UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO

CAIO VILQUER CARVALHO - 15444705

MARCELLO BRAGA DE OLIVEIRA - 16887628

SAMUEL DAMASCENO PEREIRA CALDEIRA - 14607145

GRUPO S

A Fúria do Minotauro

Facilitando a coordenação fina

São Paulo

2025

1. INTRODUÇÃO

Visão Geral do Jogo 'A Fúria do Minotauro'

A Fúria do Minotauro é concebido como um jogo sério (serious game), projetado com um propósito fundamentalmente terapêutico e focado no desenvolvimento de habilidades motoras. Este projeto transcende o entretenimento convencional ao se posicionar como uma ferramenta interativa. O jogo engaja os jogadores em atividades cuidadosamente elaboradas para alcançar objetivos específicos de desenvolvimento.

A natureza de "jogo sério" não é uma escolha acidental. Tradicionalmente, exercícios terapêuticos ou de desenvolvimento de habilidades podem ser percebidos como repetitivos ou puramente clínicos, o que pode levar à desmotivação, especialmente em públicos mais jovens ou com necessidades específicas. Os jogos, por outro lado, são inerentemente desenhados para serem envolventes e motivadores. Ao adotar o formato de jogo sério, 'A Fúria do Minotauro' busca transferir esse engajamento intrínseco dos jogos para o domínio terapêutico. Esta abordagem visa aumentar a adesão do usuário às atividades propostas, o que, por sua vez, pode potencializar os resultados terapêuticos e de aprendizado.

Documentação interativa

Para uma experiência de aprendizado mais dinâmica e uma consulta rápida a todas as funcionalidades, mecânicas e detalhes técnicos do jogo, preparamos um manual interativo completo. **Acesse agora: https://minotauro-doc.vilquer.dev**

Propósito e Objetivos do Jogo

O objetivo primário do 'A Fúria do Minotauro' é facilitar a melhoria da coordenação motora fina, com um enfoque particular em indivíduos com Transtorno do Espectro Autista (TEA). Esta habilidade é essencial para a autonomia e qualidade de vida, e o jogo se propõe a ser um instrumento para seu aprimoramento.

Além deste objetivo central, o projeto visa alcançar metas secundárias igualmente importantes, como o aumento da precisão dos movimentos, o

desenvolvimento do controle motor e a promoção de uma experiência de aprendizado que seja simultaneamente motivadora e gratificante para o usuário. Para tal, o jogo combina elementos de aprendizado com tecnologia interativa, buscando criar uma ferramenta que seja não apenas acessível, mas também eficaz em seus propósitos terapêuticos.

Escopo deste Documento

Este documento tem como finalidade fornecer uma visão técnica abrangente e, na medida do possível, atualizada do jogo 'A Fúria do Minotauro'. Serão cobertos aspectos como a motivação subjacente ao projeto, especialmente no que tange aos benefícios para pessoas com TEA, a descrição de suas funcionalidades (conforme o projeto original), a arquitetura de software e hardware, e as instruções para instalação e execução.

2. MOTIVAÇÃO (COM FOCO EM TEA)

Desafios da Coordenação Motora Fina em Indivíduos com TEA

A coordenação motora fina representa uma habilidade fundamental para a autonomia e qualidade de vida, sendo frequentemente uma área de desafio para indivíduos com Transtorno do Espectro Autista (TEA). Esta competência é crucial para a execução de uma vasta gama de atividades da vida diária, tais como escrever, utilizar talheres de forma adequada, abotoar roupas e manipular pequenos objetos com precisão. Dificuldades nesta área podem impactar significativamente a independência e a participação em atividades escolares, sociais e de autocuidado.

Como 'A Fúria do Minotauro' Aborda Esses Desafios

'A Fúria do Minotauro' propõe uma abordagem lúdica e interativa para o treinamento da coordenação motora fina. A mecânica central do jogo envolve guiar um anel condutor através de um labirinto metálico, exigindo que o jogador evite o contato entre o anel e as bordas do percurso. Esta tarefa, aparentemente simples, demanda concentração, controle preciso dos movimentos e feedback constante entre a ação e a percepção.

Para manter o engajamento e promover o desenvolvimento contínuo, o jogo incorpora um sistema de dificuldade progressiva. Isso é alcançado através da utilização de servomotores que movimentam seções do labirinto, introduzindo padrões de movimento pré-estabelecidos. Essa dinâmica obriga o jogador a adaptar seus gestos e a refinar sua destreza de maneira contínua. Adicionalmente, um sistema de feedback sensorial, com estímulos auditivos e visuais em tempo real, é empregado para guiar o jogador, fornecer reforço positivo e manter a motivação.

A ênfase no "feedback sensorial em tempo real" e nos "estímulos auditivos e visuais" é particularmente pertinente quando se considera o público-alvo de indivíduos com TEA. Pessoas no espectro autista frequentemente apresentam um processamento sensorial atípico, podendo ser hipo ou hipersensíveis a determinados estímulos. Um feedback claro, imediato e multimodal (auditivo, visual e, implicitamente, tátil pela manipulação do anel) pode ser crucial para o aprendizado e engajamento desse público. A concepção do jogo leva em conta a necessidade de um ambiente de aprendizado que forneça informações sensoriais consistentes e compreensíveis, o que pode ser mais eficaz do que abordagens com feedback limitado ou menos direto.

Fundamentação em Pesquisas

A concepção do jogo 'A Fúria do Minotauro' (originalmente "Labirinto Sensorial Tecnológico") encontra respaldo em estudos científicos que investigam o uso de jogos e tecnologia no desenvolvimento de habilidades em crianças com TEA. O documento de projeto cita especificamente:

- **"AutiBots: Jogo digital educacional para o desenvolvimento cognitivo e motor de crianças com Transtorno do Espectro Autista"**: Este estudo sugere que jogos digitais podem ser ferramentas eficazes para o desenvolvimento de crianças com TEA.
- **"Gesture-based Video Games to Support Fine-Motor Coordination Skills in Children with Autism Spectrum Disorder"**: Pesquisas nesta linha indicam que intervenções baseadas em jogos baseados em gestos podem contribuir significativamente para o aprimoramento das habilidades de coordenação motora

fina.

- **"A case study of gesture-based games in enhancing the fine motor skills of children with autism spectrum disorders"**: Este estudo de caso concluiu que a implementação de jogos interativos resultou em progresso mensurável na capacidade das crianças em realizar movimentos de precisão, reforçando a validade de intervenções direcionadas e tecnologicamente assistidas.

Estas referências sublinham a premissa de que a tecnologia, quando aplicada de forma lúdica e direcionada, pode oferecer resultados promissores no campo terapêutico e educacional para indivíduos com TEA.

3. DESCRIÇÃO DAS FUNCIONALIDADES

Mecânica Central do Jogo

A essência do jogo reside em conduzir um anel condutor ao longo de um percurso definido em um labirinto metálico, com o desafio de não encostar o anel nas bordas do labirinto.

- **Detecção de Contato:** O contato entre o anel e o labirinto é detectado eletronicamente. O labirinto é conectado ao terminal GND (terra) do Arduino, enquanto o fio do anel é conectado a uma entrada digital do microcontrolador. Para evitar estados flutuantes e garantir uma leitura precisa, um sistema de pull-up é implementado na entrada digital. Um toque resulta em uma queda de tensão que é interpretada como uma colisão.
- **Sistema de "Vidas":** Em cada nível, o jogador dispõe de três "vidas". Cada vez que o anel toca o labirinto (uma "morte"), o jogador perde uma vida. Esta ocorrência é sinalizada por meio de áudios específicos ("Duas vidas restantes", "Uma vida restante") e por um feedback visual através de um LED RGB: a cor amarela indica a perda de uma vida, enquanto a cor vermelha sinaliza que o jogador deve retornar ao início do nível. A conclusão bem-sucedida do nível é indicada pela cor verde no LED RGB.

Progressão e Dificuldade

A complexidade do jogo aumenta progressivamente à medida que o jogador

avança. O principal mecanismo para o incremento da dificuldade é a introdução de padrões de movimento dinâmicos nas seções do labirinto, que são controladas por servomotores.

Diferente de um sistema com posições fixas, o firmware do Arduino agora armazena internamente (na memória de programa, ou Flash) uma série de sequências de movimento para cada nível. Cada passo de uma sequência define um **ângulo** alvo para os servos e um tempo de **espera** (`hold`) em milissegundos.

O Arduino executa esses padrões de forma autônoma, movendo os servos gradualmente até o ângulo definido e aguardando no local pelo tempo especificado antes de prosseguir para o próximo passo. Uma máquina de estados interna garante que o tempo de espera só comece a contar após os servos terem fisicamente alcançado sua posição, garantindo que os padrões sejam respeitados em qualquer velocidade de jogo selecionada nas opções de acessibilidade. Essa arquitetura torna o desafio consistente e robusto, eliminando a necessidade de comunicação complexa de padrões via JSON.

Funcionalidades Especiais (Quick Time Events - QTEs e Power-ups)

Para adicionar um elemento de variabilidade e recompensa, o jogo incorpora um sistema de power-ups e Quick Time Events (QTEs):

- Existe uma chance aleatória de o jogador receber a oportunidade de adquirir um poder especial durante o jogo. Exemplos de power-ups incluem o ganho de uma vida extra, a redução do tempo total em 10 segundos, ou a pausa dos servomotores por 5 segundos.
- Para obter o poder especial, o jogador deve completar com sucesso um QTE. Este evento consiste em acertar uma sequência de apertos de botões, que são acoplados à manopla do anel condutor. A sequência a ser replicada é indicada visualmente de forma interativa - botão a botão - ao jogador por meio de um LED RGB também localizado na manopla (por exemplo, Azul para o botão de Cima, Vermelho para o botão de baixo). Observação, ao iniciar o QTE, os servos serão

pausados para que o jogador possa focar nos botões

Sistema de conquistas e recompensas

Foi implementado um sistema de conquistas (“achievements”) para reforçar positivamente marcos importantes e incentivar determinados comportamentos do jogador. Há diversos troféus virtuais que podem ser desbloqueados ao cumprir requisitos específicos, por exemplo: completar um nível sem nenhuma colisão, concluir todos os níveis do jogo, jogar um determinado número de vezes, ou finalizar um nível abaixo de um tempo limite. As conquistas escolhidas para o jogo foram:

- “Fio de Ariadne” – concedida ao completar seu primeiro nível com sucesso
(primeiro marco no jogo, simbolizando a descoberta do caminho dentro do labirinto).
- “Coragem de Teseu” – concedida ao completar um nível sem perder nenhuma vida
(demonstrando controle motor preciso e atenção constante do início ao fim).
- “Despertar da Fúria” – concedida ao completar um nível após perder 2 vidas
(mostra resiliência: o jogador se recupera de erros e conclui o desafio mesmo sob pressão).
- “Domador do Labirinto” – concedida ao completar 5 fases diferentes
(indica domínio progressivo dos diversos percursos e mecânicas do jogo).
- “Renascido” – concedida ao tentar a mesma fase 7 vezes consecutivas
(reforça a perseverança: insistir no treino até superar a dificuldade).
- “Herói de Atenas” – concedida ao completar todas as fases do jogo
(simboliza a jornada completa de Teseu: vencer o Minotauro e sair do labirinto).
- “Velocista Olímpico” – concedida ao completar um nível em menos de 10 segundos
(testa agilidade motora excepcional e tomada de decisão rápida).
- “Mestre dos Servos” – concedida ao vencer uma fase com a movimentação aleatória dos motores ativada
(prova adaptação a condições imprevisíveis e coordenação apurada).
- “Pegadas de Bronze” – concedida ao jogar o jogo 50 vezes

(reconhece comprometimento a longo prazo com o treinamento e prática contínua).

Interfaces do Jogo

O jogo apresenta diversas telas para gerenciar a interação do usuário, desde a seleção de perfil até a visualização de desempenho.

- **Tela de Escolha de Usuário:**
  - Apresenta uma lista de usuários previamente cadastrados.
  - Permite selecionar um usuário existente para continuar uma sessão de jogo.
  - Oferece a funcionalidade de criar um novo usuário mediante a digitação do nome e pressão da tecla ENTER.
  - Permite excluir um usuário existente (através de um botão "Del"), o que remove todo o seu progresso, armazenado em um arquivo JSON.
- **Tela de Menu Principal:**
  - Exibe as ações principais disponíveis após a seleção ou criação de um usuário.
  - Opções incluem: "Iniciar Jogo", "Ver Desempenho", "Rejogar Nível", "Voltar" (para a tela de escolha de usuário) e "Sair" do programa.
  - Inclui recursos de acessibilidade importantes: "Ativar Escala de Cinza" e "Desativar Som".
- **Tela de Jogo (Labirinto Sensorial):**
  - É a tela principal de interação durante a atividade com o labirinto.
  - Exibe informações cruciais em tempo real: nome do usuário, nível atual, número de vidas restantes e tempo decorrido.
  - Realiza a leitura das colisões detectadas pelo Arduino.
  - Permite ao jogador retornar ao menu principal a qualquer momento através de um botão "Voltar".
- **Tela de Conclusão de Nível:**
  - Aparece quando o jogador consegue completar o percurso do nível atual.

- Exibe uma mensagem de parabéns e o tempo gasto para concluir o nível.
- Oferece opções para "Avançar Nível" (prosseguir para o próximo desafio), "Rejogar Nível" (tentar novamente o mesmo nível) ou "Voltar" ao menu principal.

- **Tela de Perda de Todas as Vidas:**
  - É acionada quando o jogador esgota todas as suas vidas dentro de um nível.
  - Informa que o jogador deve recomeçar o nível desde o início.
  - Apresenta opções como "Rejogar Nível" ou "Voltar ao Menu".
- **Tela de Conclusão do Jogo:**
  - Exibida caso o jogador consiga vencer o último nível do jogo.
  - Apresenta uma mensagem comemorativa de conclusão total do jogo.
  - Oferece um botão para "Voltar" ao menu principal.
- **Tela de Desempenho:**
  - Permite ao jogador visualizar seu progresso e estatísticas.
  - Mostra o nível atual alcançado pelo jogador e seu melhor tempo geral no jogo.
  - Permite navegar entre os diferentes níveis que o usuário já tentou, exibindo o melhor tempo de conclusão para o nível selecionado.
  - Apresenta um histórico das últimas seis tentativas para o nível selecionado, detalhando tempo, vidas restantes, data e se o nível foi concluído ou não.
  - Apresenta gráficos detalhados de: Tempo médio por nível, tempo por tentativa em cada nível, colisões médias por nível, colisões por tentativa em cada nível,
- **Tela de Conquistas:**
  - Exibe todos os troféus virtuais disponíveis no jogo, mostrando claramente quais já foram desbloqueados e quais ainda faltam.
  - Cada conquista aparece como um ícone: colorido se conquistado, semitransparente se bloqueado.
  - Oferece um botão “Voltar” para retornar ao Menu Principal, mantendo

consistência visual com os demais botões do jogo.

- **Tela de Rejogar:**
  - Oferece ao jogador a possibilidade de selecionar e refazer níveis anteriores que já foram desbloqueados.
  - Lista cada nível desbloqueado como um botão clicável. Ao selecionar um nível, o jogo inicia naquela fase.
  - Inclui um botão "Voltar" para retornar ao menu principal.
- **Tela de História dos Personagens:**
  - Apresenta a biografia resumida e o papel narrativo de cada personagem do jogo (Teseu, Ariadne, Minotauro, etc.) em painéis individuais.
  - Cada painel contém o retrato ilustrado do personagem, um breve texto contextualizando sua importância na mitologia e no enredo do jogo, além de falas icônicas usadas durante a jogatina.
  - Inclui navegação por seta ou botões “Próximo/Anterior” para percorrer os personagens, com indicador de qual página está sendo visualizada.
  - Possui um botão “Voltar” que retorna diretamente ao Menu Principal, preservando o estado de áudio e escala de cinza definidos pelo usuário.
  - Serve como elemento de imersão narrativa e reforço pedagógico, permitindo que o jogador (ou terapeuta) leia e relembre a história associada a cada fase e conquista.
- **Opção de Escala de Cinza:**
  - Um recurso de acessibilidade notável é a opção de converter todas as cores dos botões e das fontes para seus correspondentes em escala de cinza. Esta funcionalidade visa atender indivíduos no espectro autista com alta sensibilidade a cores, tornando a interface visualmente menos estimulante e mais confortável.
- **Elemento narrativo e orientação temática**:
  - Como parte do engajamento lúdico, o jogo incorpora uma leve narrativa temática baseada na mitologia grega. O jogador é convidado a se imaginar no papel de um herói navegando um labirinto (como Teseu no labirinto do Minotauro). Após superar certos marcos (como concluir o

primeiro nível), o jogo apresenta diálogos orientativos com um personagem guia (Teseu), oferecendo encorajamento e dicas. Por exemplo, há um diálogo inicial após o nível 1 em que Teseu parabeniza o jogador e o prepara para desafios maiores a seguir (isso foi integrado em forma de caixa de texto) . Esses elementos narrativos são sucintos para não sobrecarregar a atenção, mas adicionam contexto e sentido de progressão épica à atividade motora, tornando-a mais atrativa. Embora não influenciem a jogabilidade diretamente, esses momentos narrativos contribuem para a motivação e imersão, conectando o desenvolvimento motor a uma história de superação (vencer o “Minotauro” simbolicamente ao terminar o jogo).

A inclusão de opções como "Escala de Cinza" e "Desativar Som" no Menu Principal é um forte indicativo de um design que considera ativamente as sensibilidades sensoriais que podem estar presentes em indivíduos com TEA. Estes não são recursos triviais ou universalmente presentes em jogos, e sua implementação sugere uma preocupação genuína com as necessidades específicas do público-alvo. Permitir que o usuário exerça controle sobre esses estímulos (visuais e auditivos) possibilita uma personalização da experiência de jogo, o que pode torná-la mais confortável, acessível e, consequentemente, prolongar o tempo de engajamento e a eficácia da intervenção terapêutica.

Opções de Acessibilidade e Personalização

Reconhecendo que cada jogador possui necessidades e sensibilidades únicas, "A Fúria do Minotauro" foi projetado com um menu de acessibilidade completo, permitindo que a experiência de jogo seja profundamente personalizada. O objetivo é remover barreiras e garantir que o foco permaneça na diversão e no desenvolvimento de habilidades, em um ambiente confortável para o usuário.

Todas as configurações são salvas por perfil de usuário, garantindo que a experiência customizada seja mantida entre as sessões de jogo.

Abaixo, um detalhamento de cada categoria e suas respectivas opções:

1. Configurações de Áudio Permite um controle granular sobre o ambiente sonoro do jogo.

- **Volume da Música:** Ajusta o volume da trilha sonora de fundo. Pode ser reduzido ou desativado para jogadores que se distraem com música contínua.
- **Volume dos Efeitos Sonoros:** Controla o som de eventos como colisões, cliques de botão e sucesso de nível.
- **Volume das Vozes:** Ajusta o volume dos diálogos dos personagens e das dicas narradas, garantindo que a orientação possa ser ouvida claramente ou silenciada, conforme a preferência.
- **Ativar Som (Geral):** Um botão mestre para ligar ou desligar todos os sons do jogo de uma só vez.

2. Configurações de Imagem Focadas em reduzir a sobrecarga de estímulos visuais.

- **Reduzir Efeitos Visuais:** Desativa ou diminui a intensidade de efeitos de *flash* e tremores de tela que ocorrem durante o jogo, um recurso essencial para jogadores com fotossensibilidade.
- **Modo Escala de Cinza:** Converte toda a interface do jogo para tons de cinza. Esta é uma funcionalidade crucial para jogadores com alta sensibilidade a cores, tornando a experiência visualmente mais calma e menos distrativa.

3. Configurações de Jogabilidade Permitem ajustar o nível de desafio e a pressão da performance.

- **Número de Vidas:** O jogador pode escolher começar cada nível com 1 a 5 vidas. Aumentar o número de vidas pode reduzir a ansiedade e incentivar a persistência.
- **Tempo de Imunidade após Colisão:** Define um pequeno intervalo (em milissegundos) após uma colisão durante o qual novos toques não são registrados. Aumentar este valor pode ajudar jogadores que têm dificuldade em retirar o anel rapidamente após um erro, evitando penalidades duplicadas.
- **Modo Prática:** Uma opção fundamental que desativa o cronômetro e as

vidas. Neste modo, o jogador pode explorar o labirinto, aprender os padrões dos servos e treinar seus movimentos sem qualquer tipo de pressão, focando unicamente na mecânica motora.

4. Configurações de Feedback e Controle Permitem customizar como o jogo se comunica e como os desafios se apresentam.

- **Velocidade do Servo:** Permite escolher entre "lento", "normal" e "rápido" para o movimento das paredes do labirinto, ajustando diretamente o nível de dificuldade do desafio motor.
- **Canal de Feedback:** Oferece ao jogador o poder de escolher como receber as informações do jogo:
  - `som`: Apenas feedback sonoro.
  - `cor`/`led`: Apenas feedback visual através dos LEDs.
  - `múltiplo`: Feedback sonoro e visual combinados (padrão).
- **Ativar Eventos de Tempo Rápido (QTE):** Permite ligar ou desligar completamente a mecânica de QTEs. Desativar pode ser útil para jogadores que desejam focar exclusivamente no desafio do labirinto.

5. Configurações de Texto

- **Velocidade do Texto em Diálogos:** Ajusta a velocidade com que o texto dos diálogos aparece na tela, permitindo que jogadores com diferentes ritmos de leitura acompanhem a história confortavelmente.

Adicionalmente, o sistema de perfis de usuário e o registro detalhado de desempenho, que inclui histórico de tentativas, tempo gasto, vidas restantes e gráficos estatísticos, estabelece uma base sólida para o acompanhamento longitudinal do progresso do jogador. Esses dados, que são métricas diretas do desempenho na tarefa de coordenação motora fina, poderiam ser utilizados por terapeutas ou educadores para avaliar a evolução do usuário ao longo do tempo. Uma evolução natural e sofisticada para um jogo sério como este seria utilizar esses dados coletados para adaptar dinamicamente a dificuldade do jogo – por exemplo, ajustando a velocidade dos servomotores, a sensibilidade ao toque ou a

complexidade do traçado do labirinto – de forma a manter o jogador constantemente em sua zona de desenvolvimento proximal, ou seja, em um nível de desafio ótimo.

4. ARQUITETURA DO SOFTWARE

O software do jogo foi desenvolvido em Python 3, utilizando a biblioteca Pygame para gerenciamento de interface gráfica, eventos e multimídia. A arquitetura é modular e organizada em pastas, seguindo o princípio de separação de responsabilidades para facilitar a manutenção e extensões futuras . Os principais componentes de código e seu propósito são:

Módulo principal (`*main.py*`)

Ponto de entrada do programa. Este script inicializa o ambiente Pygame, configura a janela gráfica e em seguida carrega os módulos de jogo. Ele gerencia o loop principal da aplicação, que consiste em exibir a tela atual (por exemplo, menu, jogo, etc.), processar entradas do usuário (cliques, teclado) e atualizar a lógica conforme necessário. O `*main.py*` também é responsável por estabelecer a conexão serial com o Arduino: ao iniciar, ele lista as portas serial disponíveis (COMx no Windows, ttyACMx no Linux, etc.) e, através da tela de seleção de porta, permite ao usuário escolher aquela a que o Arduino está conectado (ou simular o jogo). Uma vez escolhida, o programa abre a comunicação serial (usando a biblioteca PySerial) e a mantém durante todo o jogo para troca de dados em tempo real. Em resumo, `*main.py*` orquestra as transições entre as telas de alto nível (chamando funções ou métodos que renderizam cada tela) e repassa para cada módulo especializado as tarefas específicas.

Pacote de telas (`*screens/*`)

Contém módulos Python separados para cada grupo de telas ou para cada tela principal da interface do jogo. Por exemplo, há um arquivo `*main_menu.py*` que define a lógica de desenho e interação do Menu Principal , outro `*characters_screen.py*` para a tela de seleção de usuário , e módulos para telas de gameplay e de resultados (`*level_complete.py*` , `*game_over.py*` , `*game_complete.py*` , `*achievements_screen.py*` , etc.) . Cada módulo de tela

tipicamente define uma função principal (por exemplo, `*tela_menu_principal(tela, usuario)*`), que recebe a superfície de janela Pygame e desenha os elementos necessários, além de tratar cliques nos botões. Em alguns casos, telas mais complexas podem ser implementadas como classes, mas majoritariamente optou-se por funções e uso de utilitários comuns para desenhar botões e textos. O pacote de telas permite que cada screen seja desenvolvida e testada isoladamente, e o `*main.py*` simplesmente chama a função adequada conforme a navegação do jogo. As telas comunicam entre si através de valores de retorno ou variáveis globais de estado – por exemplo, a função da tela de menu pode retornar uma string indicando a ação selecionada (“JOGAR”, “CONQUISTAS”, “SAIR”, etc.), e o loop principal então invoca a próxima tela conforme esse resultado . Assim, o fluxo de telas é controlado de maneira estruturada. As telas de jogo e de resultados também interagem com outros módulos (como o de jogo e o de conquistas) para atualizar o estado do jogo e registrar dados quando necessário.

Módulo de lógica do jogo (`*game/game.py*`)

Responsável por toda a lógica do gameplay do labirinto em si. Este módulo implementa a classe central do jogo, chamada de `*JogoLabirinto*`, a qual encapsula o estado de uma partida em andamento. Dentro dessa classe estão atributos como o usuário atual, nível atual, vidas restantes, número de colisões, etc., além de referências aos sistemas de apoio (como uma instância do sistema de conquistas e do áudio, passadas na inicialização) . A classe `*JogoLabirinto*` gerencia o loop específico do nível: inicia o nível (reseta contadores, envia comando de preparo ao Arduino, zera timer), então em cada iteração recebe dados do Arduino (via serial) indicando se houve colisão (`*self.sistema_conquistas.verificar_conquistas*` é chamado após atualizar os dados do jogo em cada colisão ou fim de nível) e atualiza o estado. Ela também monitora o tempo decorrido. Caso o Arduino indique game over (sensor de fim acionado ou vidas=0), a classe finaliza o nível e retorna os resultados (ex.: nível concluído ou falhado, tempo, etc.). Em essência, esse módulo conecta a interface física ao jogo lógico: lê os eventos de hardware (colisões, sensores) e aplica as regras do jogo (diminuir vidas, avançar progresso, etc.). Ele

também envia comandos de volta ao Arduino, por exemplo: ao iniciar um nível, envia sinal para reposicionar servos na posição inicial; durante o nível, envia comandos ocasionais para mudar servo de posição (implementando os movimentos aleatórios ou predefinidos do labirinto); e ao terminar, pode envia comando para o LED sinalizar algo (embora grande parte do feedback seja disparado diretamente pelo Arduino autonomamente, dependendo do firmware).

Módulo de comunicação serial (firmware Arduino)

Embora não seja parte do código Python, é um componente de software fundamental e autônomo. O Arduino executa um firmware (escrito em C/C++ usando a biblioteca `avr/pgmspace` para otimização de memória) que agora possui a lógica completa dos desafios físicos, garantindo alta performance e responsividade.

Principais Responsabilidades do Firmware:

1. **Gerenciamento Autônomo dos Servomotores:** Diferente da versão inicial, o firmware **não recebe mais padrões de movimento via JSON**. Todos os padrões de `ângulo` e tempo de espera (`hold`) para cada nível estão pré-gravados na memória Flash (PROGMEM) do Arduino. Isso elimina problemas de memória e latência na comunicação, tornando o movimento do labirinto mais confiável. O Arduino utiliza uma máquina de estados (`MOVING`, `HOLDING`) para controlar os servos, garantindo que o tempo de espera de cada passo só comece após o servo ter fisicamente chegado à sua posição-alvo, corrigindo bugs de temporização em diferentes velocidades.

2. **Protocolo de Comunicação Reativo:** A comunicação, via Serial (9600 bps), foi otimizada para ser orientada a eventos. O Arduino agora age como a fonte da verdade para todos os eventos físicos, enviando mensagens curtas e claras que a interface Python interpreta:

    - `PLAYER_AT_START`: Enviado quando o sensor Hall de início é acionado.
    - `LEVEL_COMPLETE`: Enviado quando o sensor Hall de fim é acionado.
    - `COLLISION`: Enviado instantaneamente quando o anel toca o labirinto.
    - `BTN_C` / `BTN_B`: Enviados apenas quando um QTE está ativo, informando qual botão (Cima/Baixo) foi pressionado.
3. **Execução de Comandos da Interface:** O Python agora atua como um "maestro", enviando comandos simples para o Arduino, que os executa imediatamente:

    - **Comandos de Jogo:** NIVEL:{id} para carregar um nível e TERMINAR para encerrar uma partida.
    - **Comandos de QTE:** QTE_SHOW:{passo} (ex: QTE_SHOW:C) para instruir qual LED da manopla deve acender, e QTE_END para sinalizar o fim do evento.
    - **Comandos de Acessibilidade e Power-up:** DEBOUNCE:{ms}, SERVO:{velocidade}, FEEDBACK:{canal} e FREEZE:{estado} para configurar o comportamento do hardware em tempo real.

Esse design de responsabilidades garante que as interações críticas (detecção de colisão, leitura de botões) e o movimento complexo dos servos ocorram com latência mínima, diretamente no hardware. O Python, rodando em paralelo, cuida da lógica de jogo de alto nível (pontuação, vidas, interface gráfica), reagindo aos eventos que o Arduino envia, criando um sistema robusto e perfeitamente sincronizado.

Utilitários e outros módulos (`*utils/*`)

Para evitar repetição de código, vários utilitários foram implementados e armazenados na pasta utils . Dentre eles:

- **`*utils.drawing.py*` *:*** contém funções auxiliares para desenhar elementos na tela Pygame de forma consistente, como `*desenhar_texto*` (para renderizar textos com a fonte e cor padrão do jogo), `*desenhar_botao*` (que desenha botões retangulares estilizados (ou com background definido) e pode alterar sua aparência quando o mouse passa por cima) , etc. Também engloba a lógica de aplicar a conversão para escala de cinza nas cores quando necessário (função `*aplicar_filtro_cinza_superficie*` que adapta uma cor dada dependendo do modo de cor ativo) . Além disso, centralizou-se o cálculo de redimensionamento de textos e botões na função `*resize*` para suportar diferentes resoluções de tela de forma consistente. Esse módulo centraliza detalhes de UI, facilitando mudar a aparência do jogo em um só lugar.
- **`*utils.colors.py*` *:*** Foi complementado para suportar o modo de cor em tons de cinza, fornecendo cores convertidas conforme necessário para textos e botões.
- **`*utils.audio_manager.py*` *:*** implementa a classe `*AudioManager*` mencionada, responsável por carregar arquivos de áudio (músicas e efeitos) e disponibilizar métodos como `*play_sound(nome)*` e `*play_background()*` . Ele usa o mixer do Pygame internamente. Além disso, gerencia volumes (métodos

`*set_bg_volume(valor)*` para ajustar volume da música) e mantém um dicionário de sons carregados. Esse módulo é inicializado no começo do jogo, carrega todos os sons necessários (`*load_sounds()*`) , e um objeto `*audio_manager*` global é disponibilizado para outros módulos usarem. Por exemplo, no módulo de interface, ao desenhar um botão, se detectar evento de mouse-over, chama `*audio_manager.play_sound("hover")*` ; ao `clique, *play_sound("click")*` ; na lógica do jogo, ao existir colisão: `*play_sound("collision")*` ; ao existir vitória de nível: `*play_sound("success")*` ; etc. Dessa forma, o áudio fica acionado nos pontos desejados sem duplicar código de carregamento/execução de som em cada lugar.

- `***utils.achievements.py*` *:*** implementa o sistema de conquistas, contendo a classe `*SistemaConquistas*` e estruturas de dados para as conquistas disponíveis . Essa classe gerencia uma lista ou dicionário de conquistas definidas (chave, descrição, condição, status desbloqueado/não). Possui métodos como `*carregar_conquistas_usuario(usuario)*` e `*salvar_conquistas_usuario(usuario)*` para persistir conquistas em armazenamento (em arquivo JSON). Também implementa o método `*verificar_conquistas(usuario, dados_jogo)*` – este é chamado pelo módulo de jogo quando algum evento de interesse ocorre, passando os dados relevantes (p. ex. tempo do nível, vidas restantes, total de colisões naquela partida, nível atual, etc.). A função então verifica cada conquista: por exemplo, para “Coragem de Teseu”, checa se `*dados_jogo.colisoes == 0*` e se o nível foi concluído, concedendo a conquista se ainda não concedida. Cada conquista desbloqueada aciona uma notificação interna (armazenada numa lista de notificações pendentes). Outra função, `*desenhar_notificacao(tela)*` , é usada nas telas de conclusão para exibir visualmente as últimas conquistas desbloqueadas, caso houver . Ao final do nível, o sistema limpa notificações e salva progresso das conquistas do usuário. Integrar esse módulo no fluxo garante que conquistas sejam avaliadas sempre que o jogador terminar um nível ou jogo, e que nada seja esquecido de registrar
- `***utils.graphics.py*` *:*** implementa o subsistema de gráficos temáticos do jogo, reunindo funções decorativas e as classes de gráficos. `*GraficoBase*` fornece a

infraestrutura comum de desenho: cria uma superfície semi-transparente, aplica borda arredondada, título estilizado no tema grego, controla a animação de entrada e expõe o método `*desenhar(surface_destino)*`. `*GraficoLinha*` herda de `*GraficoBase*` para exibir a evolução temporal de métricas (p. ex. tempos de conclusão); converte valores de dados em coordenadas de tela, desenha eixos, grade, rótulos, linha principal com sombra e pontos de destaque, além de suportar animação progressiva (`*animacao_progresso*`) e visual “alvo” pontilhado até a animação completar. `*GraficoBarras*` também deriva de `*GraficoBase*`, apresentando comparações entre níveis em barras verticais coloridas; calcula largura, altura e espaços, desenha grade, rótulos nos eixos, contorno dourado, ranhuras horizontais em estilo de colunas e mostra valores numéricos quando a animação atinge 50 %. O módulo inclui ainda paleta de cores temáticas (vermelho-Minotauro, dourado-antigo, terracota etc.), função `*desenhar_colunas_gregas()*` para ornamentos laterais e utilidades de transformação de dados. Dessa forma, `*utils.graphics*` centraliza toda a lógica de renderização de gráficos animados e decorados, garantindo consistência visual com a estética mitológica do jogo e permitindo que outras telas gerem painéis estatísticos apenas instanciando as classes e alimentando-as com listas de dados, sem duplicar código de desenho.

Em suma, a arquitetura de software foi concebida para isolamento de funcionalidades (por exemplo, camadas de hardware vs lógica de jogo vs interface gráfica) e facilidade de extensão. Prova disso foi a inclusão tardia de features como modo escala de cinza e sistema de conquistas sem necessidade de modificar profundamente o núcleo – bastou anexar novos módulos e chamar nos pontos certos, graças à modularização prévia. Essa organização modular e uso de classes utilitárias melhorou a legibilidade e manutenção do código , além de tornar mais segura a integração de novos recursos sem introduzir bugs nas partes já funcionais.

5. ESPECIFICAÇÕES DE HARDWARE

A implementação do jogo 'A Fúria do Minotauro' requer um conjunto específico de componentes de hardware. Estes componentes são essenciais para a

interação física do jogador com o labirinto e para o feedback sensorial proporcionado pelo sistema.

Lista de Componentes de Hardware Requeridos

- **Controlador:** Um microcontrolador Arduino. Embora o modelo específico não seja explicitamente detalhado na lista de hardware, um diagrama elétrico no documento original exibe um "ARDUINO UNO", sugerindo este modelo como base.
- **Estrutura do Labirinto:** Uma estrutura metálica que define o percurso do labirinto. Esta estrutura é condutora para permitir a detecção de contato.
- **Interface do Jogador:** Um anel condutor, conectado a um cabo flexível, que o jogador manipula para navegar pelo labirinto.
- **Sensores:**
  - **Sensores de Efeito Hall Lineares:** Utilizados para detectar a proximidade do anel no ponto de início e no ponto de fim do percurso do labirinto.
  - **Ímã:** Um pequeno ímã deve ser posicionado no anel condutor para permitir a detecção pelos sensores de efeito Hall.
  - **Detecção de Contato:** O contato entre o anel e o labirinto é detectado de forma digital para máxima confiabilidade. O labirinto metálico é conectado ao pino Terra (GND) do Arduino, enquanto o fio do anel é conectado ao pino analógico A0, configurado como uma entrada digital. Um resistor de *pull-up* interno é ativado programaticamente no pino A0, mantendo-o em um estado lógico ALTO. Quando o anel toca a parede, o circuito se fecha para o Terra, o pino vai para o estado BAIXO, e uma colisão é registrada instantaneamente pelo firmware. Um "debounce" (tempo de imunidade) configurável via menu de acessibilidade previne registros múltiplos por um único toque.
- **Atuadores:**
  - **Micro Servomotores:** Responsáveis por movimentar seções do labirinto, alterando dinamicamente o desafio. Os servos serão alimentados com uma fonte externa para evitar oscilações de tensões no arduino, uma

alternativa caso seja necessário usar a alimentação do arduino seria usar um capacitor de desacoplamento próximo aos pinos de alimentação dos servos para estabilizar a tensão durante movimentos rápidos.

- **LED RGB (Status do Jogo):** Um LED multicor utilizado para fornecer feedback visual sobre o estado do jogo, como a perda de uma vida (amarelo), a necessidade de voltar ao início (vermelho) ou a conclusão bem-sucedida de um nível (verde).
- **LED RGB (Manopla para QTEs):** Um segundo LED RGB, localizado na manopla do anel, usado para indicar visualmente a sequência de botões a ser pressionada durante os Quick Time Events.
- **Alto-falante do Computador:** Os sinais sonoros são gerados pela aplicação Pygame e reproduzidos através do sistema de som do computador.

- **Entrada Adicional:**
  - **Botões na Manopla:** Dois botões acoplados à manopla do anel, utilizados pelo jogador para interagir durante os Quick Time Events.
- **Interface com Usuário e Processamento Central:**
  - **Computador:** Necessário para executar a aplicação Pygame, que gerencia a interface gráfica do usuário, a lógica do jogo e o registro do desempenho do jogador.

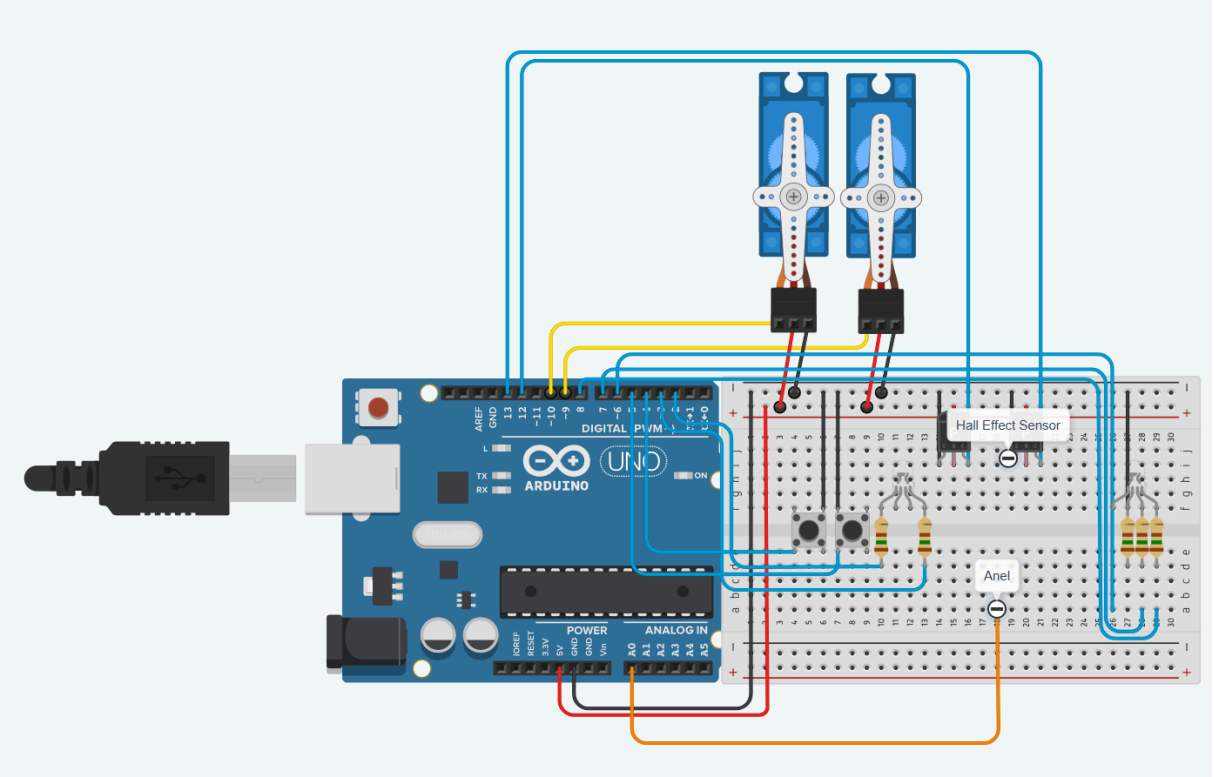
Em resumo, o hardware foi projetado para confiabilidade e responsividade. Os sensores garantem detecção precisa de eventos-chave (início, fim, colisão), enquanto os atuadores enriquecem o desafio motor. A integração com o Arduino provê a ponte necessária entre o mundo físico e o digital: todos componentes comunicam-se pela placa microcontroladora, que funciona como uma unidade de aquisição e controle em tempo real. O resultado é um aparato completo que materializa o jogo virtual em uma atividade concreta, mantendo ambos sincronizados.

Tabela 3: Especificações de Hardware

Componente	**Descrição/Modelo Específico**	**Função Principal no Jogo**
Microcontrolador	Arduino UNO	Unidade central de controle dos componentes eletrônicos
Estrutura do Labirinto	Metálica, com percurso customizado	Superfície de jogo e parte do sensor de contato
Anel Condutor	Metálico, conectado a um cabo	Elemento de interação manipulado pelo jogador
Sensores de Início/Fim	Sensores de Efeito Hall lineares	Detecção da posição do anel no início e fim do percurso
Ímã	Posicionado no anel condutor	Componente necessário para acionar os sensores de Efeito Hall
Atuadores de Movimento	Micro Servomotores (ex: SG90, modelo genérico)	Movimentação de seções do labirinto para variar a dificuldade
Feedback Visual (Status Jogo)	LED RGB	Indicação visual de eventos do jogo (perda de vida, nível concluído)
Feedback Visual (QTE Manopla)	LED RGB (integrado à manopla do anel)	Indicação visual da sequência de botões para QTEs
Botões de Interação (QTE)	Dois botões (integrados à manopla do anel)	Entrada do jogador para os Quick Time Events
Interface de Áudio	Alto-falante (conectado ao sistema de som do	Reprodução de feedback sonoro e instruções auditivas

	computador)	
Computador	PC padrão com capacidade de executar Python/Pygame	Execução da lógica principal do jogo e interface gráfica
Fonte de Alimentação 5V	Usada para alimentar os servos	-

Figura 1 - Diagrama do Hardware



Fonte: Autoria própria

6. HISTÓRICO DE ATUALIZAÇÕES (COMMITS)

Março de 2025 – Início do projeto: O repositório foi inicializado no final de março (commit “first commit” em 26/03/2025), contendo a estrutura básica do projeto e possivelmente uma primeira versão da documentação. Nos dias seguintes, foram feitos ajustes iniciais na documentação (“documentation changes” em 26 e 28 de março) e inclusão de imagens ilustrativas do documento . Nessa fase, a base do código Pygame começou a ser criada, provavelmente com uma janela simples e teste de leitura do Arduino.

Início de abril de 2025 – Protótipo da interface gráfica: No começo de abril, a equipe trabalhou em uma interface de usuário inicial. Isso é indicado pelo commit “Add first version of game interface” em 05/04/2025 , que adicionou a primeira versão das telas de jogo. Logo em seguida, houve refinamentos como “Fix button and text align” em 09/04/2025 – correção de alinhamento de botões e textos na UI após testes iniciais . Nessa época também foram integradas mudanças da branch de interface (commits merge em 09/04/2025), consolidando o código de múltiplas telas no ramo principal.

09 de abril de 2025 – Recursos de usabilidade: Ainda em 09/04, implementou-se a barra de progresso na tela de jogo e o botão de mute de áudio (commit “Add progress bar in game screen and mute button”) . Essa alteração trouxe a visualização do progresso do jogador no labirinto e a opção de desligar o som – provavelmente em resposta a testes iniciais com usuários ou requisitos de acessibilidade. Junto a isso, ajustes na documentação do projeto foram feitos para descrever a interface (“Documentation changes: interface” commit) .

10 de abril de 2025 – Transições e modularização: Neste ponto, foi adicionado um efeito de transição visual entre telas (commit “Add transition effects and integrate fade transitions in the main game loop” em 10/04) , tornando a troca de telas (ex: do jogo para tela de conclusão) mais suave. Além disso, ocorreu uma importante reestruturação do código: commit “Add all modular classes to improve maintainability and ease implementation of new features” . Essa mudança (commit em 09/04) introduziu classes modulares, reorganizando o código em módulos como AudioManager , SistemaConquistas, etc., para facilitar acréscimo de funcionalidades

sem quebrar o código existente. Esse refactoring preparou a base para as próximas funcionalidades complexas.

11 de abril de 2025 – Sistema de Conquistas: Foi nesta data implementado o sistema de conquistas completo, conforme o commit “Add achievements system with unlock notification and achievements screen” . Esse commit adicionou a lógica de verificação e armazenamento de conquistas, as imagens/icons correspondentes (mestre.png, persistente.png, etc.), a tela de conquistas e a notificação de desbloqueio exibida nas telas de conclusão de nível/jogo. Com essa adição, o jogo passou a recompensar marcos especiais do jogador e guardar essas conquistas em seu perfil.

15 de abril de 2025 – Áudio e narrativa: Em meados de abril, foram integradas melhorias multimídia e de conteúdo. O commit “feat: Implement audio manager for background music and sound effects” em 15/04 introduziu o gerenciador de áudio e integrou sons em diversos pontos da interface . Paralelamente, o commit “Add new assets, implement initial game screens with user interaction and initial dialogue with Teseu” (15/04) agregou novos recursos gráficos/sonoros e incluiu o diálogo inicial com o personagem Teseu na história do jogo, além de tornar as telas iniciais (menu, seleção) plenamente interativas . Nesse ponto, o jogo já contava com música de fundo, efeitos sonoros, narrativa e todas as telas principais funcionalmente ligadas.

16–17 de abril de 2025 – Conteúdo extra e seleção de usuário: Em 16/04, adicionaram-se fontes, imagens e sons adicionais e um diálogo pós-nível1 (commit “Add new fonts, images, and sounds; remove unused assets; implement dialogue after level 1”). Em 17/04, ocorreu o commit “feat: Add character screen and update main menu for character selection”, implementando a tela de seleção de usuário e adaptando o menu principal para suportar múltiplos perfis . Essa funcionalidade permite gerenciar progresso de diferentes jogadores e atendeu a um requisito importante para uso em contexto terapêutico multiusuário.

Essa melhoria atendeu aos feedbacks de teste sobre estímulos visuais.

23 de Maio de 2025 – Ajustes no filtro de cinza:. Em 23/05/2025, já na etapa final, um commit de refatoração foi feito (“refactor: Add gray scale filter functionality and centralize text rendering in drawing utilities”) , melhorando a

implementação do filtro de cinza e unificando funções de desenho de texto. Entretanto, a forma como foi implementada essa função, ocasiona em uma perda de frames por segundo quando ativada, deve-se portanto analisar uma possível refatoração

24 de Maio de 2025 - Adição de Gráficos estatísticos: Em 24/05/2025, foi adicionado utilitários gráficos para gráficos e animações (commit “Add graphical utilities for Greek-themed charts and animations”). Esta alteração introduziu classes base para funcionalidades de gráficos, gráficos de linha e barras com desenho animado e estilos visuais aprimorados, além de suporte para valores discretos no eixo Y, rotulagem de eixo aprimorada e linhas de grade para melhor legibilidade. As animações para desenho de gráficos foram incluídas para melhorar a experiência do usuário e para que esse consiga acompanhar seu progresso no jogo. Integrou o filtro de escala de cinza na renderização do jogo e das caixas de diálogo (commit “feat: Integrate gray scale filter in game and dialog rendering”).

28 de maio de 2025 - Adição de vozes dubladas: Em 28/05/2025 , foi adicionado áudios dublados por IA para guiar e motivar o usuários nos seguintes eventos: Colisão (quando restar duas vidas para o jogador), Colisão (quando restar uma vida para o jogador), perda de level e conclusão de level. Para cada tipo de evento foram adicionadas 10 variações de roteiros e áudios que são escolhidos de forma aleatória pela classe `AudioManager`. Esta adição foi pensada para aumentar a imersão do usuário ao ser guiado pelo próprio Teseu ao longo do labirinto. Observação: como todos os outros áudios do jogo, esse também pode ser desativado pelo usuário

29 de maio de 2025 - HUD de Vidas com Corações e Adição de backgrounds específicos por nível: Em 29/05/2025, foram integrados dois commits principais que aprimoram a experiência visual. O primeiro adiciona ao HUD corações em pixel-art animados para representar as vidas restantes, sendo cortado quando ocorre colisão. O segundo introduziu uma imagem de fundo única para cada fase do labirinto (do “Labirinto Inicial” ao “Confronto Final”). O `DialogManager` foi atualizado para carregar e exibir automaticamente o background correspondente sempre que um nível ou diálogo é iniciado, mantendo a coerência visual e reforçando a ambientação temática de cada câmara.

Junho de 2025 – Acessibilidade, QTE e Refinamento da Integração Hardware-Software: Foi implementada a tela de opções de acessibilidade completa, introduzindo funcionalidades cruciais como o filtro de escala de cinza, volumes de áudio independentes, ajuste de velocidade dos servos, modo prática e outras configurações para atender a diferentes perfis sensoriais; Ocorreram diversas iterações no firmware do Arduino para otimizar a comunicação e a lógica de hardware; A lógica de envio de padrões dos servos via JSON foi substituída por padrões armazenados diretamente na memória Flash do Arduino, resolvendo problemas de memória (`NoMemory`) e latência; A lógica dos servos foi refatorada para uma máquina de estados (`MOVING`/`HOLDING`), corrigindo um bug crítico que dessincronizava o movimento com o tempo de espera; O sistema de Quick Time Events (QTEs) foi implementado, sendo um modelo interativo passo a passo, onde o jogador reage a um estímulo de cada vez, e os servos são congelados durante o evento para focar a atenção do jogador.

7. TESTES REALIZADOS E VERIFICAÇÃO DE INTEGRAÇÃO

Uma vez implementados os componentes de hardware e software, foi conduzida uma bateria de testes para garantir que todas as partes do sistema funcionam de forma correta e integrada. Os testes focaram em diversas camadas do projeto, desde testes unitários de sensores e atuadores isoladamente até sessões completas de uso com o hardware e o software operando conjuntamente.

Teste dos sensores e atuadores (Hardware)

Inicialmente, cada sensor e atuador foi testado de forma independente para validar seu funcionamento elétrico:

- Sensores de efeito Hall
    - O anel (com ímã) foi aproximado dos sensores e a leitura bruta do ADC foi monitorada via Serial.
    - Verificou-se que a presença do ímã gerava variações consistentes na saída (por exemplo, de aproximadamente 512 para valores acima de

800 em um sensor, e abaixo de 200 no outro, dependendo da polaridade).
    - Ajustes de limiar foram feitos até obter detecção confiável do anel nas posições inicial e final, sem falsos positivos quando o anel passava em outros pontos do percurso.
- Circuito de detecção de colisão
    - O anel foi tocado em diferentes pontos do labirinto e foi testado se a colisão foi detectada.
    - Também se mediu o tempo de resposta: assim que o anel encostava, o Arduino enviava o código de colisão quase instantaneamente (ordem de milissegundos), o que é adequado para feedback imediato.
- Servomotores
    - Com o firmware Arduino em modo stand-alone, comandos de variação de ângulo foram enviados manualmente (via Serial Monitor ou código de teste) para observar a movimentação das partes móveis.
- LED RGB
    - O teste do LED RGB da mesa focou em validar o acionamento das cores primárias e secundárias através de saídas digitais. Foram testadas as combinações para gerar as cores secundárias necessárias para o feedback do jogo: amarelo (vermelho + verde), ciano (verde + azul) e branco (vermelho + verde + azul). Os testes verificaram que todas as cores programadas no firmware para os eventos de jogo — **amarelo** para colisão, **verde** para conclusão de nível e **vermelho** para falha — eram exibidas de forma clara e correta.

Após esses testes unitários, constatou-se que cada elemento de hardware operava conforme esperado individualmente.

Testes de integração hardware-software (sistema físico + jogo)

Em seguida, partiram-se para testes integrados com o Arduino conectado ao software Pygame, simulando partidas reais:

- Fluxo básico de jogo
    - Realizou-se uma simulação completa: iniciar o jogo, escolher usuário, selecionar porta COM e começar o nível 1.
    - Movimentou-se o anel deliberadamente do início ao fim sem tocar nas bordas.
    - Observou-se no monitor Serial e na interface se o sensor de início foi detectado; o cronômetro da HUD começou a contar a partir do momento de partida.
    - Ao chegar ao final sem colisões, confirmou-se que o sensor final foi capturado e a tela de conclusão de nível apareceu exibindo o tempo decorrido.
    - Esse teste evidenciou que comunicação e sincronização entre Arduino e software estavam corretas em cenário ideal (sem erros).
- Detecção de colisões e vidas
    - Repetiu-se o teste introduzindo colisões intencionais. No nível 1, o anel foi encostado 1, 2 e 3 vezes em pontos diferentes.
    - A HUD diminuiu as vidas uma a uma e cada colisão gerou imediatamente:
        - Piscar do LED em amarelo.
        - Incremento do contador de colisões na tela (ou outra indicação visual, dependendo da implementação).
    - Verificou-se que não houve decremento duplicado: se o anel permanecesse encostando continuamente (dentro do tempo de debounce), o sistema contava apenas uma colisão até que ele se afastasse e encostasse de novo (lógica de debounce no firmware).
    - Quando as colisões atingiram 3 (quantidade de vidas configurada), a tela de perda de vidas foi acionada automaticamente, informando o fracasso no nível.
    - Testaram-se os botões dessa tela: “Reiniciar Nível” retornou ao jogo do mesmo nível com vidas resetadas; “Voltar ao Menu” interrompeu o jogo e voltou ao menu principal sem travamentos.
- Tempo de resposta e sincronização

    - Durante os testes integrados, mediu-se subjetivamente o tempo entre um evento físico e a resposta na tela.
    - Não houve atrasos perceptíveis que pudessem confundir o jogador.
- Movimentos do labirinto
    - Os testadores avaliaram os movimentos dos servos durante o jogo.
    - Observou-se que, no nível 2 em diante, os servos começaram a se mexer conforme programado, aumentando a dificuldade.
    - Verificou-se que isso obrigava o jogador a pausar ou ajustar o movimento do anel, confirmando o aumento de desafio.
    - Importante: testou-se se esses movimentos não criavam situações impossíveis; se um servo movesse uma parte do labirinto para posição inevitavelmente colidida, seria injusto. Ajustes finos foram feitos na amplitude e na velocidade para garantir que, mesmo com movimento, o jogador tivesse chance de reagir.
    - Esses testes garantiram que a dinâmica de dificuldade estava adequada e funcionava sem falhas; comandos de servo do PC/Arduino aconteciam nos momentos corretos.
- Comunicação Serial robusta
    - Para testar robustez, o jogo foi executado por períodos prolongados (15–20 minutos), observando-se se ocorria perda de sincronismo ou travamento. Nenhum teste resultou em congelamento do jogo – a taxa de transmissão de dados mostrou-se administrável, pois eventos de colisão e sensores são espalhados.
    - Também foram realizados testes de desconexão: se o Arduino fosse desconectado no meio do jogo, o software detectava a ausência e pausava ou indicava erro graciosamente. Em protótipo, desconectar a porta travou a leitura até reconexão, mas em uso normal isso não ocorre; recomenda-se evitar desconexão abrupta durante o uso.

Testes de funcionalidades de software (Interface e Lógica)

Paralelamente aos testes físicos, cada funcionalidade de software foi

verificada:

- Sistema de Perfis
    - Criaram-se múltiplos usuários e jogou-se níveis alternadamente com cada perfil.
    - Confirmou-se que progresso, conquistas e melhores tempos permanecem independentes; ao voltar para um usuário anterior, seus dados estavam intactos.
    - Testou-se exclusão de usuário e esta removeu corretamente o perfil selecionado.
- Sistema de Conquistas
    - Conquistas específicas foram testadas ao longo de várias partidas
    - Testou-se que conquistas já obtidas não aparecem duplicadas mesmo se os critérios forem cumpridos de novo.
- Interface Gráfica e Navegação
    - Navegação entre telas: menu, desempenho, conquistas, jogo, volta ao menu, etc., foi testada; cada transição carregou dados corretos (por exemplo, desempenho atualizava informações do último nível).
    - Botões em estados hover: ao passar o cursor, mudavam de cor ou aspecto; ao clicar, som de clique tocava e ação correspondente executava. Não houve botões inativos nem áreas de clique incorretas; problemas de alinhamento relatados em testes preliminares já tinham sido corrigidos.
    - Layout em diferentes resoluções: graças ao utilitário de redimensionamento, em resoluções maiores os elementos mantiveram proporções sem desalinhamento.
    - Desempenho: mesmo com muitos elementos (texto e imagens de conquistas), o Pygame manteve taxa de frames próxima de 30–60 FPS, sem slowdown perceptível. Entretanto, ao ativar o modo de escala de cinza o desempenho do jogo foi consideravelmente prejudicado, caindo para taxa de frames próxima de 15-30 FPS, em versões futuras pode ser considerado uma refatoração do sistema de

conversão de cores dos elementos de tela.

Verificações de integração final

Foi realizado um teste completo “do início ao fim”: perfil novo iniciou no nível 1 e jogou sucessivamente até o último nível, desbloqueando o final do jogo.

- Não houve vazamento de memória; o jogo ficou estável durante todo o percurso.
- Todos os resultados (melhores tempos, conquistas 100 %) foram salvos e estavam disponíveis para consulta.
- Logs de comunicação Serial mostraram troca correta de comandos e eventos, sem mensagens inesperadas.

Futuros testes de usabilidade e experiência do Usuário

Com as funcionalidades técnicas validadas, no futuro deve-se realizar simulações de uso contínuo do ponto de vista de usuários finais:

- Simulação com criança
- Avaliação de acessibilidade
- Motivação por conquistas

Resultados

O sistema atingiu grau sólido de confiabilidade e usabilidade. Problemas como alinhamento de botões, incoerência na lógica de cor cinza ou calibração de sensores foram corrigidos em commits específicos, garantindo integração fluida de hardware, firmware e software. Todos os subsistemas operam em harmonia.

CONCLUSÃO

O presente trabalho teve por objetivo conceber, implementar e validar o jogo sério *A Fúria do Minotauro* como ferramenta de apoio ao desenvolvimento da coordenação motora fina, especialmente voltado a pessoas com Transtorno do Espectro Autista (TEA). Partindo-se da revisão bibliográfica sobre intervenções lúdicas assistidas por tecnologia, definiu-se uma proposta que alia desafios motores graduais, narrativa mitológica motivadora e recursos de acessibilidade auditiva e visual.

No âmbito da engenharia de software, foi alcançada uma arquitetura modular em Python/Pygame que separa com nitidez as camadas de interface gráfica, lógica de jogo e comunicação com o hardware, garantindo fácil manutenção e extensibilidade. Do ponto de vista de hardware, o protótipo integrou sensores de efeito Hall, circuito de detecção de contato, micro-servomotores, LED RGB e buzzer, todos coordenados por firmware no Arduino UNO. A interação em tempo real entre esses componentes e o software demonstrou latência imperceptível para o usuário, assegurando feedback imediato — requisito essencial para intervenções terapêuticas eficazes.

Os testes unitários confirmaram o funcionamento elétrico confiável de cada sensor e atuador; os testes de integração validaram a troca de dados contínua entre firmware e aplicação; e as sessões completas de uso revelaram robustez, ausência de travamentos e aderência da mecânica de jogo às metas terapêuticas. Além disso, o sistema de conquistas e a representação estatística do desempenho mostraram-se fatores motivacionais adicionais, estimulando a prática repetida e consciente das atividades motoras.

Como limitações, destaca-se a queda de desempenho observada ao ativar o filtro de escala de cinza em computadores de menor capacidade gráfica e a necessidade de calibração específica caso se adotem modelos de sensores ou servos diferentes dos testados. Recomenda-se, como trabalhos futuros, (i) otimizar o algoritmo de conversão para tons de cinza, (ii) implementar adaptação automática

de dificuldade com base nos dados de desempenho históricos, (iii) expandir o conjunto de níveis e padrões de movimento, (iv) disponibilizar um modo on-line para acompanhamento remoto por terapeutas e (v) investigar a aplicação da plataforma em outras populações que também se beneficiem do treino de coordenação fina.

Em síntese, *A Fúria do Minotauro* atingiu plenamente os objetivos propostos ao comprovar viabilidade técnica, compatibilidade entre hardware e software, usabilidade satisfatória e potencial terapêutico. O projeto contribui tanto para a área de jogos sérios quanto para práticas de reabilitação motora, oferecendo uma solução acessível, motivadora e alinhada às necessidades sensoriais de usuários com TEA. Espera-se que, com os aperfeiçoamentos, o sistema possa ser adotado em ambientes clínicos e educacionais, ampliando o impacto social desta pesquisa.

REFERÊNCIAS

AUTIBOTS: Jogo Digital Educativo para Desenvolvimento Cognitivo e Motor de Crianças com Transtorno do Espectro Autista. SAMPAIO, L. P.; PEREIRA, C. P., 2023. Disponível em: https://journals-sol.sbc.org.br/index.php/rbie/article/view/3300. Acesso em: 18 mar. 2025.

Gesture-based Video Games to Support Fine-Motor Coordination Skills of Children with Autism. RUIZ-RODRIGUEZ, Aurora; CARO, Karina; MARTÍNE, Ana, 2019. Disponível em: https://dl.acm.org/doi/10.1145/3311927.3325310. Acesso em: 18 mar. 2025.

A CASE STUDY OF GESTURE-BASED GAMES in Enhancing the Fine Motor Skills of Children with Autism Spectrum Disorders. CAI, Su; ZHU, Gaoxia; WU, Ying-Tien; LIU, Enrui; HU, Xiaoyi., 2018. Disponível em: https://www.researchgate.net/publication/323214177_A_case_study_of_gesture-based_games_in_enhancing_the_fine_motor_skills_and_recognition_of_children_with_autism. Acesso em: 18 mar. 2025.

Documentação interativa: https://minotauro-doc.vilquer.dev. Acesso em: 24 mai. 2025

Projeto no GitHub: https://github.com/caiovilquer/A-furia-do-minotauro. Acesso em: 24 mai. 2025

Projeto no GitLab-USP: https://gitlab.uspdigital.usp.br/caio.vilquer/A-furia-do-minotauro. Acesso em: 28 mai. 2025