados: resultados negativos para opções de discordância; um ponto neutro ao centro e resultados positivos para opções de concordância. Essa diferenciação é importante para abstrair a intensidade de uma opção e focar em sua qualidade.

Por exemplo, para conhecer o público respondente, foi feita uma pergunta para identificar a familiaridade dos respondentes com jogos de plataforma 2D, o objeto de pesquisa. O gráfico abaixo demonstrado na Figura 32 mostra o resultado segundo uma escala de Likert de 5 opções:

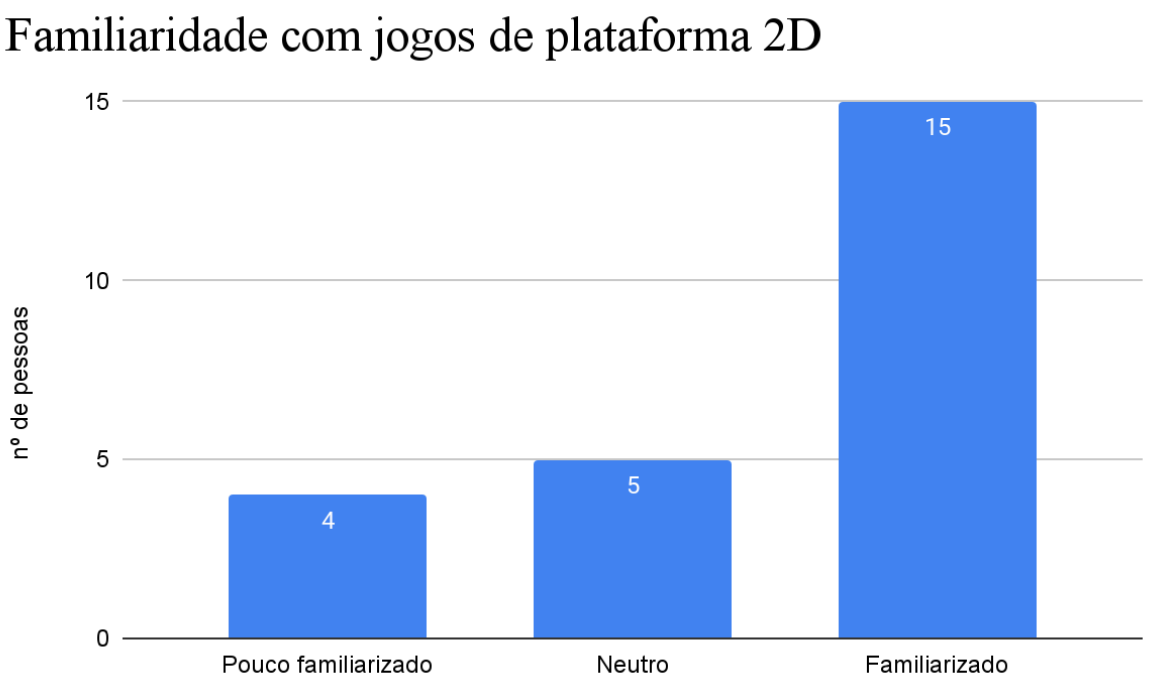
Figura 32 - Gráfico sobre familiaridade com jogos de plataforma 2D na escala Likert



Fonte: Autor, 2024.

Em contrapartida, as mesmas informações podem gerar um gráfico mais legível que explora os mesmos dados e resultam na mesma conclusão, mesmo que com uma abordagem menos prolixa, como ilustrado na Figura 33 abaixo:

Figura 33 - Gráfico sobre familiaridade com jogos de plataforma 2D agrupado

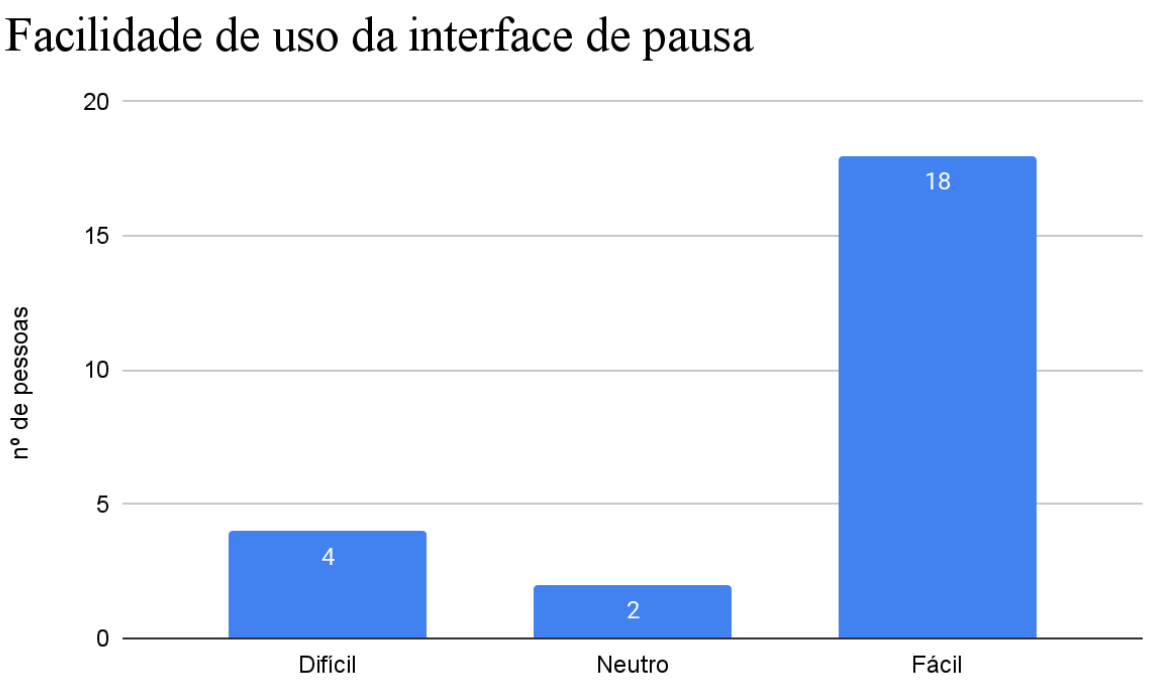


Fonte: Autor, 2024.

Como pode ser observado no gráfico da Figura 33, a maioria dos participantes apresenta uma boa familiaridade com jogos com o mesmo gênero apresentado pelo objeto de estudo. Essa informação apresenta um contexto importante para abordar temas sobre a praticidade do sistema proposto.

Um dos aspectos mais importantes da praticidade do sistema é o contato do jogador com a principal mecânica do jogo: a alteração de atributos do personagem, realizada por meio de ajustes de elementos em uma interface. Logo, é imperativo que a interface de acesso à essa mecânica seja intuitiva e apresente um bom fluxo de atividades para manejo e entendimento de todas as variáveis, como mostrado na Figura 34.

Figura 34 - Gráfico sobre a facilidade de uso do menu de pausa

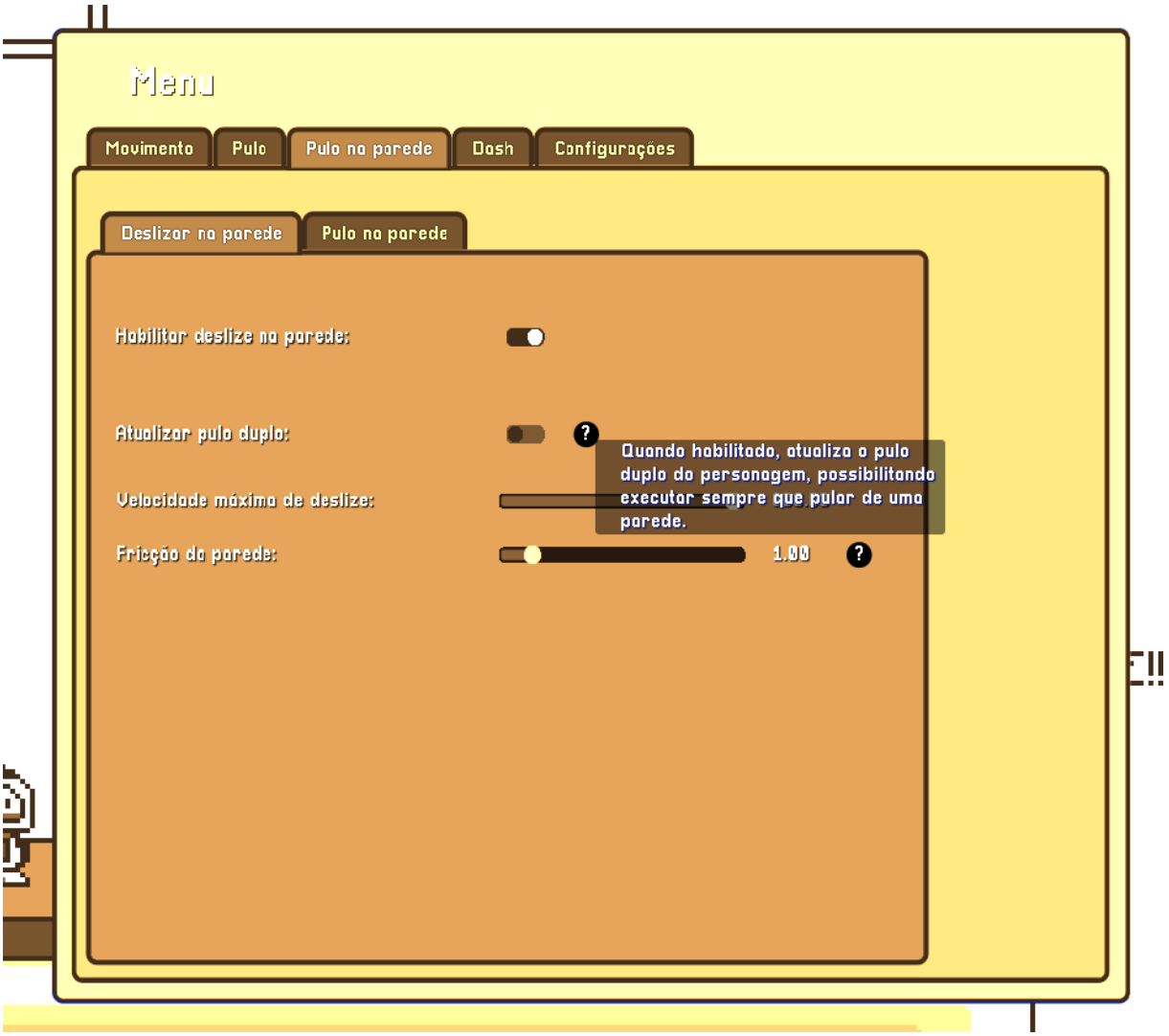


Fonte: Autor, 2024.

Como pode ser observado no gráfico acima, a aceitação entre os respondentes sobre a interface, que apresenta a mecânica principal do jogo, foi muito positiva. O manejo fácil, intuitivo e responsivo dessa interface possibilita uma conexão do jogador com a mecânica que o permite explorá-la para encontrar soluções para os desafios sem muitos atritos.

Porém, para que a mecânica funcione no contexto do PBL, é importante também que o jogador tenha acesso à explicações rápidas do conteúdo para entender o contexto e poder estudar táticas e desenvolver possíveis soluções. Para isso, toda variável no menu que necessita de explicação mais extensa, apresenta um botão para explicação de conceitos que podem guiar o jogador (aluno) para compreender as alterações que podem ser feitas. Essas descrições são chamadas de *tooltips* e podem ser acessadas ao posicionar o ponteiro do *mouse* acima de botões com uma "?" ao lado da variável correspondente, como ilustrado na Figura 35 abaixo:

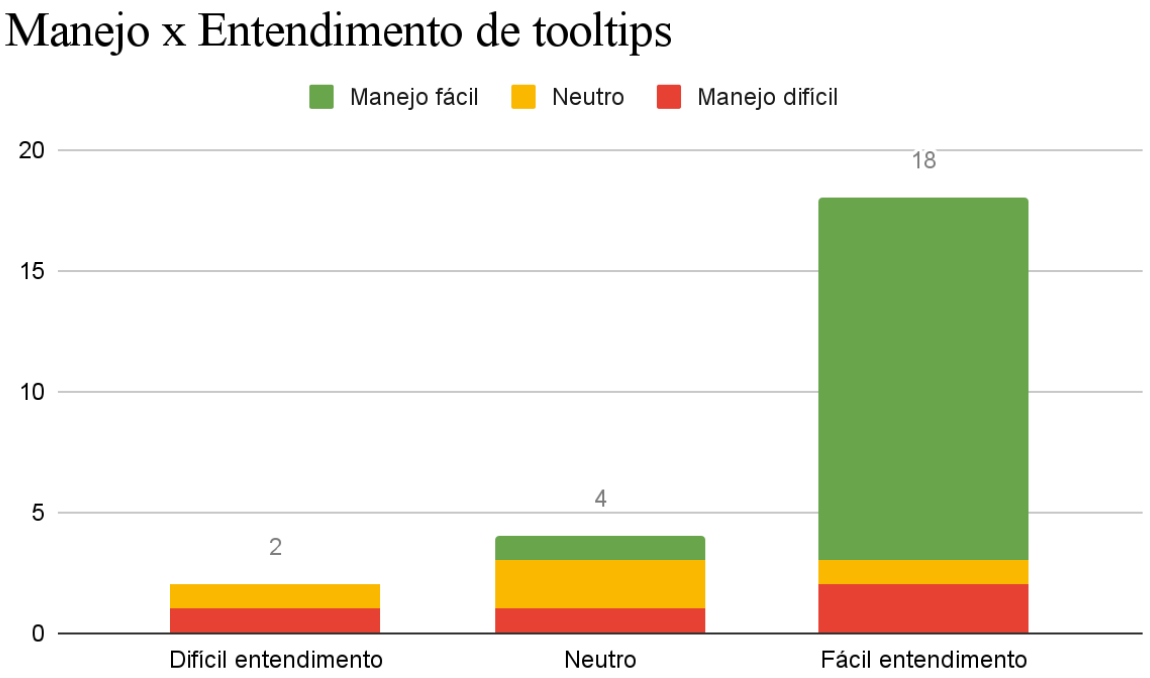
Figura 35 - Exemplo de *tooltip* na menu de pausa do protótipo



Fonte: Autor, 2024.

Foram coletados dados sobre o manejo e compreensão dessas *tooltips*, que são elementos essenciais para a aplicação do método e exercício da aprendizagem. O gráfico da Figura 36 abaixo ilustra os resultados acerca do manejo e compreensão desse elemento de interface:

Figura 36 - Gráfico sobre manejo e compreensão de *tooltpis*



Fonte: Autor, 2024.

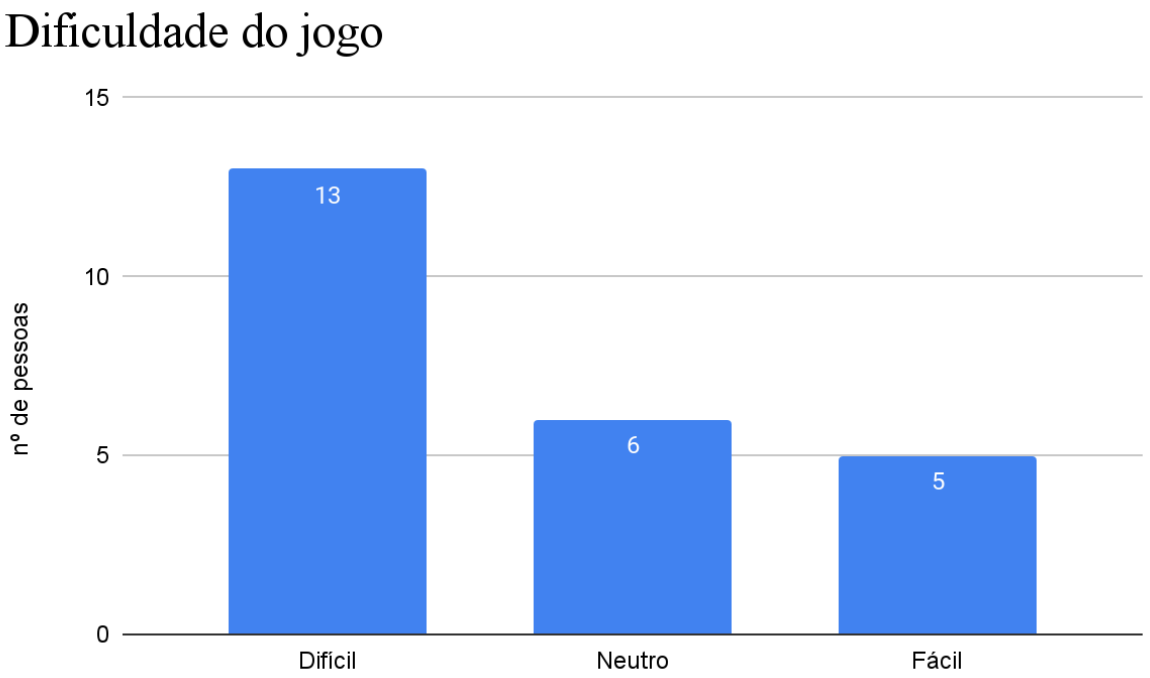
Como pode ser observado, o eixo correspondente ao entendimento das descrições apresenta tendência crescente, sendo majoritariamente positivo. No entanto, é possível observar também que, mesmo com um alto grau de entendimento, os resultados referentes ao manejo e acesso das *tooltips* ilustram uma persistência na dificuldade de acesso a essas explicações.

Acerca deste problema, foi constatado que os respondentes encontraram uma barreira que impediu o acesso intuitivo dessa ferramenta importante e, em muitos casos, prejudicou o uso da mecânica principal. Essa barreira foi identificada como um *bug*, onde, mesmo quando o ponteiro do *mouse* dispunha-se acima do botão da *tooltip*, era necessário esperar 0,5 segundo para a resposta da interface. Geralmente, esse comportamento pode ser encarado como padrão em outros *softwares*, porém, no contexto do jogo apresentado, a resposta imediata da interface é esperada da interface pelo jogador e, por isso, torna-se imprescindível para uso da ferramenta desenvolvida.

O segundo aspecto da praticidade no jogo é a dificuldade enfrentada para concluir os objetivos. Por mais que uma das principais dinâmicas do jogo apresentado seja o desafio

encarado pelo jogador, é importante balancear a experiência para que a dificuldade enfrentada não venha a desincentivar o jogador a continuar para a próxima fase. O gráfico ilustrado na Figura 37 abaixo mostra o resultado da pesquisa acerca da dificuldade do jogo apresentado:

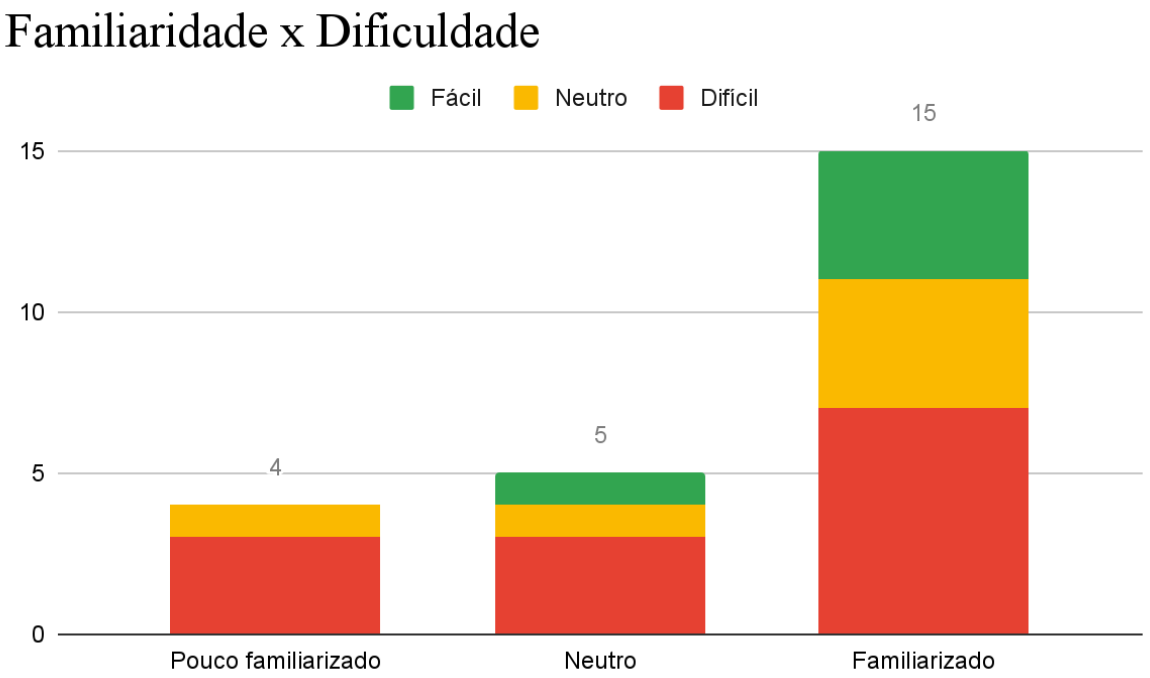
Figura 37 - Gráfico sobre dificuldade do jogo "No Controle"



Fonte: Autor, 2024.

É possível observar no gráfico uma clara tendência para um jogo difícil. Esse fenômeno pode ser examinado por diferentes lentes. Por exemplo, o histórico de um jogador e a familiaridade com gêneros distintos influenciam na sua capacidade de performar como esperado no primeiro momento. Diante disso, é importante também analisar o repertório do jogador e conhecer suas competências prévias ao jogo analisado. Para isso, é possível traçar um gráfico que ilustra o grau de dificuldade experienciado pelos jogadores em diferentes níveis de familiaridade com o gênero de plataforma 2D, como ilustrado na Figura 38 abaixo:

Figura 38 - Gráfico comparando a dificuldade enfrentada por jogadores em diferentes níveis de conhecimento prévio com o gênero

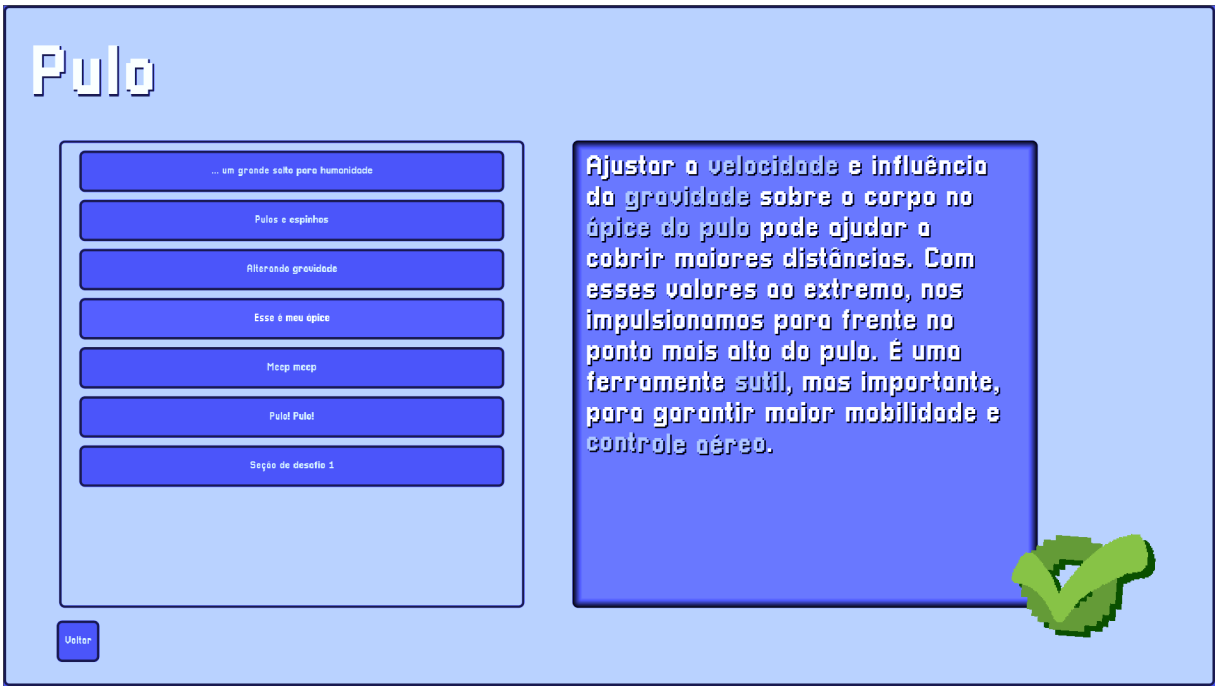


Fonte: Autor, 2024.

Por esse gráfico, é possível observar um grau de alta dificuldade constante em todos os níveis de contato prévio com o gênero. Com isso, é possível inferir que o jogo tende a se tornar mais fácil, porém, uma parcela significativa dos jogadores enfrentaram barreiras para conclusão dos desafios – mesmo aqueles que apresentam maior contato com este gênero de jogos. Isoladamente, esse fato não apresenta valor significativo, porém, no contexto no qual está inserido, pode também representar um bloqueio para o aprendizado e inclusão de jogadores (alunos) para os temas abordados nas fases e também futuras seções.

Um dos elementos mais importantes presentes no processo de aprendizagem do método PBL é a descrição de uma *Fuzzy Situation* que precede a pesquisa para solução de um problema apresentado. Esse elemento foi traduzido para o jogo como uma descrição presente na seleção de cada fase. Essa descrição apresenta vagamente o problema, como o exemplo apresentado na Figura 39:

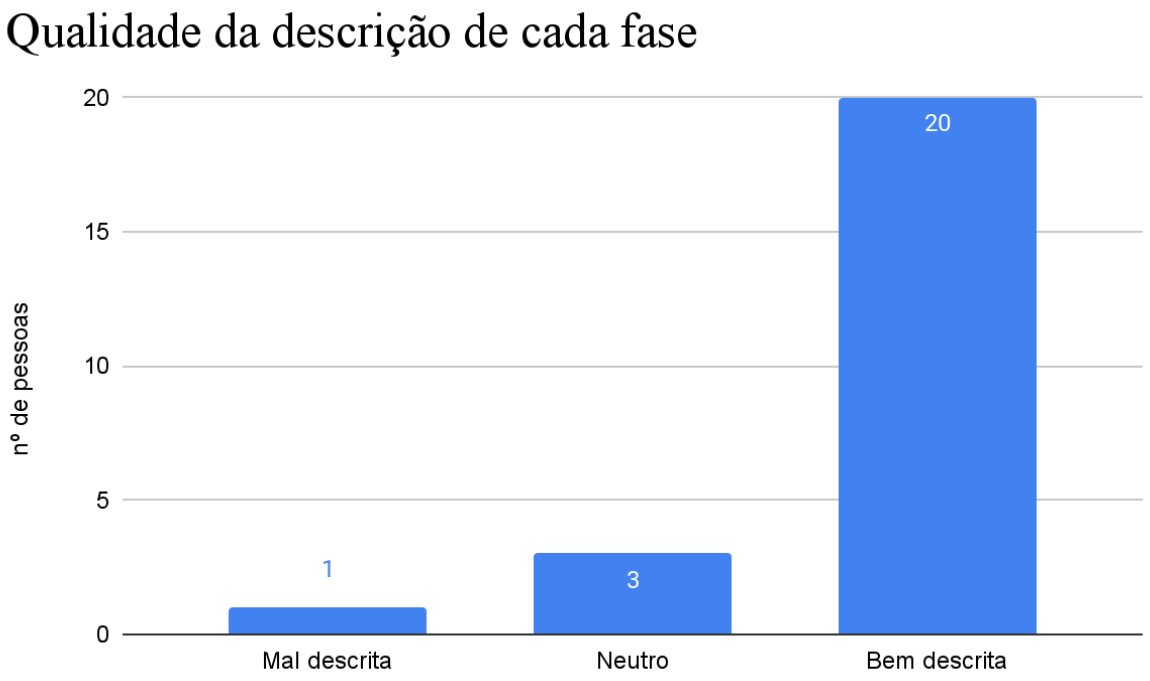
Figura 39 - Tela de seleção de fases com descrição da fase selecionada à direita



Fonte: Autor, 2024.

A recepção dessa ferramenta de ensino foi muito bem recebida pelos respondentes, eles avaliaram as descrições como compreensivas, como mostrado na Figura 40 abaixo:

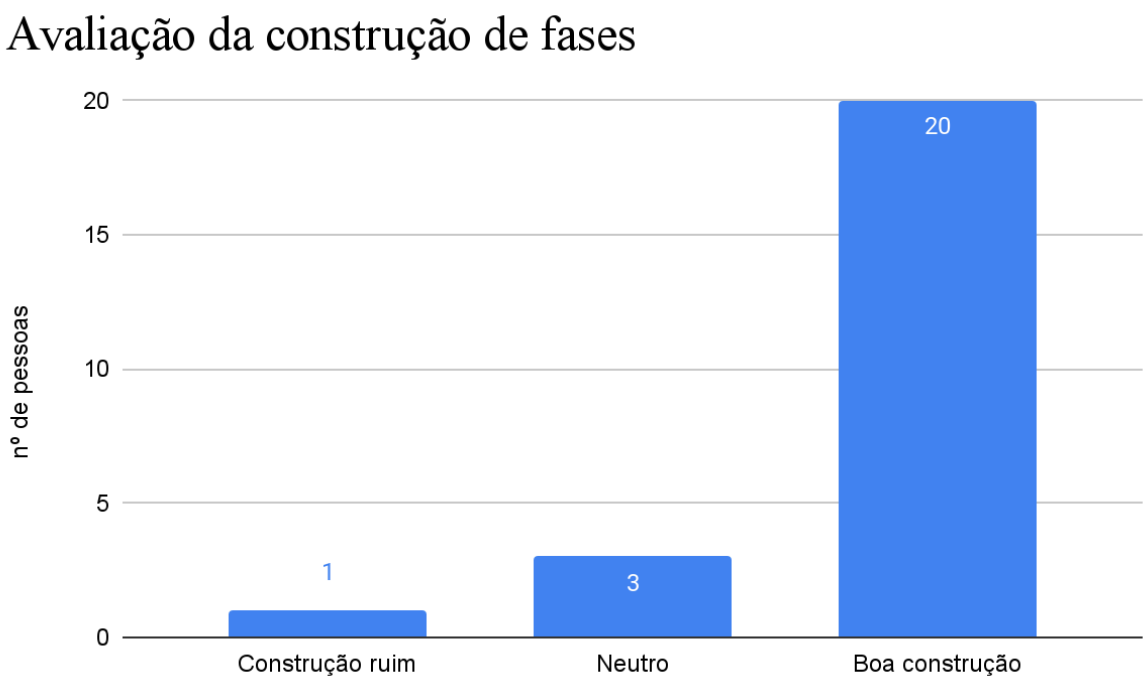
Figura 40 - Gráfico de avaliação de qualidade da descrição de fases



Fonte: Autor, 2024.

É importante também que o jogador, além de conhecer o problema, disponha de dicas e orientação suficientes para resolver os desafios apresentados. Para isso, é possível guiar o jogador implicitamente para o objetivo com uma boa construção de níveis, onde o fluxo de movimento e os próximos passos a serem dados estejam sempre à vista ou sejam dedutíveis pelo jogador. O próximo gráfico, ilustrado na Figura 41, avalia a construção das fases de acordo com os respondentes:

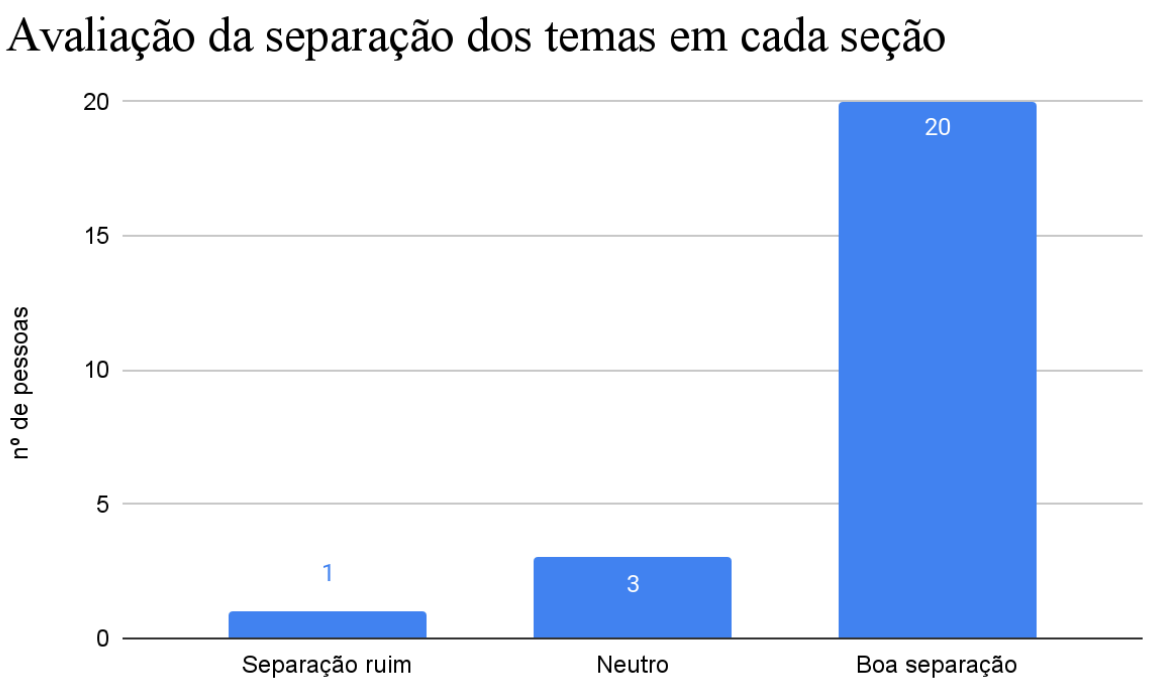
Figura 41 - Gráfico de avaliação de qualidade da construção de fases



Fonte: Autor, 2024.

Para melhor compreensão do aluno, é importante que a progressão dos assuntos abordados sigam uma lógica que o permita realizar tarefas complexas e inferir soluções com base em conhecimentos coletados até então. Isso posto, é necessário que o aluno (jogador) possa explorar diferentes temas com graus de dificuldade progressivos, para que a solução de desafios se torne mais imersiva e acessível. O gráfico ilustrado na Figura 42 coleciona os dados sobre a avaliação de separação de temas no jogo apresentado, esse dado reflete também a progressão de complexidade dentro do jogo:

Figura 42 - Gráfico de avaliação de qualidade da separação de temas e seções



Fonte: Autor, 2024.

6 CONCLUSÃO

Com a crescente demanda por jogos de plataforma 2D no cenário de desenvolvimento de jogos, desenvolvedores podem se perder ao se depararem com a grande quantidade de opções para construir uma movimentação envolvente para o jogador e que traduza o objetivo desejado no jogo. Para auxiliá-los, a implementação de um jogo como ferramenta de estudo de movimentação em jogos de plataforma pode ser uma solução viável para o problema de apresentar tantos conceitos sobre um tema tão complexo.

O desenvolvimento do jogo No Controle, produto estudado neste artigo, é introduzido neste contexto. No Controle é um jogo que compila muitos métodos de movimentação do personagem e explora junto ao jogador as variáveis utilizadas para criar um personagem. O jogo apresenta conceitos e desafios por meio de fases construídas para explorar aspectos específicos do personagem criado.

Para isso, o método de ensino PBL é utilizado a fim de entregar mais autonomia para o jogador e, por meio de sugestões e problemas desenhados em cada fase, fazer com que o jogador (aluno) consiga aprender os mecanismos por trás do movimento criado por ele.

Para avaliar a praticidade do modelo apresentado, uma pesquisa foi conduzida. Os resultados apontam que, embora o protótipo tenha sido bem recebido, mudanças precisam ser feitas para aperfeiçoar as próximas versões do projeto. O jogo se mostrou consistente, apresentando bem os conceitos que pretendia, mas a alta dificuldade para resolução dos problemas apresentados foi um fator persistente na avaliação.

Por conta disso, vê-se necessário reavaliar a construção de fases nas próximas versões do projeto. Como a interação do jogador com o mundo acompanha também seu aprendizado neste jogo, faz-se imperativo criar desafios mais acessíveis para todos os jogadores, dividindo ou criando fases menores que abordem temas recorrentes para revisão e sedimentação do conhecimento.

Muitos ajustes também fazem-se necessários para aperfeiçoar a interação do jogador com as ferramentas dispostas para ele durante o jogo. Como a interface de jogo é um dos principais meios de acesso do jogador para expressão e compreensão de conceitos apresentados, é necessário que a experiência de usuário seja uma prioridade, criando um ambiente acessível, intuitivo e seguro para experimentação e aprendizagem.

REFERÊNCIAS

ADÃES, Luis; PINTO, Paulo; FERREIRA, Tiago. **Jogo de Plataformas 2D:** Aprendizagem baseada na resolução de problemas. 2015. Monografia - Faculdade de Ciências, Universidade do Porto, Porto, 2015.

AGUIAR, Bernardo; CORREIA, Walter; CAMPOS, Fábio. **Uso da Escala Likert na Análise de Jogos.** In: SBC - Proceedings of SBGames 2011 Arts & Design Track - Short Papers, Universidade Federal de Pernambuco, 2011.

ANDRADE, Alan Jones Faria. **Criação de um jogo 2D de plataforma com Unity.** 2020. Trabalho de Graduação (Curso Superior em Análise de Desenvolvimento de Sistema). Faculdade de tecnologia " Dr. Thomaz Novelinho", franca, 2020.

BARROWS, H. S. **A taxonomy of problem-based learning methods.** [S. l.: s. n.], Novembro 1986.

BROWN, Mark. **Platformer Toolkit.** In: Itch. [S. l.], 7 jun. 2022. Disponível em: https://gmtk.itch.io/platformer-toolkit. Acesso em: 2 dez. 2023.

CAPCOM (Japão). **DuckTales**, out. 1989. Vídeo game.

CAPCOM (Japão). **Mega Man**, 17 dez. 1987. Vídeo game.

EXTREMELY OK Games (Canadá). **Celeste**, 25 jan. 2018. Vídeo game.

GODOT Engine. [S. l.], 4 dez. 2023. Disponível em: https://godotengine.org. Acesso em: 3 dez. 2023.

GOOGLE. **Google Forms.** Disponível em: https://www.google.com/forms/about. Acesso em: 27/10/2023.

HUNICKE, Robin; LEBLANC, Marc; ZUBEK, Robert. **MDA**: A Formal Approach to Game Design and Game Research. [S. l.]: Northwestern University, 2004.

ID TECH. **10 Types of Platforms in Platform Video Games.** [S. l.], 22 out. 2012. Disponível em: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games. Acesso em: 7 dez. 2023.

IMPROVE your Platformer's Jump (and Wall Jump) | Unity. Direção: Dawnosaur. Intérprete: Dawnosaur. Roteiro: Dawnosaur. Gravação de Dawnosaur. Youtube: [s. n.], 2022. Disponível em: https://www.youtube.com/watch?v=2S3g8CgBG1g. Acesso em: 13 dez. 2023.

IMPROVE Your Platformer with Forces | Examples in Unity. Direção: Dawnosaur. Intérprete: Dawnosaur. Roteiro: Dawnosaur. Gravação de Dawnosaur. Youtube: [s. n.], 2021. Disponível em: https://www.youtube.com/watch?v=KbtcEVCM7bw. Acesso em: 13 dez. 2023.

KIM, Bohyum. **Understanding Gamification**. Library Technology Reports, [s. l.], v. 51, ed. 2, fev/mar 2015.

KONAMI (Japão). **Castlevania**, 26 set. 1986. Vídeo game.

LEITE , P. S.; MENDONÇA , V. G.. **Diretrizes para Game Design de Jogos Educacionais**. Art & Design Track, SBGames, 2013.

MORATORI, P. B. **Por que Utilizar Jogos Educativos no Processo de Ensino Aprendizagem?**. Orientador: Dr. Fábio Ferrentini Sampaio. 2003. 33 p. Trabalho de conclusão (Mestrado de Informática aplicada à Educação) - UFRJ, Rio de Janeiro, 2003.

NINTENDO (Japão). **Metroid**, 6 ago. 1986. Vídeo game.

NINTENDO (Japão). **Super Mario Bros**, 13 set. 1985. Vídeo game.

NOEMÍ, Peña-Miguel; MÁXIMO, Sedano Hoyuelos. **Educational Games for Learning**. Universal Journal of Educational Research, Bilbao, Espanha, p. 230-238, 2014.

PLOMP, Tjeerd; NIEVEEN, Nienke; NONATO, Emanuel; MATTA, Alfredo. **Pesquisa-aplicação em educação**. Tradução de Emanuel do Rosário Santos Nonato. 1. ed. São Paulo: Artesanato Educacional, 2018. (Série Tecnologia Educacional; 20). Título original: *Educational Design Research*. ISBN 978-85-64803-20-6.

PRODUÇÃO DE JOGOS (Brasil). **Jogo de plataforma:** 7 técnicas que deixam o jogo bom de jogar. [S. l.], 2022. Disponível em: https://producaodejogos.com/jogo-de-plataforma-7-tecnicas/. Acesso em: 12 dez. 2023.

SAVERY, John R.; DUFFY, Thomas M. **Problem Based Learning:** An Instructional Model and Its Constructivist Framework. In: WILLSON, B. Constructivist Learning Environments: Case Studies in Instructional Design. 5. ed. Englewood Cliffs, NJ: Inc. Educational Technology Publications, 1996. v. 35, p. 31-38.

SEGA (Japão). **Sonic the Hedgehog**, 1991. Vídeo game.

TEAM Cherry (Austrália). **Hollow Knight**, 24 fev. 2017. Vídeo game.

TEAM Meat (Estados Unidos). **Super Meat Boy**, 20 out. 2010. Vídeo game.

UNITY Engine. [S. l.], 4 dez. 2023. Disponível em: https://unity.com/pt. Acesso em: 2 dez. 2023.

WAGAR, Celia. **5 Games, 5 Jumps.** In: CELIA ALEXIS WAGAR‘S CRITPOINTS. [S. l.], 18 maio 2015. Disponível em: https://critpoints.net/2015/05/18/5-games-5-jumps/. Acesso em: 11 dez. 2023.

WATSON, William R. **PBL as a Framework for Implementing Video Games in the Classroom**. In: INTERNATIONAL Journal of Game-Based Learning. [S. l.]: IGI Global, Janeiro - Março 2012. cap. 2, p. 77-89.