Universidade Federal de São Carlos - UFSCar
Departamento de Ciência da Computação - DC
Programa de Pós-Graduação em Ciência da Computação - PPGCC

Ricardo Henrique da Silva Assis

Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM

São Carlos - SP
2025

RICARDO HENRIQUE DA SILVA ASSIS

Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação.

Área de Concentração: Sistemas de Automação e Robótica - SAR

Orientador: Rafael Vidal Aroca

São Carlos - SP

2025

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Relatório de Defesa de Dissertação
Candidato: Ricardo Henrique da Silva Assis

Aos 30/06/2025, às 08:00, realizou-se na Universidade Federal de São Carlos, nas formas e termos do Regimento Interno do Programa de Pós-Graduação em Ciência da Computação, a defesa de dissertação de mestrado sob o título: Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM, apresentada pelo candidato Ricardo Henrique da Silva Assis. Ao final dos trabalhos, a banca examinadora reuniu-se em sessão reservada para o julgamento, tendo os membros chegado ao seguinte resultado:

Participantes da Banca	**Função**	**Instituição**	**Resultado**	**Resultado Final**
Prof. Dr. Rafael Vidal Aroca	Presidente	UFSCar	Aprovado	Aprovado
Prof. Dr. Roberto Santos Inoue	Titular	UFSCar	Aprovado	
Prof. Dr. Daniel Varela Magalhães	Titular	USP	Aprovado	

Parecer da Comissão Julgadora*:

A comissão examinadora considerou que o trabalho satisfaz as exigências do programa de mestrado.
Na apresentação o candidato mostrou bom conhecimento do assunto, tendo respondido com segurança as questões formuladas pela banca.
A comissão solicita, no entanto, que o candidato faça pequenos ajustes no texto.

Encerrada a sessão reservada, o presidente informou ao público presente o resultado. Nada mais havendo a tratar, a sessão foi encerrada e, para constar, eu, Ivan R Silva, representante do Programa de Pós-Graduação em Ciência da Computação, lavrei o presente relatório, assinado por mim e pelos membros da banca examinadora.

Prof. Dr. Rafael Vidal Aroca

Representante do PPG: Ivan R Silva

Prof. Dr. Roberto Santos Inoue

Prof. Dr. Daniel Varela Magalhães

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Rafael Vidal Aroca, Roberto Santos Inoue, Daniel Varela Magalhães e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

gov.br Documento assinado digitalmente
RAFAEL VIDAL AROCA
Data: 30/06/2025 09:44:06-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Rafael Vidal Aroca

(X) Não houve alteração no título ( ) Houve alteração no título. O novo título passa a ser:

Observações:
a) Se o candidato for reprovado por algum dos membros, o preenchimento do parecer é obrigatório.
b) Para gozar dos direitos do título de Mestre ou Doutor em Ciência da Computação, o candidato ainda precisa ter sua dissertação ou tese homologada pelo Conselho de Pós-Graduação da UFSCar.

Ricardo Henrique da Silva Assis
Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM/ Ricardo Henrique da Silva Assis. – São Carlos - SP, 2025- 137 p. : il.(alguma color.); 30 cm.
Prof. Dr.Rafael Vidal Aroca
Dissertação de Mestrado –
Universidade Federal de São Carlos - UFSCar
Departamento de Ciência da Computação - DC
Programa de Pós-Graduação em Ciência da Computação - PPGCC, 2025.
Palavras chave principais: 1. BIPES. 2. Blocos. 3. Jogos. 4. STEAM.
CDU 02:141:005.7

Ricardo Henrique da Silva Assis

Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação.

Área de Concentração: Sistemas de Automação e Robótica - SAR

Orientador: Rafael Vidal Aroca

Folha de aprovação. São Carlos - SP, 2025:

Prof. Dr. Rafael Vidal Aroca
Presidente da Banca - Membro Interno

Prof. Dr. Roberto S. Inoue
Membro Título Interno

Prof. Dr. Daniel Varela Magalhães
Membro Título Externo

São Carlos - SP
2025

RESUMO

SILVA, A. R. **Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM**. 2025. Dissertação (Mestrado Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Paulo, 2025

O presente trabalho aborda a necessidade premente de atrair talentos para áreas de STEM (Ciência, Tecnologia, Engenharia e Matemática), reconhecendo a importância desses campos para o avanço tecnológico e a inovação. Uma ferramenta que pode desempenhar um papel significativo nesse processo é o BIPES, uma plataforma que utiliza a programação por blocos para simplificar o desenvolvimento de projetos em sistemas embarcados. Reconhecendo o potencial atrativo dos jogos para estudantes do ensino fundamental e médio, o trabalho propõe uma abordagem inovadora que utiliza o BIPES para auxiliar os alunos na construção de protótipos de jogos físicos, integrando componentes como teclas, potenciômetros e displays. Dessa forma, o presente trabalho estruturou e implementou blocos lúdicos, trechos de códigos, bem como roteiros e guias de construção que instruem alunos interessados a construir jogos físicos através da programação em blocos e construção de um protótipo com componentes eletrônicos facilmente acessíveis no mercado. Foi possível construir (e testar) roteiros da construção do dispositivo (jogo) físico e sua programação para implementar um jogo do tipo PONG e um jogo do tipo Space Invaders. Este resultado, que é ligado a temática de jogos, é motivador para estudantes de vários níveis e busca contribuir para fomentar a formação de novos talentos e o avanço contínuo nas áreas de STEM.

Palavras-chave: BIPES, MicroPython, STEM, Jogos, Construcionismo, Educação.

ABSTRACT

SILVA, A. R. **Uma proposta de uso de programação em blocos para desenvolvimento de jogos físicos e incentivo a área STEM**. 2025. Dissertação (Mestrado Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Paulo, 2025

The present work addresses the urgent need to attract talent to STEM fields (Science, Technology, Engineering, and Mathematics), recognizing the importance of these areas for technological advancement and innovation. One tool that can play a significant role in this process is BIPES—a platform that uses block-based programming to simplify the development of embedded systems projects. Acknowledging the strong appeal of games for elementary and high school students, this study proposes an innovative approach that leverages BIPES to assist students in building physical game prototypes. These projects integrate components such as buttons, potentiometers, and displays. To this end, the project developed and implemented playful programming blocks, code snippets, and instructional guides that support students in constructing physical games through block-based programming and the assembly of prototypes using low-cost, easily accessible electronic components. It was possible to build (and test) step-by-step guides for assembling physical game devices and programming them to implement a version of the classic PONG game and a Space Invaders-style game. This outcome—centered around the theme of gaming—proves to be highly motivating for students at various levels and aims to foster the development of new talent while supporting continued progress in STEM education.

Keywords: BIPES, MicroPython, STEM, Games, Constructionism, Education.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 . INTRODUÇÃO

Atualmente, observa-se uma queda significativa no interesse dos jovens por cursos pertencentes às áreas STEM (*Science, Technology, Engineering and Math*). Esse declínio despertou a atenção de pesquisadores e educadores, que buscam compreender suas causas e encontrar soluções para reverter essa tendência preocupante. Inúmeros estudos têm sido conduzidos para investigar as possíveis razões por trás do crescente desinteresse das novas gerações por esses campos fundamentais para o avanço da sociedade (MATA; TOBON, 2018; WESTER et al., 2021).

Entre os diversos fatores que contribuem para esse fenômeno, destacam-se a falta de representatividade na área para certos subgrupos, como as mulheres, e os desafios socioeconômicos enfrentados por muitos jovens (STAUS et al., 2020; WANG et al., 2023). Além disso, a falta de recursos educacionais adequados, a ausência de conscientização sobre as oportunidades de carreira em STEM e as diferenças culturais também têm impacto significativo, gerando uma percepção negativa sobre esses campos (MATA; TOBON, 2018).

Uma das estratégias que vem se destacando para reverter esse quadro de desinteresse é a gamificação do processo de aprendizagem. A utilização de jogos como recurso pedagógico tem demonstrado grande potencial para tornar o aprendizado mais atrativo, principalmente entre jovens que já estão familiarizados com o universo dos games. Ao aliar conteúdos das áreas de STEM com a construção de jogos físicos e digitais, é possível proporcionar um ambiente educacional mais interativo, criativo e centrado no aluno. Essa abordagem favorece não apenas a motivação e o engajamento, mas também o desenvolvimento de competências importantes como pensamento lógico, resolução de problemas e trabalho em equipe. Nesse sentido, ferramentas educacionais acessíveis e adaptadas ao nível dos estudantes, como o BIPES, tornam-se essenciais para viabilizar esse tipo de proposta didática. Estudos como o de (XU; JIN, 2021). demonstram que workshops de desenvolvimento de jogos conduzidos por mentores têm impacto positivo na curiosidade e no interesse dos alunos em cursos introdutórios de programação, reforçando o potencial dessa abordagem.

Nesse contexto, a ideia do construcionismo proposto por Seymour Papert em seu livro *"Mindstorms: Children, Computers, and Powerful Ideas"*, oferece uma abordagem valiosa para enfrentar o desafio do desinteresse em STEM (PAPERT, 1980). Papert propõe uma pedagogia construtivista que enfatiza o aprendizado ativo e prático, no qual os alunos se envolvem em projetos significativos que têm relevância para suas vidas. Ele argumenta que os estudantes aprendem de maneira mais eficaz quando são agentes ativos na construção do próprio conhecimento, em vez de meros receptores passivos de informações.

A integração das ideias de Papert com a plataforma BIPES *(Block Based Integrated Platform for Embedded Systems)* representa uma oportunidade promissora para abordar

Figura 1 – Código para piscar o LED do microcontrolador (AROCA et al., 2021)

o desinteresse em STEM de forma mais eficaz. Como pode ser visto na figura 1, BIPES oferece uma interface de programação baseada em blocos que simplifica o desenvolvimento de projetos de sistemas embarcados, tornando a programação mais acessível e envolvente para os alunos, mesmo aqueles sem experiência prévia em programação (JUNIOR et al., 2020). A convergência dessas ideias cria um ambiente educacional estimulante e dinâmico, no qual os alunos podem explorar, experimentar e aplicar conceitos STEM em projetos práticos e significativos (PAPERT, 1980). Esta abordagem integrada não apenas combate o desinteresse em STEM, mas também promove uma compreensão mais profunda e duradoura dos conceitos fundamentais nessas áreas.

Considerando a crescente preocupação com o declínio no interesse dos jovens por cursos nas áreas STEM, assim como a convergência das ideias construtivistas de Seymour Papert com a inovação tecnológica representada pela plataforma BIPES, há um interesse crescente em explorar o potencial dessas abordagens para reverter essa tendência preocupante. Com a rápida evolução das tecnologias educacionais e a disponibilidade de plataformas como o BIPES, Scratch, App Inventor, micro::bit, dentre outros, há um renovado interesse em avaliar o impacto dessas ferramentas na motivação e no engajamento dos alunos em disciplinas STEM.

Diante do contexto apresentado, este estudo propõe investigar o uso da plataforma BIPES para promover o engajamento dos alunos nessas áreas. Por meio de revisão bibliográfica e desenvolvimento de novos blocos de programação, busca-se melhorar a adequação da plataforma aos objetivos educacionais de STEM. O objetivo é fornecer uma experiência prática e interativa no desenvolvimento de jogos, contribuindo para o aumento do interesse nos cursos STEM.

Desta forma, este trabalho propõe uma análise sobre os fatores que contribuem para o declínio do interesse dos jovens por cursos nas áreas STEM, utilizando as ideias e

referências de Papert (1980) como base teórica. Será conduzida uma revisão bibliográfica abrangente para investigar as possíveis causas desse fenômeno preocupante, considerando aspectos como falta de representatividade, desafios socioeconômicos e falta de recursos educacionais adequados. Além disso, serão exploradas as soluções propostas por Papert para promover um aprendizado mais ativo e prático, enfatizando a importância da integração entre teoria e prática no ensino de STEM.

Nesse contexto, será considerado o potencial do BIPES como uma ferramenta para atrair e engajar os alunos, especialmente através da criação de jogos em dispositivos físicos. No mais, estudos destacam a influência positiva dos jogos no interesse e na curiosidade dos alunos em disciplinas STEM (ARZTMANN et al., 2022; XU; JIN, 2021). Integrando o BIPES com a criação de jogos, os alunos terão a oportunidade de desenvolver não apenas habilidades técnicas em programação e sistemas embarcados, mas também habilidades criativas e de resolução de problemas.

Além disso, ao integrar o pensamento computacional na educação matemática, é possível notar uma melhora significativa nas habilidades matemáticas e na resolução de problemas do mundo real (BARCELOS; SILVEIRA, 2012; KONG; ABELSON, 2019). Ao criar seus próprios jogos usando o BIPES, os alunos serão incentivados a aplicar o pensamento computacional para projetar e implementar soluções criativas, promovendo assim uma compreensão mais profunda e significativa dos conceitos STEM.

Dessa forma, a integração do BIPES com a criação de jogos não só pode atrair os alunos para os cursos STEM, mas também pode ajudá-los a desenvolver habilidades essenciais para o sucesso no século XXI, incluindo pensamento crítico, criatividade e colaboração.

A Figura 2 mostra um jogo físico / dispositivo (jogo PONG) construído por um estudante de ensino médio seguindo o material produzido neste trabalho

1.1 Objetivo Geral

Desenvolver melhorias, blocos, implementações e produzir material didático que permite que estudantes de ensino fundamental II e médio criem jogos e outros dispositivos interativos utilizando a ferramenta BIPES, fazendo com que eles desenvolvam um pensamento computacional que pode ser útil em várias áreas

1.2 Objetivos Específicos

1. Identificar possíveis melhorias no BIPES por meio da criação de novos blocos de programação;

Figura 2 – Jogo Pong implementado no display OLED.

2. Definir os materiais necessários para garantir o acesso dos alunos, considerando disponibilidade e baixo custo;

3. Identificar as bibliotecas necessárias para a utilização dos novos blocos de programação;

4. Desenvolver blocos de programação na plataforma BIPES para permitir o uso de sensores ultrassom e possibilitar alterações nos displays OLED;

5. Apresentar a plataforma BIPES e suas diversas aplicações, incluindo a criação de jogos, para alunos do ensino médio e fundamental;

6. Promover e atrair alunos para os cursos STEM por meio do uso da plataforma BIPES e de suas funcionalidades.

7. Desenvolver material didático (roteiro de prática) e avaliar sua facilidade de uso

2 . REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta duas seções complementares. Na primeira parte, realiza-se uma revisão bibliográfica das tecnologias empregadas no desenvolvimento do projeto. Na segunda parte, conduz-se uma pesquisa sobre trabalhos correlatos por meio de uma revisão sistemática da literatura.

2.1 Métodos de pesquisa da literatura

A revisão bibliográfica realizada nesta pesquisa teve como objetivo identificar e analisar trabalhos relacionados a três principais eixos: (i) ambientes de programação visual para sistemas embarcados; (ii) metodologias ativas de aprendizagem, com foco em jogos e computação física; e (iii) democratização do acesso à educação tecnológica, especialmente no ensino de STEM.

Para garantir a abrangência e a relevância das fontes, a busca bibliográfica adotou três estratégias principais: **busca sistemática**, **indicações do orientador** e **snowballing**.

A **busca sistemática** foi conduzida em bancos de dados acadêmicos como IEEE Xplore, ACM Digital Library, Scopus, Google Scholar e Portal de Periódicos da CAPES utilizando combinações de descritores como *"programming by blocks"*, *"physical computing"*, *"STEM education"*, *"educational games"* e *"embedded systems"*. Esta estratégia permitiu identificar referências fundamentais, como os trabalhos de Resnick et al. (2009), que discutem o papel dos ambientes de programação visual na educação, e de Arztmann et al. (2022), que analisam a utilização de jogos no ensino de STEM.

As **indicações do orientador** também desempenharam um papel importante na seleção de referências relevantes, orientando a leitura de autores clássicos e fundamentais para a compreensão do campo, como Papert (1980), com sua proposta do construcionismo, e também a literatura do proprio BIPES

Por fim, aplicou-se a técnica de **snowballing**, na qual, a partir das leituras iniciais selecionadas por busca sistemática e recomendações do orientador, foram exploradas referências citadas nos trabalhos e também buscas relacionadas a conceitos emergentes identificados durante a revisão. Esse procedimento permitiu ampliar a análise, levando à identificação de estudos complementares, como os de Wester et al. (2021), que analisam motivações e desafios para o ingresso em carreiras de STEM, e de Staus et al. (2020), que investigam barreiras ao engajamento estudantil nessas áreas.

Ainda que tais textos não tenham sido identificados diretamente através das referências uns dos outros, a técnica de snowballing foi útil ao permitir expandir a busca a partir de *keywords* e temáticas comuns que surgiram na leitura crítica do material

previamente selecionado.

Além de artigos acadêmicos, foram consultadas documentações técnicas e tutoriais relacionados à plataforma BIPES (BIPES PROJECT, 2025) e ao ambiente Blockly (GOOGLE BLOCKLY, 2025), essenciais para orientar a criação e integração dos novos blocos de programação.

Essa combinação de estratégias garantiu uma revisão bibliográfica sólida e abrangente, que fundamenta as escolhas metodológicas e técnicas realizadas no desenvolvimento desta pesquisa.

2.2 Fundamentação Teórica

Nesta seção serão abordadas as principais tecnologias utilizadas para o desenvolvimento do presente trabalho.

2.2.1 Educação STEM

A Educação STEM desempenha um papel crucial no desenvolvimento de habilidades interdisciplinares essenciais para o sucesso em diversas áreas profissionais e acadêmicas (MATA; TOBON, 2018). Ao integrar ciência, tecnologia, engenharia e matemática, os alunos são expostos a uma ampla gama de conceitos e práticas que estimulam o pensamento crítico, a resolução de problemas e a criatividade (KONG; ABELSON, 2019).

Por exemplo, as atividades práticas de laboratório em ciências permitem aos alunos explorar o método científico, coletar e analisar dados, e tirar conclusões baseadas em evidências (MATA; TOBON, 2018). Essas habilidades são transferíveis para campos como pesquisa, medicina e engenharia, onde a capacidade de analisar informações e tomar decisões informadas é fundamental (KONG; ABELSON, 2019).

Da mesma forma, o uso de tecnologia e programação em projetos de engenharia e matemática promove o desenvolvimento de habilidades de resolução de problemas algorítmicos e lógicos (MATA; TOBON, 2018). Os alunos aprendem a decompor problemas complexos em etapas menores, identificar padrões e desenvolver algoritmos eficientes para encontrar soluções (KONG; ABELSON, 2019). Essas habilidades são altamente valorizadas em campos como desenvolvimento de software, análise de dados e tomada de decisões estratégicas (MATA; TOBON, 2018).

Além disso, a colaboração em equipe é uma parte integrante da Educação STEM, onde os alunos frequentemente trabalham em projetos de grupo para resolver desafios complexos (MATA; TOBON, 2018). Ao colaborar com colegas de diferentes formações e perspectivas, os alunos aprendem a comunicar efetivamente suas ideias, negociar soluções e valorizar a diversidade de pensamento (KONG; ABELSON, 2019). Essas habilidades são cruciais em ambientes de trabalho colaborativos, onde a capacidade de trabalhar em

equipe e resolver conflitos construtivamente é essencial para o sucesso (MATA; TOBON, 2018).

2.2.2 Pensamento Computacional

O Pensamento Computacional é uma habilidade cognitiva fundamental que envolve a formulação e resolução de problemas de uma maneira que aproveita os conceitos e processos essenciais à computação. A integração do Pensamento Computacional na educação matemática destaca sua importância em promover uma compreensão mais profunda dos princípios matemáticos e sua aplicação no mundo real (BARCELOS; SILVEIRA, 2012). Atividades práticas e desafios baseados em problemas demonstraram aprimorar as habilidades de raciocínio lógico e algorítmico dos alunos, preparando-os para enfrentar uma variedade de situações reais (KONG; ABELSON, 2019).

2.2.3 Sistemas Embarcados

Os Sistemas Embarcados têm uma ampla aplicação em nossa sociedade, sendo encontrados em dispositivos médicos vitais, sistemas de controle de tráfego e dispositivos IoT (Internet das Coisas). Esses sistemas são projetados para funções específicas em dispositivos eletrônicos e desempenham um papel essencial em nossa vida cotidiana. Compreender os princípios de design e implementação de sistemas embarcados é fundamental para adquirir habilidades práticas e conhecimentos valorizados no mercado de trabalho.

Uma visão abrangente dos princípios e práticas do design de sistemas embarcados aborda desde os fundamentos da eletrônica até estratégias avançadas de depuração e teste (MARWEDEL, 2021), oferecendo uma base sólida para exploração. Ao estudar exemplos de sistemas embarcados do mundo real, os alunos podem compreender como esses conceitos são aplicados em contextos práticos, desde dispositivos médicos implantáveis até sistemas de controle industrial automatizados.

Além disso, uma revisão sistemática sobre a seleção de ferramentas em engenharia de sistemas baseada em modelos para sistemas embarcados destaca a importância de escolher as ferramentas certas para o desenvolvimento eficiente e confiável de sistemas embarcados (RASHID; ANWAR; KHAN, 2015). Essa abordagem ajuda os alunos a entender como as melhores práticas de engenharia de software se aplicam ao desenvolvimento de sistemas embarcados e como essas habilidades são cruciais para enfrentar os desafios complexos encontrados nesse campo em constante evolução.

2.2.4 Desenvolvimento de Jogos

O Desenvolvimento de Jogos é uma área que combina elementos de programação, design de jogos e criatividade para criar experiências interativas e envolventes. Pesquisas

como as conduzidas por Xu (2021) destacam a importância dos workshops de desenvolvimento de jogos, oferecendo aos alunos a oportunidade de aplicar conceitos de programação em um contexto prático e significativo. Tais workshops não apenas aumentam o interesse dos alunos em disciplinas de programação, mas também são associados ao desenvolvimento de habilidades de pensamento crítico, resolução de problemas e colaboração em equipe (DUCH; JAWORSKI, 2018; HAVA; GüYER; CAKIR, 2020). Além disso, a criação de jogos pode servir como uma forma de expressão criativa, permitindo que os alunos desenvolvam e compartilhem suas próprias narrativas e experiências (CHURSIN; SEMENOV, 2020; ZHU; WANG, 2019).

2.2.5 BIPES

A plataforma BIPES representa uma ferramenta inovadora no campo da educação STEM, oferecendo uma interface de programação baseada em blocos que simplifica o desenvolvimento de projetos de sistemas embarcados. Junior (JUNIOR et al., 2020) descreve a arquitetura e funcionalidades da plataforma, destacando sua capacidade de tornar a programação mais acessível e envolvente para os alunos, mesmo aqueles sem experiência prévia em programação. A integração das ideias de Seymour Papert com a tecnologia do BIPES cria um ambiente educacional estimulante, no qual os alunos podem explorar, experimentar e aplicar conceitos STEM em projetos práticos e significativos. Ao fornecer uma plataforma intuitiva e acessível para o desenvolvimento de projetos de sistemas embarcados, o BIPES desempenha um papel fundamental em promover o engajamento dos alunos em disciplinas STEM e prepará-los para os desafios do século XXI.

2.3 Trabalhos Relacionados

Este tópico oferece uma descrição dos temas abordados em 26 artigos identificados por meio dos metodos de busca citados anteriormente, ressaltando suas relações com os objetivos desta pesquisa.

2.3.1 Resultados da revisão da literatura

A revisão literatura foi conduzida com base em questões de pesquisa, as quais podem ser encontradas na Tabela 1.

Tabela 1 – Revisão da Literatura

Título	**Relevância**	**Motivo da Relevância**
Effects of games in STEM education: a meta-analysis on the moderating role of student background characteristics	ALTA	O estudo fornece uma análise abrangente sobre o uso de jogos na educação STEM, destacando a importância do contexto do aluno.
Game development workshops designed and delivered by peer mentors to increase student curiosity and interest in an introductory programming course	ALTA	Enfoque em workshops de desenvolvimento de jogos como estratégia para promover interesse em programação, relevante para o contexto educacional.
Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things	ALTA	Oferece uma visão abrangente dos princípios e práticas do design de sistemas embarcados, relevante para a compreensão de conceitos essenciais.
Computational Thinking Education	ALTA	Explora a teoria e a prática da educação em pensamento computacional, oferecendo insights valiosos para educadores e pesquisadores interessados em promover o pensamento computacional entre os alunos.
Enriching Computer Science Programming Classes with Arduino Game Development	ALTA	Destaca a integração de desenvolvimento de jogos Arduino nas aulas de programação, relevante para a educação em ciência da computação.
Model-driven Game Development: A Literature Review	ALTA	Oferece uma revisão abrangente do desenvolvimento de jogos orientado a modelos, relevante para a compreensão de abordagens alternativas.
Learning game development with Unity3D engine and Arduino microcontroller	MÉDIA	Explora a aprendizagem do desenvolvimento de jogos com Unity3D e Arduino, relevante para a integração de tecnologias.

Gifted students' learning experiences in systematic game development process in after-school activities	MÉDIA	Foca nas experiências de aprendizado de estudantes talentosos no processo de desenvolvimento de jogos, relevante para estratégias educacionais específicas.
Pensamento Computacional e Educação Matemática: Relações para o Ensino de Computação na Educação Básica	ALTA	Explora a relação entre pensamento computacional e educação matemática, relevante para abordagens interdisciplinares.
Pensamento computacional na formação de professores: experiências e desafios encontrados no ensino da computação em escolas públicas	MÉDIA	Aborda a formação de professores em pensamento computacional, relevante para estratégias de ensino.
Towards the Tools Selection in Model Based System Engineering for Embedded Systems - A Systematic Literature Review	ALTA	Oferece uma revisão sistemática sobre a seleção de ferramentas em engenharia de sistemas baseada em modelos, relevante para a escolha de ferramentas neste trabalho.
Mindstorms: Children, Computers, and Powerful Ideas	ALTA	Obra clássica sobre a utilização de computadores na educação de crianças, relevante para a compreensão de abordagens pedagógicas.
BIPES: Block Based Integrated Platform for Embedded Systems	ALTA	Apresenta uma plataforma integrada baseada em blocos para sistemas embarcados, relevante para a proposta deste trabalho.
Uma Introdução à Internet das Coisas e Sistemas Embarcados utilizando Programação por Blocos com BIPES e ESP8266 / ESP32	ALTA	Apresenta de forma acessível os fundamentos do uso da plataforma BIPES com ESP8266/ESP32, útil para ensino e experimentação prática.

Student Engagement Declines in STEM	MÉDIA	Aborda a queda no engajamento de estudantes de STEM durante o ensino remoto causado pela COVID-19, relevante para considerações sobre educação remota.
Analysis of Factors Associated to the Enrollment and Demand of Computing-Related Careers	ALTA	Investigação de fatores associados à demanda por carreiras relacionadas à computação, relevante para compreensão das tendências educacionais.
Interested, Disinterested, or Neutral: Exploring STEM Interest Profiles and Pathways in A Low-Income Urban Community	MÉDIA	Explora os perfis de interesse em STEM em uma comunidade urbana de baixa renda, relevante para questões de diversidade e inclusão.
Gender differences in high school students' interest in STEM careers: a multigroup comparison based on structural equation model	MÉDIA	Analisa as diferenças de gênero no interesse de estudantes do ensino médio em carreiras STEM, relevante para considerações sobre diversidade de gênero em STEM.
Reasserting US Leadership in Microelectronics: The Role of Universities	MÉDIA	Destaca a escassez de profissionais em microeletrônica nos EUA e propõe medidas para fomentar talentos em STEM.
Programming Embedded Systems: With C and MicroPython	ALTA	Apresenta práticas modernas de programação para sistemas embarcados, incluindo MicroPython, essencial para o domínio técnico da área.
Scratch: Programming for all	MÉDIA	Apresenta o Scratch como ferramenta de introdução à programação, relevante para o ensino inicial de lógica computacional.
Programming environments for novices	MÉDIA	Aborda ambientes de programação voltados a iniciantes, essencial para compreensão de ferramentas educacionais.

Blockly: A visual programming editor	MÉDIA	Explora o editor visual Blockly, base de diversas plataformas de programação por blocos, inclusive o BIPES.
Criando novos blocos no BIPES	MÉDIA	Explica como personalizar blocos no BIPES, relevante para expansão das funcionalidades da plataforma.
Blockly Developer Tools and Samples	MÉDIA	Fornece recursos e exemplos para desenvolvimento com Blockly, relevante para desenvolvedores da plataforma BIPES.
MicroPython: Python for microcontrollers	MÉDIA	Apresenta a linguagem MicroPython e sua aplicação em microcontroladores, essencial para aplicações em BIPES.
Raspberry Pi Pico Color OLED (SSD1331) display tutorial using CircuitPython	BAIXA	Tutorial específico para visualização gráfica com CircuitPython, útil como referência técnica.
Raspberry Pi Pico running Pong	BAIXA	Demonstra um exemplo prático de jogo em Pong com Raspberry Pi Pico, ilustrativo para aplicações lúdicas.
MicroPython for Kids	BAIXA	Apresenta tutoriais básicos para crianças em MicroPython, importante para introdução de jovens à programação.
Raspberry Pi Buzzer Delay	BAIXA	Exemplo técnico prático sobre uso de buzzer no Raspberry Pi, com aplicação limitada.
Building MicroPython for the Raspberry Pi PICO with Ethernet and WebREPL Support	MÉDIA	Tutorial técnico sobre construção do MicroPython customizado para o Raspberry Pi Pico, relevante para aplicações avançadas no BIPES.

Após revisão dos artigos classificados com alta relevância na tabela fornecida, destaca-se que foram atribuídos 13 textos a essa categoria. Abaixo estão os resumos individuais deles:

"Effects of games in STEM education: a meta-analysis on the moderating role of student background characteristics" de Arztmann et al., (2022): conduz uma meta-análise para investigar os efeitos dos jogos na educação STEM, com foco no papel das características de fundo dos alunos como moderadoras desses efeitos. A pesquisa examinou uma variedade de estudos sobre o assunto e descobriu que os jogos têm um impacto significativo na educação STEM. No entanto, os resultados mostraram que esse impacto

pode variar dependendo do contexto do aluno, como sua formação educacional e experiência prévia com jogos. Essa análise fornece visões valiosas para educadores e pesquisadores interessados no uso de jogos para promover a aprendizagem em STEM.

"Game development workshops designed and delivered by peer mentors to increase student curiosity and interest in an introductory programming course" de Xu et al., (2021): investiga o impacto de workshops de desenvolvimento de jogos conduzidos por mentores de colegas na promoção da curiosidade e interesse dos alunos em um curso introdutório de programação. A pesquisa demonstra que esses workshops são eficazes para engajar os alunos e despertar seu interesse pela programação, oferecendo uma experiência prática e colaborativa na criação de jogos. Essa abordagem inovadora de aprendizado pode ser uma maneira eficaz de tornar a programação mais acessível e atrativa para os alunos, especialmente para aqueles que estão começando seus estudos nessa área.

"Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things": escrito por Marwedel (MARWEDEL, 2021) fornece uma visão abrangente dos fundamentos do design de sistemas embarcados, cobrindo aspectos desde a base dos sistemas até aplicações em sistemas ciberfísicos e Internet das Coisas (IoT). Destinado tanto a estudantes quanto a profissionais, o texto aborda conceitos essenciais e práticas de design, fornecendo uma base sólida para compreensão e aplicação desses princípios em uma variedade de contextos.

"Computational Thinking Education": de Kong et al., (2019), explora o papel do pensamento computacional na educação, fornecendo uma análise abrangente dos fundamentos e aplicações desse conceito. Os autores discutem como o pensamento computacional pode ser integrado de maneira eficaz no currículo educacional, tanto formal quanto informalmente, e oferecem *insights* sobre como promover uma compreensão mais profunda desses conceitos entre os alunos.

"Enriching Computer Science Programming Classes with Arduino Game Development" de Duch et al., (2018): aborda a enriquecedora integração do desenvolvimento de jogos com Arduino nas aulas de programação de ciência da computação. O estudo destaca como essa abordagem pode aumentar o interesse dos alunos e melhorar sua compreensão dos conceitos de programação, fornecendo uma experiência prática e envolvente. Ao incorporar o desenvolvimento de jogos com Arduino no currículo, as aulas se tornam mais dinâmicas e contextualizadas, preparando os alunos para enfrentar desafios reais na área de ciência da computação.

"Pensamento Computacional e Educação Matemática: Relações para o Ensino de Computação na Educação Básica" de Barcelos et al., (2012): explora a interseção entre o pensamento computacional e a educação matemática, destacando como esses dois campos podem ser integrados no ensino de computação nas escolas de ensino fundamental e médio. O texto explora como o ensino de conceitos computacionais pode promover

uma compreensão mais profunda e prática dos princípios matemáticos, oferecendo uma perspectiva sobre como a educação em computação pode ser integrada eficazmente no currículo escolar para melhorar o aprendizado em ambas as disciplinas.

"Towards the Tools Selection in Model Based System Engineering for Embedded Systems — A Systematic Literature Review" de Rashid et al., (2015): explora a seleção de ferramentas em engenharia de sistemas baseada em modelos para sistemas embarcados. Por meio de uma revisão sistemática da literatura, os autores investigam as melhores práticas na seleção de ferramentas para o desenvolvimento eficiente e confiável de sistemas embarcados. Esta pesquisa fornece informações valiosas para profissionais que trabalham com sistemas integrados e destaca a importância de escolher as ferramentas certas para garantir o sucesso do projeto.

"BIPES: Block Based Integrated Platform for Embedded Systems" de Junior et al., (2020): apresenta a plataforma BIPES, uma ferramenta inovadora no campo da educação STEM que simplifica o desenvolvimento de projetos de sistemas embarcados. O artigo destaca como o BIPES torna a programação mais acessível e envolvente para os alunos, contribuindo significativamente para o engajamento e aprendizado na área de STEM.

"Analysis of Factors Associated to the Enrollment and Demand of Computing-Related Careers" de Silva et al., (2018): examina os fatores associados à inscrição e à demanda de carreiras relacionadas à computação. Este estudo examina por que os alunos escolhem carreiras em ciência da computação e os desafios e oportunidades que enfrentam ao longo da carreira. Os autores fornecem uma análise abrangente de questões relacionadas à busca de uma carreira em ciência da computação e discutem implicações para a educação e os mercados de trabalho na área.

Por fim, *"Mindstorms: Children, Computers, and Powerful Ideas"* de Papert (1980): fornece uma perspectiva única sobre o uso de computadores na educação. Publicado em formato de livro, Papert explora como a tecnologia pode ser usada para promover ideias fortes e inovadoras nas mentes das crianças. Segundo ele, os computadores são poderosas ferramentas de aprendizagem que estimulam a criatividade, o pensamento crítico e a resolução de problemas.Os professores que desejam compreender o impacto da tecnologia na educação deveriam considerar a leitura do livro de Papert.

No que diz respeito a obras de relevância media temos o artigo *"Learning game development with Unity3D engine and Arduino microcontroller"*: o texto de Chursin et al., (2020) aborda o desenvolvimento de jogos utilizando a engine Unity3D e o microcontrolador Arduino. O estudo explora como integrar essas duas tecnologias para criar experiências de jogo interativas e inovadoras. Os autores demonstram como os desenvolvedores podem usar os recursos programáveis do Arduino e do Unity3D para criar jogos com maior funcionalidade e interação física, ao mesmo tempo que se beneficiam da flexibilidade deste último.

"Gifted students' learning experiences in systematic game development process in after-school activities": de Hava et al., (2020), analisa as experiências de aprendizagem de estudantes talentosos no processo de desenvolvimento de jogos de forma sistemática em atividades extracurriculares. O estudo investiga como alunos talentosos se envolvem com o desenvolvimento de jogos e como essa atividade pode impactar seu aprendizado e desenvolvimento. Os autores destacam a importância de oferecer oportunidades para que estudantes talentosos explorem sua criatividade e habilidades técnicas por meio do desenvolvimento de jogos, proporcionando um ambiente enriquecedor e desafiador para o seu crescimento educacional e pessoal.

"Model-driven Game Development: A Literature Review" de Clarke et al., (2019): apresenta uma revisão da literatura sobre desenvolvimento de jogos orientado a modelos (Model-Driven Game Development - MDGD). O artigo analisa como abordagens baseadas em modelagem têm sido aplicadas no contexto de desenvolvimento de jogos digitais, com o objetivo de melhorar a produtividade, modularidade e reutilização de componentes. Os autores identificam lacunas na literatura e sugerem direções para futuras pesquisas, evidenciando o potencial do MDGD para transformar práticas na criação de jogos educacionais e comerciais.

"Uma Introdução à Internet das Coisas e Sistemas Embarcados utilizando BIPES: Uma Proposta para Iniciação Científica no Ensino Médio" de Marini et al., (2021): propõe o uso da plataforma BIPES como ferramenta introdutória para o ensino de IoT e sistemas embarcados no ensino médio. O artigo detalha uma abordagem prática para iniciação científica, com ênfase em acessibilidade e motivação dos estudantes por meio de programação em blocos. O estudo destaca como a plataforma contribui para o desenvolvimento do pensamento computacional e o engajamento em temas contemporâneos de tecnologia.

"Programming Embedded Systems: With C and MicroPython" de Valvano (2021): é um livro técnico que oferece uma introdução prática ao desenvolvimento de sistemas embarcados utilizando as linguagens C e MicroPython. Destinado a estudantes e profissionais, o texto combina fundamentos teóricos com aplicações práticas, explorando desde o controle de hardware até estratégias de programação eficiente. É especialmente relevante para a formação em engenharia e ciência da computação com foco em sistemas embarcados.

"Pensamento computacional na formação de professores: experiências e desafios encontrados no ensino da computação em escolas públicas": escrito por França et al., (2017), o texto explora a importância do pensamento computacional na formação de professores, especialmente em relação ao ensino da computação nas escolas públicas. O texto aborda as experiências e desafios enfrentados pelos educadores ao incorporar o pensamento computacional nas suas práticas de ensino, destacando a necessidade de desenvolver habilidades computacionais entre os professores para melhor prepará-los para

orientar os alunos nesse campo em constante evolução.

"Student Engagement Declines in STEM": de Wester et al., (2021) analisa o declínio do engajamento dos estudantes de graduação em STEM durante o aprendizado remoto impulsionado pela COVID-19. O estudo destaca os desafios enfrentados pelos estudantes de STEM durante o período de ensino à distância e investiga as causas subjacentes desse declínio no engajamento. Os autores oferecem ideias e recomendações para mitigar os efeitos negativos do ensino à distância na educação STEM.

Interested, Disinterested, or Neutral: Exploring STEM Interest Profiles and Pathways in A Low-Income Urban Community": escrito por Staus et al., (2020) o estudo examina os perfis de interesse em STEM em uma comunidade urbana de baixa renda. Para isso, o trabalho investiga os fatores que influenciam o interesse dos alunos em STEM, considerando as circunstâncias socioeconômicas específicas dessa comunidade. Os autores realizam uma análise detalhada dos diferentes perfis de interesse e das trajetórias educacionais associadas, fornecendo percepções importantes para o desenvolvimento de programas e políticas educacionais inclusivas e voltadas para equidade.

"Reasserting US Leadership in Microelectronics: The Role of Universities": é um relatório do MIT que destaca a importância das universidades na produção de profissionais em STEM para a indústria de semicondutores. Propõe medidas como disciplinas multidisciplinares, cursos práticos, pesquisa, design com CAD, estágios e mentoria para atrair e capacitar estudantes, fortalecendo a competitividade dos EUA.

Finalizando, o artigo *"Gender differences in high school students' interest in STEM careers: a multi-group comparison based on structural equation model"*: examina as diferenças de gênero no interesse de estudantes do ensino médio por carreiras em STEM. Escrito por Wang et al., (2023), o estudo utiliza um modelo de equação estrutural para realizar uma comparação entre grupos múltiplos. Os autores investigam os fatores que influenciam o interesse dos alunos em carreiras STEM, considerando diferenças de gênero e fornecem conhecimentos essenciais para promover a igualdade de oportunidades e a inclusão de ambos os gêneros nas áreas STEM.

Diversos estudos apontam que atividades práticas e envolventes, como oficinas de desenvolvimento de jogos e projetos "mão na massa", têm grande potencial para estimular o interesse dos alunos por áreas STEM. Ao permitir que os estudantes construam, programem e testem suas próprias criações — como jogos físicos com componentes eletrônicos — essas atividades tornam o aprendizado mais significativo, promovendo autonomia, criatividade e pensamento crítico (XU; JIN, 2021; DUCH; JAWORSKI, 2018; HAVA; GüYER; CAKIR, 2020). Além disso, tais experiências favorecem uma melhor compreensão dos conteúdos técnicos ao conectá-los a aplicações concretas e lúdicas. De acordo com Duch et al. (2018), a integração entre desenvolvimento de jogos e plataformas como Arduino contribui para tornar as aulas de programação mais dinâmicas e contextualizadas. Hava et al. (2020)

também destacam que o desenvolvimento sistemático de jogos em ambientes extracurriculares oferece oportunidades de aprofundamento e crescimento pessoal, especialmente entre estudantes talentosos. Assim, iniciativas pedagógicas que unem tecnologia, criatividade e prática podem desempenhar um papel fundamental no aumento do engajamento e da permanência de jovens nas áreas de ciência, tecnologia, engenharia e matemática.

Nossa proposta vai além dos trabalhos existentes, ao oferecer a possibilidade de os próprios alunos do ensino fundamental e médio criarem seus jogos físicos e interativos com botões, displays, sons e outras funcionalidades no mundo físico. Ao permitir que os alunos desenvolvam seus próprios jogos físicos, estamos promovendo não apenas a aprendizagem de conceitos de programação e eletrônica, mas também estimulando a criatividade, o pensamento crítico e a resolução de problemas de forma prática e envolvente. Essa abordagem não apenas enriquece o processo de aprendizagem, mas também capacita os alunos a se tornarem criadores ativos e inovadores de tecnologia.

3 . ABORDAGEM PROPOSTA

A proposta visa atrair novos talentos para as áreas STEM ao oferecer uma abordagem prática e envolvente para a aprendizagem. Ao permitir que os alunos desenvolvam seus próprios jogos físicos e interativos, utilizando botões, displays, sons e outras funcionalidades no mundo físico, a abordagem visa despertar o interesse e a curiosidade dos estudantes em disciplinas relacionadas à STEM. Essa abordagem lúdica e hands-on não apenas estimula a criatividade e o pensamento crítico dos alunos, mas também os capacita com habilidades práticas e relevantes para o mercado de trabalho STEM, preparando-os para enfrentar os desafios e oportunidades do futuro.

Ao fim do projeto, esperasse que os alunos consigam implementar um jogo simples como, por exemplo, o pong mostrado na figura 3.

Figura 3 – Jogo Pico pong.

Já na figura 4 é mostrado uma primeira forma para a montagem do circuito para ser possível criar o jogo.

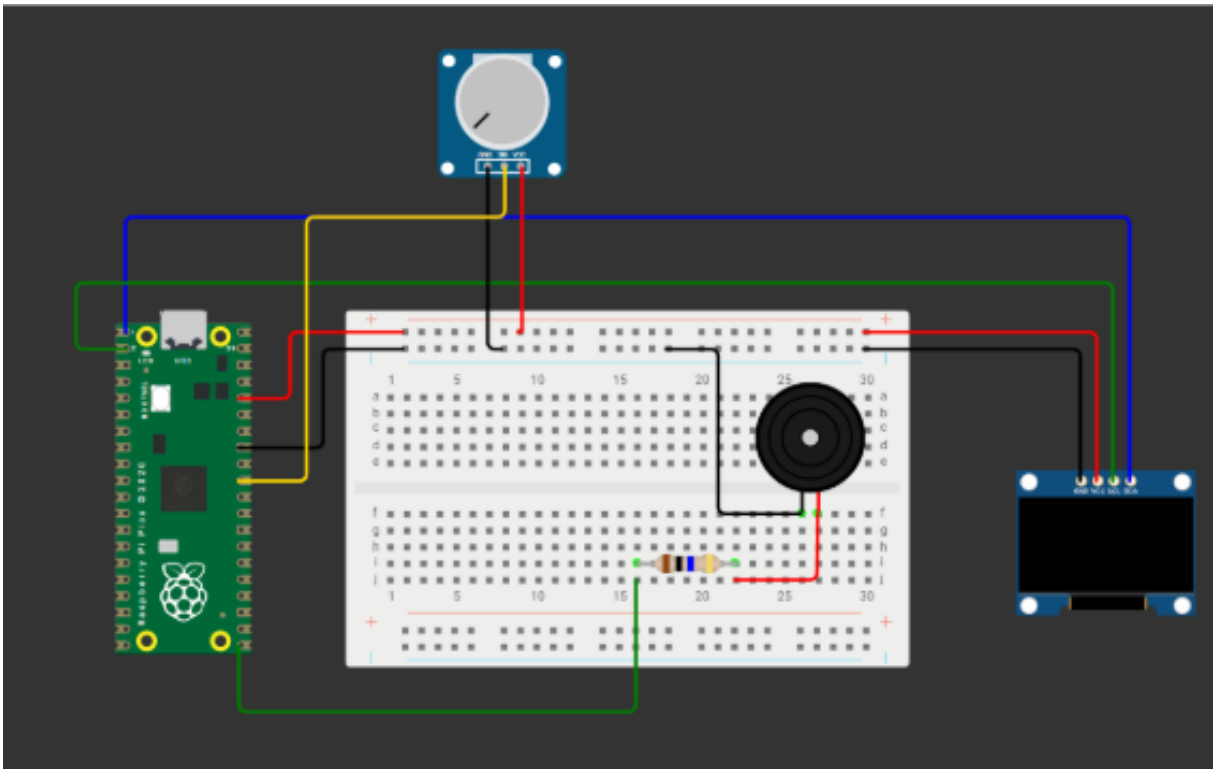


Figura 4 – Circuito do pico pong.

Para a implementação dos jogos físicos no contexto deste projeto, utilizou-se a linguagem *MicroPython* como mostrado na Figura 5. Trata-se de uma versão otimizada da linguagem Python, projetada especificamente para rodar em microcontroladores e sistemas embarcados com recursos limitados (GEORGE, 2025). O *MicroPython* combina a simplicidade e legibilidade do Python com a capacidade de manipular diretamente pinos e periféricos do hardware, tornando-se uma ferramenta poderosa para aplicações educacionais em STEM. O microcontrolador utilizado neste projeto, o Raspberry Pi Pico, foi previamente configurado com o interpretador *MicroPython* gravado em sua memória flash, permitindo que os alunos desenvolvam, testem e executem seus códigos diretamente no dispositivo. Essa abordagem oferece aos estudantes uma introdução acessível à programação embarcada, ao mesmo tempo que os familiariza com práticas e ferramentas utilizadas em contextos profissionais. A escolha do *MicroPython* também se alinha às tendências atuais da educação em computação, que priorizam linguagens de alto nível, com ampla documentação e suporte da comunidade, favorecendo o processo de ensino-aprendizagem (MARWEDEL, 2021; JUNIOR et al., 2020).

```
# Pong game on Raspberry Pi Pico with a OLED and two Potentimeters
from machine import Pin, PWM, SPI
import ssd1306
from utime import sleep
import random # random direction for new ball

spi_sck=machine.Pin(2)
spi_tx=machine.Pin(3)
spi=machine.SPI(0,baudrate=100000,sck=spi_sck, mosi=spi_tx)
CS = machine.Pin(1)
DC = machine.Pin(4)
RES = machine.Pin(5)
oled = ssd1306.SSD1306_SPI(128, 64, spi, DC, RES, CS)
# connect the center tops of the potentiometers to ADC0 and ADC1
pot_pin_1 = machine.ADC(26)
pot_pin_2 = machine.ADC(26) # make them the same for testing

# lower right corner with USB connector on top
SPEAKER_PIN = 16
# create a Pulse Width Modulation Object on this pin
speaker = PWM(Pin(SPEAKER_PIN))

# globals variables
# static variables are constants are uppercase variable names
WIDTH = 128
```

Figura 5 – Código parcial do pico pong. Adaptado de (STEVIESTL, 2021)

Como ilustrado na Figura 4, a construção de um jogo físico do tipo “PONG” pode ser realizada de forma relativamente simples e com baixo custo, utilizando a placa Raspberry Pi Pico — um microcontrolador amplamente difundido e de fácil acesso no mercado — em conjunto com alguns fios de conexão (*jumper cables*) e componentes eletrônicos básicos. Dentre os elementos utilizados para controlar a dinâmica do jogo, destacam-se dois potenciômetros, conectados aos pinos analógicos da placa, que funcionam como controladores das “raquetes” do jogo. Ao girar os potenciômetros, o jogador consegue mover a raquete para cima e para baixo, simulando o controle clássico do jogo PONG de forma física e interativa.

Embora o hardware envolvido na construção do jogo seja simples, a programação da placa geralmente requer conhecimentos prévios em áreas como linguagem Python, firmware, bootloaders, lógica de programação e manipulação de interfaces físicas — o que representa uma barreira de entrada considerável para alunos iniciantes ou com pouca familiaridade com sistemas embarcados. O código Python utilizado para controlar o funcionamento do jogo (conforme apresentado na Figura 5) implementa rotinas de leitura analógica dos potenciômetros, lógica de movimentação da bola e detecção de colisões, além do controle do display para exibir o jogo.

Neste contexto, a plataforma BIPES (JUNIOR et al., 2020) oferece uma alter-

nativa poderosa e acessível, permitindo que todo o código apresentado em Python seja implementado de forma visual, por meio de blocos de programação criados especificamente para este fim. Esses blocos encapsulam funcionalidades complexas — como leitura de entradas analógicas, controle de saídas digitais, atualizações em displays e execução de laços de repetição — e tornam possível que os alunos desenvolvam projetos completos sem a necessidade de escrever uma única linha de código textual. A utilização do BIPES elimina etapas técnicas como a configuração do ambiente de desenvolvimento, gravação de firmware e tratamento de erros sintáticos, permitindo que o foco do aluno permaneça na lógica do projeto e na compreensão dos conceitos fundamentais de programação e eletrônica.

Além disso, o BIPES pode ser executado diretamente no navegador, sem necessidade de instalação local, o que o torna altamente acessível para ambientes educacionais com recursos computacionais limitados. Dessa forma, um estudante pode projetar, construir e programar todas as etapas de um jogo PONG, visualizando os resultados em tempo real e desenvolvendo, ao longo do processo, habilidades essenciais em pensamento lógico, criatividade, resolução de problemas e familiaridade com sistemas embarcados — todas competências valorizadas nas áreas de STEM.

A ideia é que jogos semelhantes a esse exemplo, e outros similares, possam ser criados pelos alunos utilizando-se blocos e outras ferramentas desenvolvidas no projeto. Essa abordagem simplificada e acessível visa remover barreiras técnicas e tornar o processo de criação de jogos mais acessível, o projeto visa democratizar o acesso à educação em STEM.

Pensando no contexto de jogos feitos com blocos, é possível encontrar alguns exemplos a respeito. Dentre eles temos o site blockly games, o qual disponibiliza alguns jogos (figura 6) que possuem o intuito de familiarizar os usuários com programação em blocos.

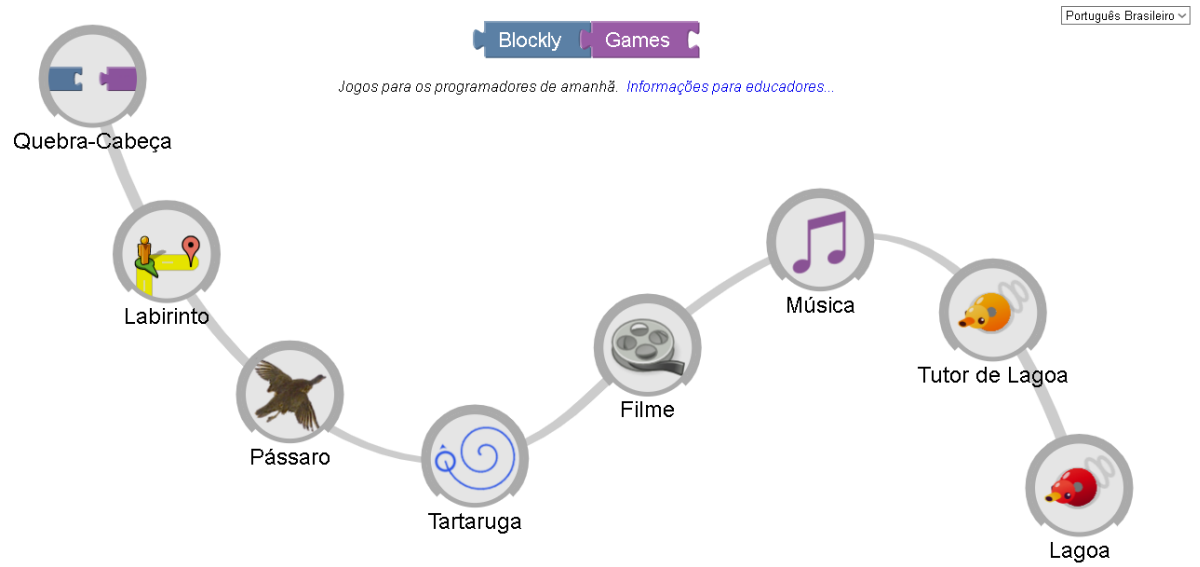


Figura 6 – Jogos disponíveis na blockly games. Adaptado de (GOOGLEDEVELOPERS, 2019)

Um exemplo é mostrado na figura 7, no qual o usuário deve usar os blocos disponíveis para fazer com que o personagem sai de dentro do labirinto. Além disso, ao fim do desafio é mostrado como seria o código (figura 8) correspondente para executar a mesma tarefa.

Figura 7 – Jogo de labirinto do blockly. Adaptado de (GOOGLEDEVELOPERS, 2019)

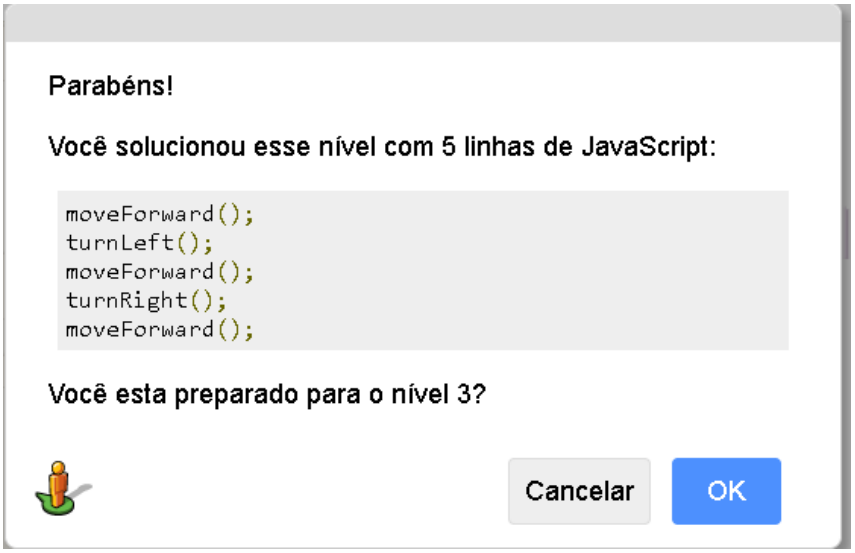


Figura 8 – Código equivalente aos blocos. Adaptado de (GOOGLEDEVELOPERS, 2019)

4 . MATERIAIS E MÉTODOS

Nesta seção, serão apresentados os materiais utilizados no projeto, como também a justificativa do uso de cada um.

Considerando que será feito o primeiro contato de muitos jovens com as áreas STEM, viu-se necessário escolher uma plataforma de programação que seja de fácil entendimento mesmo para quem nunca teve uma experiência prévia. Devido a isso, escolheu-se a plataforma BIPES por possibilitar a programação em blocos, que é bem mais intuitiva do que a programação em linhas de código.Além disso, o BIPES possui versões em várias línguas, inclusive em inglês, facilitando o entendimento e interação com estudantes brasileiros que conhecem pouco a língua inglesa.

Além disso, é necessário disponibilizar o maior número possível de componentes que poderão ser utilizados pelos alunos. Essa necessidade surge para ser possível abranger mais projetos e ideias propostas pelos alunos, como também para mostrar a grande diversidade de possibilidades da área.

Dentre os materiais que serão disponibilizados, estão os microcontroladores, a ESP32 e a Raspberry Pi Pico mostradas nas figuras 9 e 10. Num primeiro momento cogitou-se utilizar a ESP8266 (figura 11 devido a sua conectividade Wi-Fi embutida e baixo custo, mas devido a problemas que serão discutidos da próxima seção, ela foi excluída. A ESP8266 é conhecida por sua conectividade Wi-Fi embutida e baixo custo, sendo uma opção popular para projetos IoT simples que exigem comunicação sem fio. A ESP32, por sua vez, oferece maior capacidade de processamento, além de Wi-Fi e Bluetooth integrados, sendo ideal para projetos mais complexos. Por fim, a Raspberry Pi Pico, com seu processador ARM Cortex-M0+ de 32 bits, é versátil e poderosa, adequada para uma variedade de aplicações de hardware.

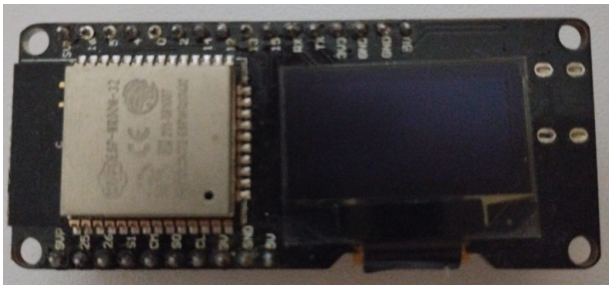


Figura 9 – ESP32 com display OLED

Figura 10 – Raspberry Pi Pico (PiPico)

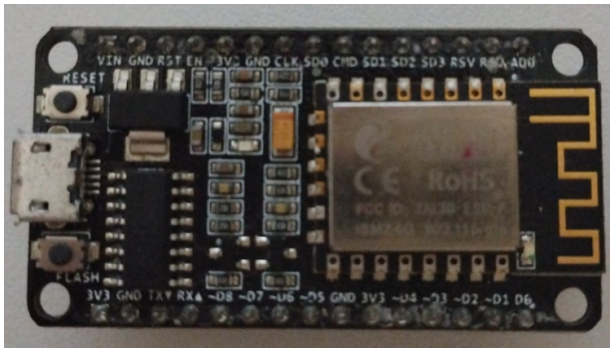


Figura 11 – ESP8266

Além dos microcontroladores, serão incluídos outros componentes essenciais para enriquecer a experiência do projeto. Entre esses componentes, destacam-se os displays OLED *(Organic Light-Emitting Diode)*. Eles são conhecidos por proporcionar cores mais vibrantes, pretos mais profundos e ângulos de visão mais amplos. Além disso, sua rápida resposta e eficiência energética tornam-nos ideais para aplicações em jogos e dispositivos interativos. Além disso, *buzzers*, que são dispositivos de áudio que permitem a reprodução de efeitos sonoros e adicionam uma dimensão sonora à interação do usuário, serão incluídos como um recurso adicional.

Adicionalmente, Também é possivel utilizar luzes LED, que podem ser empregadas tanto para oferecer feedback visual quanto para iluminar o ambiente do projeto. Essas luzes podem ser programadas para indicar diferentes estados do sistema ou para criar efeitos visuais interessantes, tornando a interação mais envolvente e estimulante.

Além disso, serão integradas teclas e botões, que servirão como controles para os usuários interagirem com o sistema. Esses controles podem ser usados para acionar diferentes funcionalidades do projeto, proporcionando uma maneira intuitiva e direta de operar o dispositivo.

Por fim, serão utilizados *power banks* ou baterias, que são dispositivos de armazenamento de energia portáteis, para alimentar os projetos. Isso permite que os dispositivos sejam móveis e independentes de uma fonte de energia fixa, aumentando sua versatilidade e facilitando o uso em diferentes ambientes e situações. A inclusão desses componentes

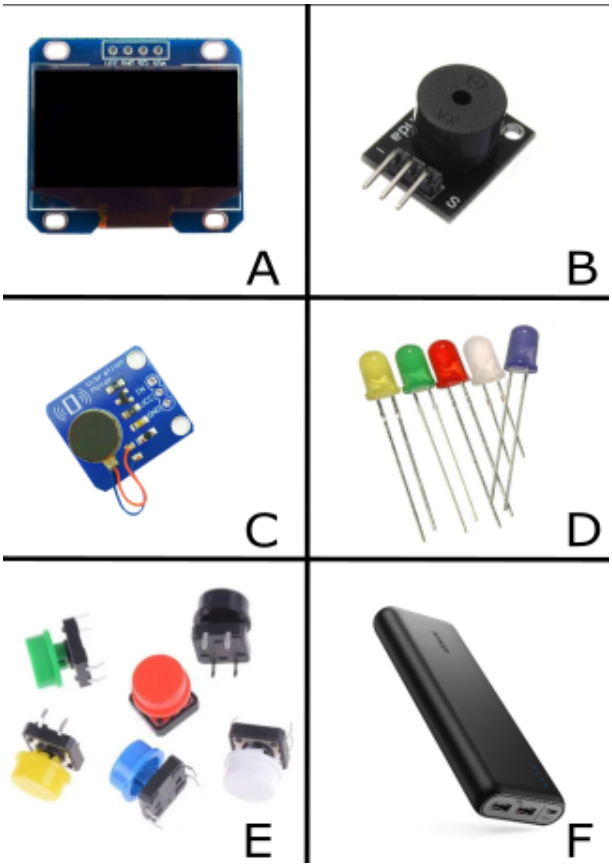


Figura 12 – ComponentesComponentes a serem utilizados, A:display OLED, B: *buzzer*, C: vibrador, D: luz LED, E: botões, F: *power bank*.

adicionais amplia as possibilidades de interação e uso dos projetos, proporcionando uma experiência mais completa e enriquecedora para os usuários. Na figura 12 é mostrado cada um dos componentes citados anteriormente, onde o display OLED é representado pela letra A, o *buzzer* por B, vibrador por C, luz LED por D, os botões pela letra E e por fim, o *power bank* é assinalado pela letra F.

O primeiro passo necessário é identificar quais tipos de blocos serão úteis para a criação de jogos e para tornar os projetos mais atrativos. Para isso, é fundamental realizar uma análise detalhada das funcionalidades necessárias para o desenvolvimento dos jogos, considerando os objetivos específicos de cada projeto. Uma vez definidos os blocos necessários, é importante buscar bibliotecas adequadas que ofereçam suporte para a implementação desses blocos no ambiente do BIPES.

É crucial considerar os requisitos das bibliotecas selecionadas, garantindo que sejam compatíveis com as placas disponíveis para uso no BIPES. Essa compatibilidade é essencial para assegurar o bom funcionamento dos blocos e a integração adequada com o ambiente de programação. Caso contrário, podem surgir problemas de incompatibilidade que comprometam o desenvolvimento dos projetos.

Um caso de insuficiência de requisitos será apresentado no próximo capítulo,

destacando a importância de uma seleção criteriosa das bibliotecas e uma avaliação cuidadosa de suas capacidades e limitações.

Após identificar e garantir a compatibilidade das bibliotecas selecionadas, o próximo passo será criar os blocos novos necessários para implementar as funcionalidades desejadas nos jogos. Essa etapa envolverá o desenvolvimento dos blocos no ambiente do BIPES, seguindo as especificações e requisitos previamente estabelecidos.Para se ter uma melhor visualização, serão apresentados visualmente alguns blocos que serão criados.

Na figura 13 é mostrado o bloco que poderá ser usado para criar formas geométricas, mais especificamente retângulos. Tal bloco pode servir para criar objetos, partes dos cenários ou até mesmo delimitadores nos jogos.

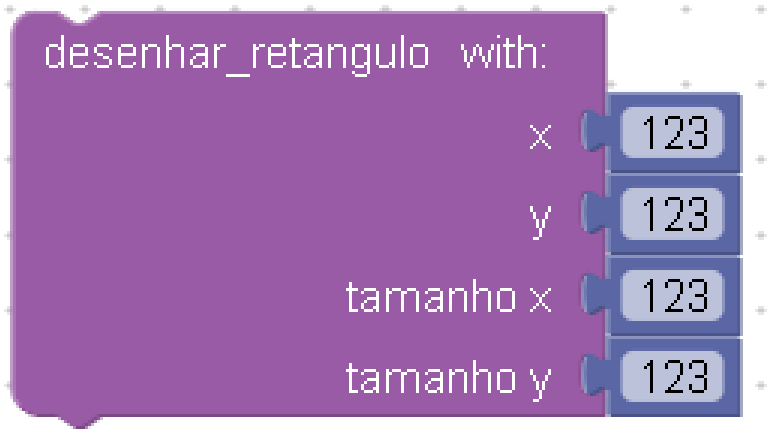


Figura 13 – Bloco para criar retângulos

Já a figura 14 mostra o bloco cujo objetivo é mover objetos do jogo, seja a "raquete"do jogo Pong ou o personagem do labirinto apresentados anteriormente.

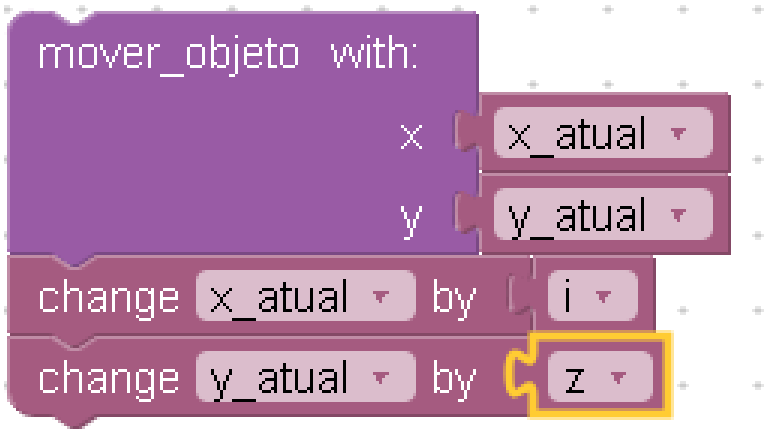


Figura 14 – Bloco para mover objetos

Para averiguar o funcionamento correto dos blocos criados, será realizado um conjunto de testes e verificações. Esses testes visam garantir que os blocos atendam aos

requisitos de funcionalidade e desempenho estabelecidos, além de verificar sua integração adequada com o ambiente de desenvolvimento do BIPES. Essa etapa é essencial para assegurar a qualidade e a eficácia dos blocos, bem como para verificar se estão aptos a cumprir os objetivos propostos.

Por fim, utilizando-se dos novos blocos e dos componentes disponíveis, será criado um jogo que servirá de exemplo para os alunos, servindo assim como um motivacional e demonstrando um pouco das inúmeras possibilidades que o BIPES traz.

4.1 Jogos Escolhidos

4.1.1 Pong

O jogo escolhido para foi o *Pong*, um clássico dos videogames que simula uma partida de tênis de mesa. Nele, uma bola se movimenta continuamente pela tela, enquanto cada jogador controla a posição vertical de uma raquete localizada nas extremidades esquerda e direita do mapa do jogo. O objetivo consiste em utilizar a raquete para interceptar e "rebater"a bola, evitando que ela ultrapasse a linha de fundo, ao mesmo tempo em que se busca marcar pontos quando o adversário falha na defesa.

Embora originalmente concebido para dois jogadores, o *Pong* também pode ser adaptado para um único participante, que passa a controlar simultaneamente as duas raquetes. Nesse modo, o desafio não reside na competição, mas sim na tentativa de impedir que a bola ultrapasse qualquer um dos lados, mantendo o jogo em andamento pelo maior tempo possível.

Para a implementação do jogo Pong com a Raspberry Pi Pico, foi realizada uma montagem física utilizando componentes eletrônicos simples e de fácil acesso. Esta seção descreve, passo a passo, a montagem do circuito responsável pela interface entre o hardware e o software desenvolvido.

4.1.2 Space Invader

Outro jogo escolhido foi o *Space Invader*, um dos clássicos mais influentes da história dos videogames, que simula uma batalha espacial entre um jogador e um exército de inimigos. Nele, o jogador controla uma nave que se movimenta horizontalmente na parte inferior da tela, enquanto dispara projéteis para eliminar as fileiras de inimigos que descem progressivamente em direção à sua posição.

O objetivo do *Space Invader* consiste em destruir todos os inimigos antes que eles alcancem a parte inferior da tela ou que atinjam a nave do jogador com seus próprios disparos. A dificuldade aumenta à medida que o número de inimigos diminui, pois sua velocidade de movimentação tende a aumentar, tornando a dinâmica do jogo mais desafiadora.

Embora originalmente projetado para arcades, o *Space Invader* pode ser facilmente adaptado para sistemas embarcados, permitindo a utilização de componentes eletrônicos simples e acessíveis para sua implementação.

Para a implementação do jogo *Space Invader* com a Raspberry Pi Pico, foi realizada uma montagem física utilizando um display OLED para exibição gráfica e controles de entrada analógicos, como potenciômetros, para o movimento da nave. Esta seção descreve, passo a passo, a montagem do circuito responsável pela interface entre o hardware e o software desenvolvido. Além disso, cabe destacar que neste projeto foi implementado uma versão simplificada do jogo, onde não existe aumento da velocidade dos inimigos e eles também não disparam tiros.

4.2 Montagem Física do Pong

A montagem física do jogo desenvolvida neste trabalho constitui uma etapa essencial para a materialização do protótipo e a aplicação prática dos conceitos de sistemas embarcados. A escolha dos componentes, assim como a organização do circuito, foi fundamentada em critérios de simplicidade, baixo custo e acessibilidade, a fim de promover um ambiente de aprendizagem alinhado com a proposta construcionista de (PAPERT, 1980) e com as diretrizes contemporâneas de educação STEM (MATA; TOBON, 2018).

4.2.1 Materiais Utilizados

Para a montagem do circuito, foram selecionados os seguintes componentes, conforme a Tabela 2:

Tabela 2 – Componentes utilizados na montagem física do Pong.

Componente	**Função**
Raspberry Pi Pico	Microcontrolador central
Display OLED (SSD1306, 128x64, I2C)	Exibição gráfica do jogo
2 Potenciômetros	Controle analógico das barras do jogo
Protoboard	Montagem do circuito sem solda
Buzzer	Feedback sonoro
Resistor	Proteção do buzzer
Fios de conexão	Interligação dos componentes

A escolha da Raspberry Pi Pico deve-se à sua ampla disponibilidade, baixo custo, consumo energético reduzido e facilidade de programação via MicroPython (AROCA, 2022). O display OLED foi selecionado por apresentar alto contraste e boa resposta temporal, essenciais para a fluidez do jogo (DAVIES, 2021).

4.2.2 Procedimentos de Montagem

A montagem seguiu uma sequência lógica e sistemática, com foco na organização e segurança das conexões, visando minimizar erros comuns no processo de prototipagem (MARWEDEL, 2021).

1. **Fixação dos Componentes:** A Raspberry Pi Pico e o display OLED foram conectados firmemente na protoboard, garantindo o isolamento adequado das trilhas, evitando interferências elétricas que poderiam comprometer a integridade do circuito.

2. **Alimentação do Circuito:** A alimentação foi realizada a partir do pino `3V3(OUT)` da Raspberry Pi Pico, distribuída através das trilhas laterais da protoboard, com o `GND` conectado ao barramento negativo.

3. **Conexão do Display OLED:**

   - Pino `VCC` do OLED conectado ao barramento de `3V3`.
   - Pino `GND` conectado ao barramento de `GND`.
   - Pino `SDA` ligado ao `GP0` (pino físico 1) da Raspberry Pi Pico.
   - Pino `SCL` ligado ao `GP1` (pino físico 2).

   A interface escolhida foi a I2C, por sua simplicidade e redução no número de fios necessários.

4. **Conexão dos Potenciômetros:** Cada potenciômetro foi conectado conforme padrão:

   - Um terminal externo ao `GND`.
   - O outro terminal externo ao `3V3(OUT)`.
   - Terminal central ao pino analógico: `ADC0` (`GP26`, pino 31) e `ADC1` (`GP27`, pino 32) para o segundo jogador.

   Esta configuração permite um mapeamento linear dos valores lidos para as posições das barras no jogo, conforme implementado na função `valmap`.

5. **Configuração do Buzzer:** O buzzer foi posicionado na protoboard com:

   - Terminal positivo conectado a um resistor, que por sua vez foi conectado ao pino `GP16` (pino físico 21) da Raspberry Pi Pico.
   - Terminal negativo ao `GND`.

   O uso do resistor visa proteger o circuito de sobrecorrente e garantir a durabilidade do componente (STEVIESTL, 2021).

6. **Configuração do Modo de Jogo:** A montagem permite o funcionamento em modo de um ou dois jogadores:

   - Para um jogador, apenas o primeiro potenciômetro é necessário.
   - Para dois jogadores, conecta-se o segundo potenciômetro e ajusta-se a configuração no código-fonte.

   Esta flexibilidade amplia as possibilidades pedagógicas do projeto, permitindo adaptações conforme o público-alvo.

O circuito completo (Figura 15) proporciona a interface necessária para o funcionamento do jogo Pong, permitindo a interação via potenciômetros e a emissão de sons através do buzzer, com a exibição gráfica realizada no display OLED.

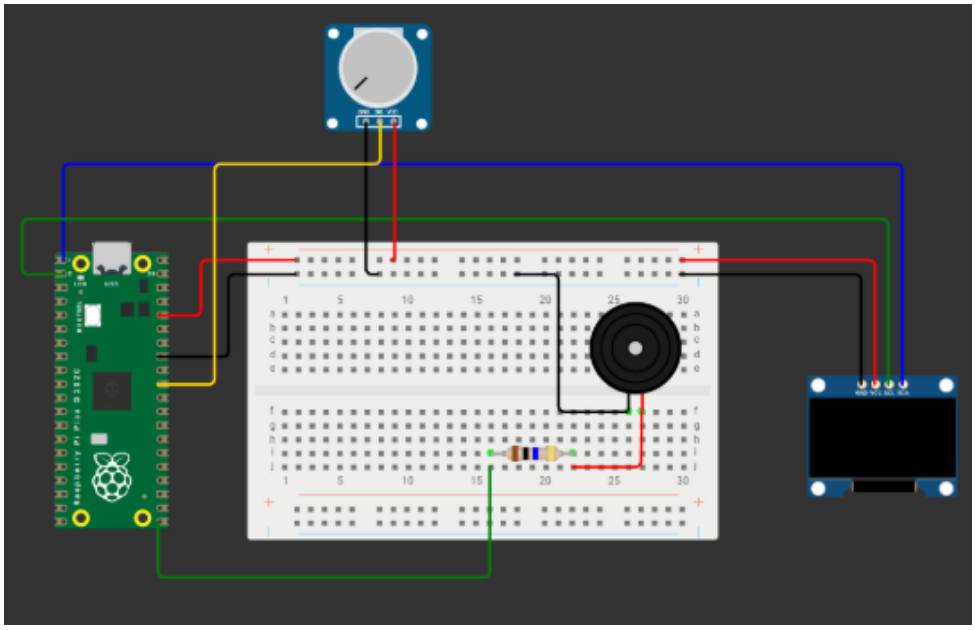


Figura 15 – Montagem final do jogo com Raspberry Pi Pico e display OLED.

4.3 Montagem Física do Space Invader

A montagem física do **Space Invader**, seguiu de forma semelhante ao do Pong, diferenciando-se pelo uso de 3 botões no lugar dos potenciômetros e por não usar nenhum **buzzer**.

4.3.1 Materiais Utilizados

Para a montagem do circuito, foram selecionados os seguintes componentes, conforme a Tabela 3:

Tabela 3 – Componentes utilizados na montagem física do **Space Invader**.

Componente	**Função**
Raspberry Pi Pico	Microcontrolador central
Display OLED (SSD1306, 128x64, I2C)	Exibição gráfica do jogo
3 Botões	Controle da movimentação e disparo do jogador
Protoboard	Montagem do circuito sem solda
3 Resistores de 10kΩ	Estabilização do sinal dos botões
Fios de conexão	Interligação dos componentes

A escolha da Raspberry Pi Pico deve-se à sua ampla disponibilidade, baixo custo, consumo energético reduzido e facilidade de programação via MicroPython (AROCA, 2022). O display OLED foi selecionado por apresentar alto contraste e boa resposta temporal, essenciais para a fluidez do jogo (DAVIES, 2021).

4.3.2 Procedimentos de Montagem

A montagem seguiu uma sequência lógica e sistemática, com foco na organização e segurança das conexões, visando minimizar erros comuns no processo de prototipagem (MARWEDEL, 2021).

1. **Fixação dos Componentes:** A Raspberry Pi Pico e o display OLED foram conectados firmemente na protoboard, garantindo o isolamento adequado das trilhas e evitando interferências elétricas que poderiam comprometer a integridade do circuito.

2. **Alimentação do Circuito:** A alimentação foi realizada a partir do pino `3V3(OUT)` da Raspberry Pi Pico, distribuída através das trilhas laterais da protoboard, com o `GND` conectado ao barramento negativo.

3. **Conexão do Display OLED:**

   - Pino `VCC` do OLED conectado ao barramento de `3V3`.
   - Pino `GND` conectado ao barramento de `GND`.
   - Pino `SDA` ligado ao `GP0` (pino físico 1) da Raspberry Pi Pico.
   - Pino `SCL` ligado ao `GP1` (pino físico 2).

   A interface escolhida foi a I2C, por sua simplicidade e redução no número de fios necessários.

4. **Conexão dos Botões:** Os três botões foram conectados diretamente aos pinos GPIO da Raspberry Pi Pico, conforme descrito a seguir:

   - Botão de movimentação para a esquerda: pino `GP14` (pino físico 19).

- Botão de movimentação para a direita: pino `GP15` (pino físico 20).
- Botão de disparo: pino `GP13` (pino físico 17).

Cada botão foi conectado com um terminal ao `GND` e o outro ao pino correspondente da Pico.

5. **Instalação dos Resistores:** Embora o código do jogo configure os resistores de *pull-up* internos da Raspberry Pi Pico, foram adicionados resistores de `10kΩ` externos como *pull-down* para garantir maior estabilidade na leitura dos sinais. Cada resistor foi conectado entre o terminal do botão ligado ao pino da Pico e o `GND`, conforme indicado no roteiro de montagem.

6. **Finalização:** Após a realização de todas as conexões, o circuito ficou apto para executar o jogo *Space Invader*, integrando as entradas de controle com a exibição gráfica no display OLED.

O circuito completo do jogo *Space Invader* é mostrado na Figura 16)

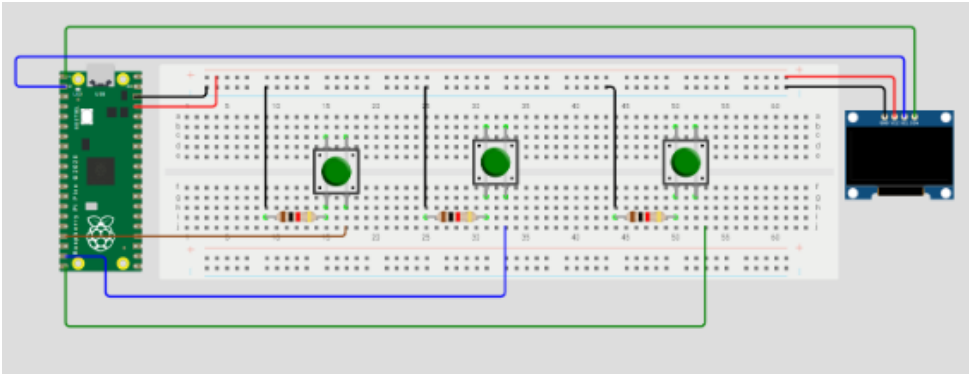


Figura 16 – Montagem final do Space Invader.

4.3.3 Importância Educacional da Montagem

A montagem física do jogo é coerente com os princípios do *learning by doing*, propostos por Papert (1980), onde o aprendizado ocorre a partir da experimentação prática. Ao permitir que os estudantes manipulem fisicamente componentes como potenciômetros, displays e buzzers, fomenta-se o desenvolvimento de habilidades motoras finas, raciocínio lógico e competências associadas ao pensamento computacional (KONG; ABELSON, 2019).

Além disso, esta montagem evidencia o potencial do uso de jogos físicos no ensino de sistemas embarcados, promovendo uma aprendizagem interdisciplinar que integra eletrônica, programação e design (DUCH; JAWORSKI, 2018; HAVA; GÜYER; CAKIR, 2020). Tais abordagens têm se mostrado eficazes para elevar o engajamento e a motivação dos alunos em áreas STEM (ARZTMANN et al., 2022).

4.3.4 Considerações Técnicas e de Segurança

Durante a montagem, foram observadas as seguintes boas práticas recomendadas para trabalhos com prototipagem eletrônica (MARWEDEL, 2021):

- Verificação prévia das ligações de alimentação, evitando curtos-circuitos.
- Teste individual de cada componente antes da inserção no circuito completo.
- Organização dos fios na protoboard, facilitando futuras manutenções ou modificações.
- Evitar tensão e corrente superiores às especificadas pelos fabricantes dos componentes.

Adicionalmente, destaca-se que a simplicidade e modularidade do circuito facilitam sua replicação por estudantes de diferentes níveis de escolaridade, democratizando o acesso ao conhecimento em sistemas embarcados.

4.4 Criação de Blocos

Como dito anteriormente, a implementação do jogo no ambiente BIPES demandou a utilização de blocos de programação que, até então, não estavam disponíveis na plataforma. Por esse motivo, foi essencial compreender e aplicar o processo de criação de novos blocos, ampliando as funcionalidades do sistema e permitindo que o desenvolvimento do jogo ocorresse de forma mais intuitiva e acessível.

A criação de blocos no BIPES é uma tarefa que envolve o domínio de conceitos relacionados à programação visual, utilizando a biblioteca `Blockly` como base (FRASER, 2013), bem como o entendimento da estrutura de geração de código e integração com o ambiente de desenvolvimento.

De maneira geral, o processo pode ser dividido em três etapas principais, conforme descrito a seguir:

4.4.1 Geração do Código do Bloco

Nesta etapa, é realizada a definição da funcionalidade do bloco, ou seja, especifica-se qual será o código equivalente gerado em `MicroPython` quando o bloco for utilizado na programação visual. Esta associação é fundamental para garantir que o bloco criado tenha comportamento coerente com as necessidades do projeto e com a sintaxe da linguagem alvo.

A geração de código é implementada no arquivo `generator_stubs.js`, onde cada bloco possui uma função de conversão que transforma a configuração visual definida pelo usuário em uma instrução de código textual válida. Por exemplo, ao criar um bloco que

desenha uma linha no display OLED, deve-se garantir que ele gere adequadamente a instrução `oled.line(x0, y0, x1, y1, cor)` no código final.

4.4.2 Definição Estrutural do Bloco

Após a definição do código, procede-se à estruturação do bloco propriamente dita, especificando seus aspectos visuais e funcionais. Esta etapa é realizada no arquivo `block_definitions.js`, onde são declaradas:

- O **tipo do bloco**, que serve como identificador único.
- As **entradas e parâmetros**, determinando quais valores o usuário poderá configurar diretamente na interface gráfica (por exemplo, coordenadas, mensagens ou variáveis).
- As **saídas**, indicando se o bloco possui valor de retorno ou se é um bloco de ação.
- As **restrições de conexão**, que definem de que forma o bloco pode ser encadeado a outros blocos no ambiente visual.

Além disso, nesta etapa são configuradas propriedades estéticas, como cores, ícones e textos explicativos, que contribuem para a usabilidade e compreensão do bloco por parte dos usuários, conforme sugerido por (GUZDIAL; KAY, 2010) no contexto de ambientes de programação visual educativa.

4.4.3 Inserção e Organização na Interface

Por fim, o bloco criado é inserido na lista de blocos disponíveis no BIPES, permitindo que seja utilizado pelos usuários durante a construção de seus programas. Nesta fase, também é determinada a organização do bloco, especificando:

- A **categoria principal** à qual o bloco pertence (por exemplo, "Display", "Som", "Movimentação").
- A **subseção ou grupo** dentro da categoria, facilitando a localização e promovendo uma organização didática da biblioteca de blocos.

Esta classificação é fundamental para que o ambiente mantenha sua proposta de acessibilidade e clareza, especialmente considerando seu uso em contextos educacionais, conforme destacado por (RESNICK et al., 2009) em relação à importância de interfaces amigáveis para o ensino de programação.

4.4.4 Importância da Criação de Blocos

O domínio do processo de criação de blocos não apenas viabilizou a implementação do jogo de maneira eficiente e adequada ao contexto do BIPES, como também contribuiu para a personalização do ambiente de programação, tornando-o mais alinhado às especificidades do projeto.

Além disso, a habilidade de expandir o conjunto de blocos disponíveis promove a autonomia dos desenvolvedores e educadores, permitindo a adaptação do ambiente para diferentes aplicações, projetos ou níveis de complexidade, fortalecendo a proposta construtivista e interdisciplinar que orienta o uso de plataformas como o BIPES.

4.4.5 Materiais de Apoio para o Desenvolvimento de Blocos

Criar novos blocos no BIPES pode ser um desafio, principalmente para quem está começando ou não tem muita familiaridade com programação visual. Para ajudar nesse processo, dois recursos são especialmente úteis: o tutorial oficial do BIPES(BIPES Project, 2025) e o conjunto de exemplos e ferramentas para desenvolvedores mantido pelo Blockly (BLOCKLY, 2025).

O tutorial do BIPES explica de forma clara e prática como criar blocos personalizados. Ele mostra desde os primeiros passos até a configuração dos arquivos principais, como o `block_definitions.js`, que define a estrutura e a aparência do bloco, e o `generator_stubs.js`, que cuida da geração do código correspondente em `MicroPython`. Além disso, orienta como integrar o bloco criado ao ambiente, escolher a categoria correta e garantir que ele funcione dentro da interface gráfica.

Já o site do Blockly oferece várias ferramentas e exemplos para ajudar no desenvolvimento de novos blocos. Um dos destaques é o *Blockly DevTools*, que permite criar a definição dos blocos de forma visual, gerando automaticamente os arquivos em `JSON` ou `JavaScript`. Isso facilita bastante a prototipação e evita muitos erros. Além disso, o site apresenta boas práticas para a criação de blocos, incluindo como tratar entradas, saídas e eventos.

Usar esses dois recursos juntos torna o processo de criar novos blocos bem mais simples e organizado. Eles ajudam não só na parte técnica, mas também no desenvolvimento de boas práticas, como deixar a interface mais clara, os blocos mais modulares e o código gerado mais consistente.

Por isso, esses materiais são essenciais para quem quer ampliar o potencial do BIPES, criando novos blocos para projetos específicos, seja na educação, na pesquisa ou em aplicações profissionais. Eles reforçam a ideia de que ambientes de programação visual podem democratizar o acesso à programação e à computação física.

5 . RESULTADOS

Uma questão inicial do BIPES a ser melhorada era o seu bloco para displays OLED, originalmente o bloco não se mostrava adequado pra o uso em jogos, pois a biblioteca utilizada estava desatualizada e nela não era possivel fazer as importações da maneira adequada. Para possibilitar uma maior variação de usos do display, foi decidido que seria necessário melhorar o bloco. Para cumprir tal objetivo, foi necessário buscar uma biblioteca que permitisse o uso de outras fontes de texto, além de também permitir plotar algo que não fosse um texto, por exemplo, uma Figura geométrica. Após analise, a biblioteca escolhida para testes foi a MicroPython-oled-1.13 juntamente da ESP8266.

O primeiro problema surgiu ao tentar instalar a biblioteca no microcontrolador, devido às características intrínsecas da ESP ocorreu um erro de alocação de memória, conforme mostrado na Figura 17, impossibilitando a instalação da biblioteca.

```
exec(open('./test.py').read(),globals())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 7, in <module>
MemoryError: memory allocation failed, allocating 72 bytes
>>>
```

Figura 17 – Erro de alocação de memória na ESP8266

Para contornar esse problema, foi necessário recorrer a outro microcontrolador, a ESP32. Essa escolha foi feita, pois a ESP32 se mostra superior à ESP8266 em diversos aspectos, como poder de processamento, memoria RAM e FLASH. Com este novo controlador, foi possível utilizar a biblioteca escolhida e assim deixar o texto mostrado pelo display mais atraente.

Para realizar a implementação dos jogos no BIPES, foi necessária a criação de novos blocos (Figura 18) que suprissem as demandas específicas do projeto. Inicialmente, poucos blocos foram adicionados, apenas complementando os já existentes.

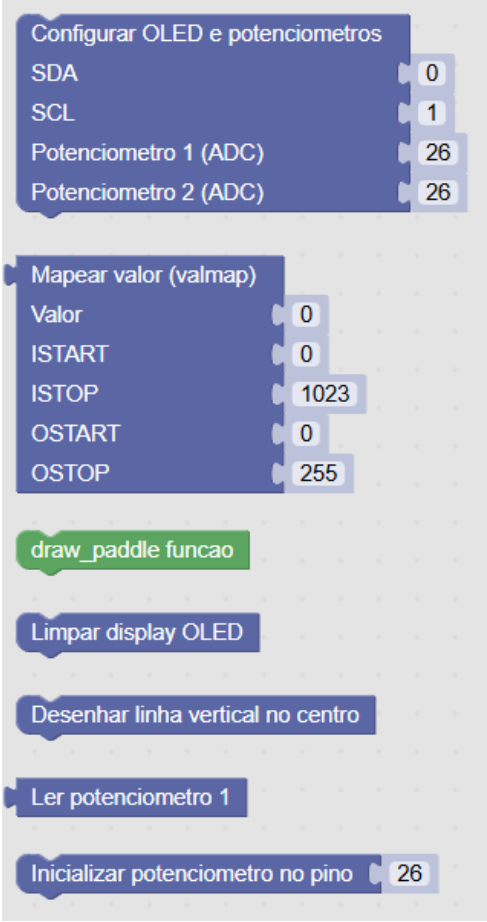


Figura 18 – Novos blocos criados para o desenvolvimento do Pong.

Contudo, percebeu-se que, mesmo com a programação ficando mais acessível em comparação ao uso direto de código, ainda poderia ser complexa demais para novos usuários, como é mostrado nas Figuras 19 e 20.

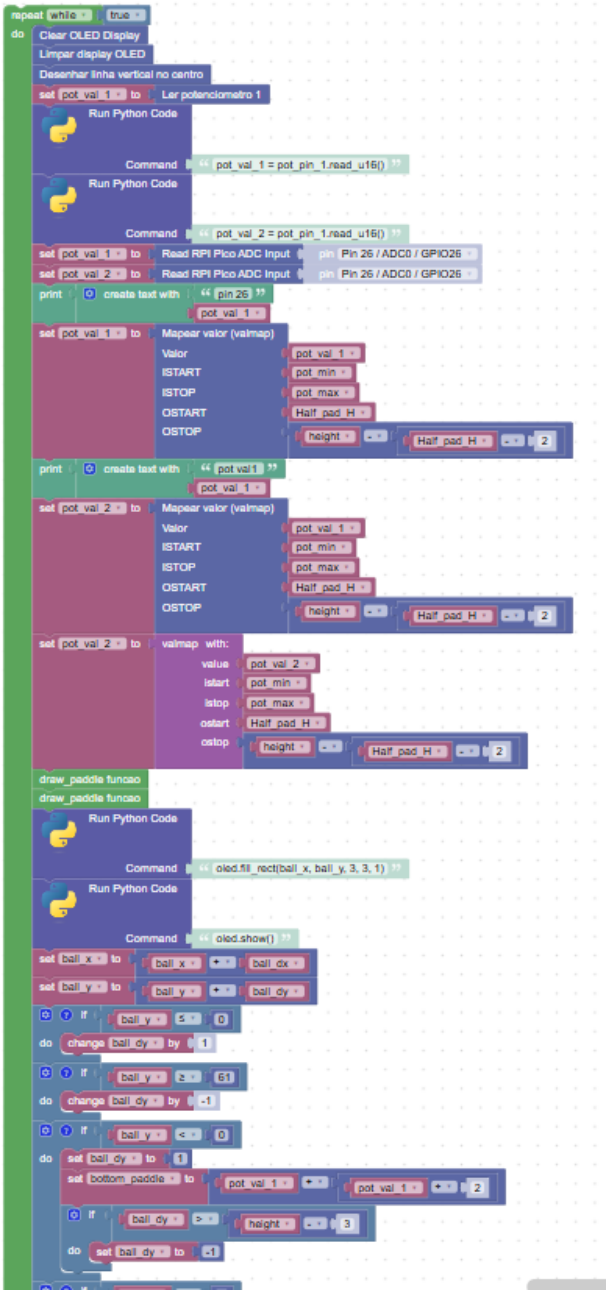


Figura 19 – Pong com os novos blocos pt 1.

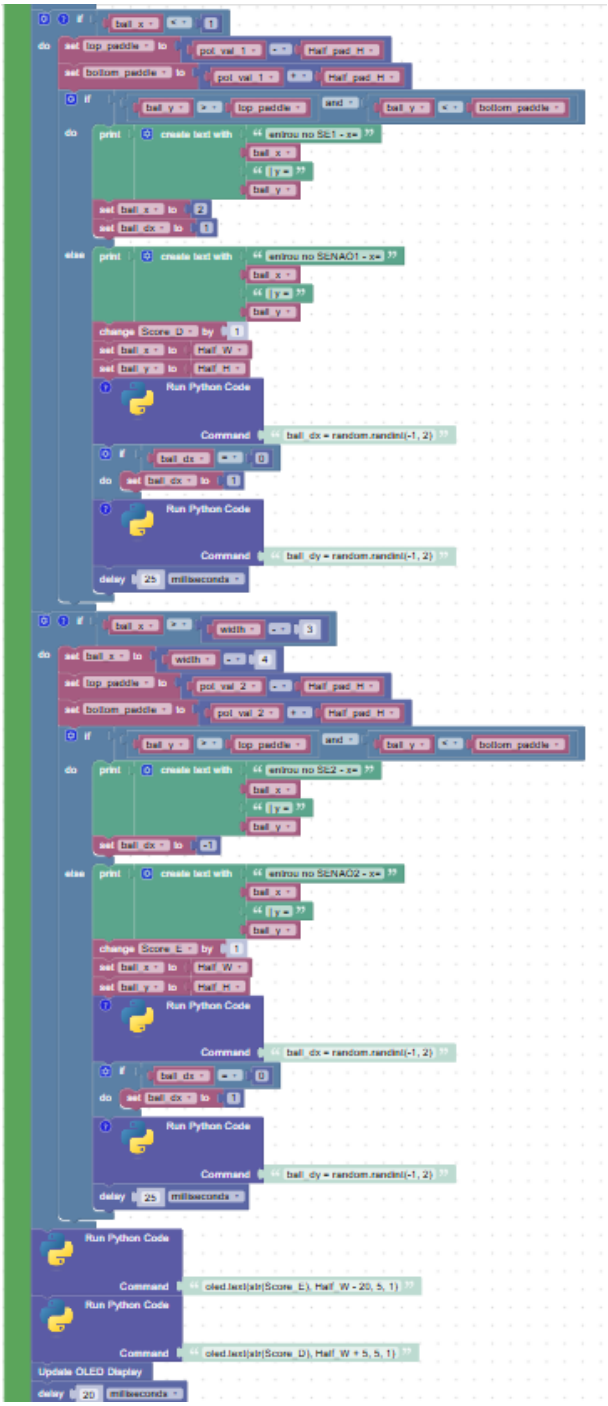


Figura 20 – Pong com os novos blocos pt 2.

Pensando nisso, foram desenvolvidos novos blocos mais abrangentes, que além de suprirem lacunas, também encapsulam e simplificam funções anteriormente dispersas. Para melhor organização, os blocos foram classificados em quatro categorias, seguindo a estrutura lógica do código original: **Entrada e Sensores**, **Som**, **Lógica do Jogo** e **Configuração**.

5.0.1 Entrada e Sensores

Esta categoria abrange blocos responsáveis pela leitura e mapeamento dos valores dos potenciômetros, utilizados como controle de entrada no jogo. Foram criados os seguintes blocos (Figura 21):

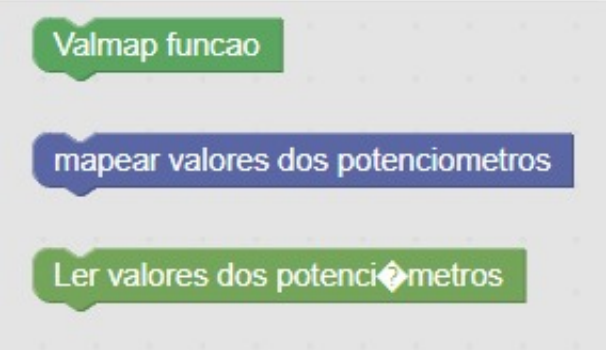


Figura 21 – Blocos de Entrada e Sensores.

- **Valmap função**: implementa a função `valmap` em MicroPython, que realiza o mapeamento linear de valores de entrada para o intervalo desejado.

```
def valmap(value, istart, istop, ostart, ostop):
  return int(ostart + (ostop - ostart) * ((value - istart) / (istop - istart))
```

- **Mapear valores dos potenciômetros**: bloco que encapsula a leitura dos valores brutos dos potenciômetros e realiza o mapeamento necessário para o controle das barras.

```
pot_val_1 = valmap(pot_val_1, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT - HALF
pot_val_2 = valmap(pot_val_2, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT - HALF
```

- **Ler valores dos potenciômetros**: bloco responsável por efetuar a leitura analógica dos potenciômetros conectados à placa.

```
pot_val_1 = pot_pin_1.read_u16()
pot_val_2 = pot_pin_2.read_u16()
```

5.0.2 Som

Esta categoria reúne blocos para reproduzir sons relacionados a eventos específicos no jogo, melhorando a experiência do usuário (Figura 22).

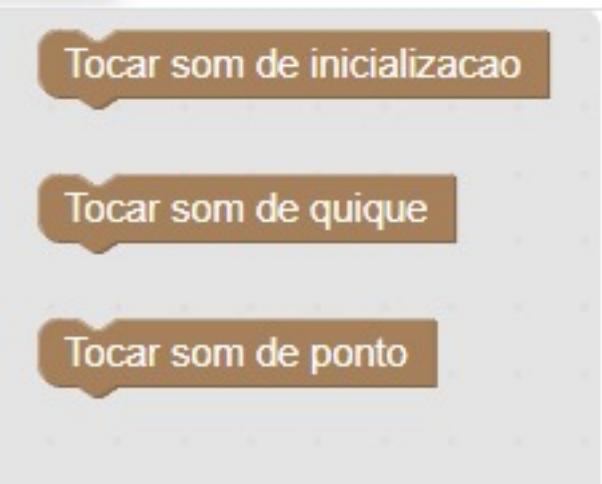


Figura 22 – Blocos relacionados ao som.

- **Tocar som de inicialização**: reproduz um som curto no momento em que o jogo é iniciado.

```
play_startup_sound()
```

- **Tocar som de quique**: reproduz um som sempre que a bola colide com as bordas ou com as barras.

```
play_bounce_sound()
```

- **Tocar som de ponto**: reproduz um som sempre que um jogador marca ponto.

```
play_score_sound()
```

5.0.3 Lógica do jogo

Nesta categoria estão os blocos que compõem a lógica fundamental do jogo, garantindo o funcionamento correto da mecânica (Figura 23).

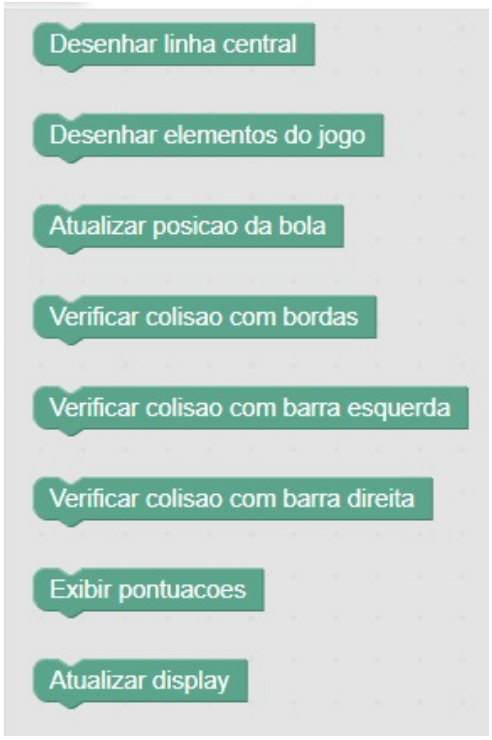


Figura 23 – Blocos da lógica do Pong.

- **Desenhar linha central**: responsável por desenhar a linha que divide o campo de jogo.

```
oled.vline(int(WIDTH / 2), 0, HEIGHT, 1)
```

- **Desenhar elementos do jogo**: desenha as barras dos jogadores e a bola.

```
draw_paddle(1, pot_val_1 + HALF_PAD_HEIGHT)
draw_paddle(2, pot_val_2 + HALF_PAD_HEIGHT)
draw_ball()
```

- **Atualizar posição da bola**: implementa o cálculo de movimento da bola com base nas direções e colisões.

```
ball_x = ball_x + ball_x_dir
ball_y = ball_y + ball_y_dir
```

- **Verificar colisão com bordas**: detecta se a bola colidiu com a borda superior ou inferior, invertendo sua direção.

```
if ball_y < 0:
    ball_y_dir = 1
if ball_y > HEIGHT - 3:
    ball_y_dir = -1
```

- **Verificar colisão com barra esquerda**: verifica colisão da bola com a barra do jogador 1.

```
if ball_x < 1:
    top_paddle = left_paddle - HALF_PAD_HEIGHT
    bottom_paddle = left_paddle + HALF_PAD_HEIGHT
if ball_y > top_paddle and ball_y < bottom_paddle:
    ball_x_dir = 1
    ball_x = 2
    print('paddle hit on left edge', pot_val_1, top_paddle, bottom_paddle)
else:
    r_score += 1
    ball_x = int(WIDTH / 2)
    ball_y = int(HEIGHT / 2)
    ball_x_dir = random.randint(-1, 2)
        if ball_x_dir == 0:
            ball_x_dir = 1
            ball_y_dir = random.randint(-1, 2)
            print('score on left edge', pot_val_1, top_paddle, bottom_paddle)
    sleep(.25)
```

- **Verificar colisão com barra direita**: verifica colisão da bola com a barra do jogador 2.

```
if ball_x > WIDTH - 3:
        ball_x = WIDTH - 4
        top_paddle = pot_val_2 - HALF_PAD_HEIGHT
        bottom_paddle = pot_val_2 + HALF_PAD_HEIGHT
        if ball_y > top_paddle and ball_y < bottom_paddle:
            ball_x_dir = -1
            print('bounce on right paddle', pot_val_1, top_paddle, bottom_paddle)
        else:
            l_score += 1
            play_score_sound()
            ball_x = int(WIDTH / 2)
            ball_y = int(HEIGHT / 2)
            ball_x_dir = random.randint(-1, 2)
            if ball_x_dir == 0:
                ball_x_dir = 1
            ball_y_dir = random.randint(-1, 2)
```

```
            print('score on right edge', pot_val_1, top_paddle, bottom_paddle)
            sleep(.25)
```

- **Exibir pontuações**: mostra as pontuações atualizadas no display.

```
oled.text(str(l_score), HALF_WIDTH - 20, 5, 1)
oled.text(str(r_score), HALF_WIDTH + 5, 5, 1)
```

- **Atualizar display**: efetua a atualização final do display após todas as alterações visuais.

```
oled.show()
```

5.0.4 Configuração

Os blocos desta categoria são responsáveis pela configuração inicial do hardware e das variáveis necessárias para o funcionamento do jogo (Figura 24).

Figura 24 – Blocos de configuração do Pong.

- **Inicializar display OLED**: realiza a configuração e inicialização do display OLED.

```
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl)
oled = SSD1306_I2C(128, 64, i2c)
```

- **Importar bibliotecas necessárias**: bloco que representa a importação dos módulos essenciais, como `machine` e `ssd1306`.

```
from machine import PWM, SPI
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C, SSD1306_SPI
from utime import sleep
import random # random direction for new ball
```

- **Inicializar potenciômetros**: configura as entradas analógicas utilizadas para ler os potenciômetros.

```
pot_pin_1 = machine.ADC(26)
pot_pin_2 = machine.ADC(26)
```

- **Inicializar alto-falante**: configura a saída PWM utilizada para gerar os sons.

```
SPEAKER_PIN = 16
speaker = PWM(Pin(SPEAKER_PIN))
```

- **Inicializar variáveis globais**: define e inicializa variáveis usadas em todo o jogo, como posições e velocidades.

```
WIDTH = 128
HALF_WIDTH = int(WIDTH / 2)
HEIGHT = 64
HALF_HEIGHT = HEIGHT
BALL_SIZE = 3
PAD_WIDTH = 2
PAD_HEIGHT = 16
HALF_PAD_WIDTH = int(PAD_WIDTH / 2)
HALF_PAD_HEIGHT = int(PAD_HEIGHT / 2)
POT_MIN = 3000
POT_MAX = 65534
MAX_ADC_VALUE = 65534
paddle1_vel = 0
paddle2_vel = 0
l_score = 0
r_score = 0
ball_x = int(WIDTH / 2)
```

```
ball_y = int(HEIGHT / 2)
ball_x_dir = 1
ball_y_dir = 1
```

5.0.5 Considerações

A criação dessas categorias e blocos permitiu uma estruturação mais didática e organizada da programação no ambiente BIPES. Com esses blocos, foi possível encapsular operações complexas em comandos simples e intuitivos, tornando o processo de desenvolvimento mais acessível, especialmente para iniciantes.

5.1 Implementação do Pong

Com a criação dos novos blocos no ambiente BIPES, foi possível implementar de forma eficiente o jogo **Pong**, clássico dos videogames, adaptado para a plataforma com a utilização da Raspberry Pi Pico e um display OLED.

O jogo consiste em duas barras, controladas por potenciômetros, e uma bola que se movimenta automaticamente pela tela. O objetivo de cada jogador é impedir que a bola ultrapasse sua barra, desviando-a de volta para o adversário. Cada vez que a bola passa por uma barra, o jogador adversário marca um ponto.

5.1.1 Controle

O controle das barras foi implementado utilizando dois potenciômetros conectados às entradas analógicas da Raspberry Pi Pico. Os valores analógicos são lidos e mapeados para as posições verticais das barras, permitindo um controle suave e preciso.

5.1.2 Mecânica do Pong

A bola inicia no centro da tela e se movimenta automaticamente, com direção e velocidade definidas por variáveis globais. As colisões com as bordas superiores e inferiores resultam na inversão do movimento vertical, enquanto as colisões com as barras causam a inversão do movimento horizontal. Caso a bola ultrapasse uma das barras, o adversário marca um ponto e a bola é reposicionada no centro.

5.1.3 Interface Gráfica

O display OLED é utilizado para exibir a interface gráfica do jogo. A tela apresenta:

- As duas barras dos jogadores.
- A bola em movimento.

- A linha central que divide o campo.
- A pontuação de cada jogador, atualizada em tempo real.

Todos os elementos são desenhados e atualizados a cada ciclo de execução, garantindo uma jogabilidade fluida.

5.1.4 Feedback Sonoro

Para enriquecer a experiência do jogo, foram adicionados sons utilizando um alto-falante controlado por sinal PWM. Diferentes sons são emitidos de acordo com eventos específicos:

- Som de inicialização: emitido quando o jogo é iniciado.
- Som de quique: emitido nas colisões da bola com bordas ou barras.
- Som de ponto: emitido quando um jogador marca ponto.

5.1.5 Estrutura do Código

O código do jogo foi estruturado de forma modular, utilizando os blocos desenvolvidos no BIPES para:

- Configuração inicial do hardware (display, potenciômetros e alto-falante).
- Leitura e mapeamento das entradas.
- Implementação da lógica de movimentação, colisão e pontuação.
- Atualização da interface gráfica e emissão de sons.

5.1.6 Código Final

A implementação do jogo no BIPES demonstrou a viabilidade de utilizar blocos visuais para o desenvolvimento de aplicações interativas com a Raspberry Pi Pico. Como mostrado na Figura 25, a utilização dos novos blocos simplificou significativamente o processo de programação, tornando-o mais acessível a usuários iniciantes, ao mesmo tempo em que manteve a possibilidade de criar jogos funcionais e completos.

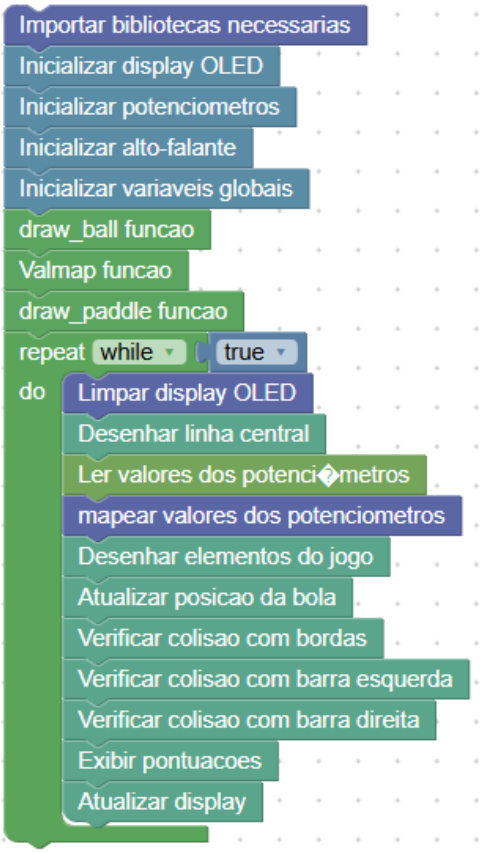


Figura 25 – Jogo pong feito em blocos do BIPES.

5.1.7 Montagem Final do Pong

Além da organização e simplificação do código, foi feita uma melhoria na montagem física do jogo. Imprimindo um case para colocar o circuito do jogo, tornando-o mais atraente e de facil manuseio. Na Figura 26 é possivel ver a montagem final e o jogo funcionando.

Figura 26 – Jogo Pong implementado no display OLED.

Por fim, cabe destacar que o uso do buzzer acarretou em um lag no jogo, por isso, optou-se por não usa-lo na montagem final

5.2 Implementação do Space Invader

Após a implementação do Pong, foi implementado de forma simplificada o jogo **Space Invader**, adaptado para a plataforma com a utilização da Raspberry Pi Pico e um display OLED.

O jogo consiste em uma nave controlada pelo jogador, que pode se mover horizontalmente na parte inferior da tela e disparar projéteis para eliminar os inimigos que descem em formação. O objetivo é destruir todos os inimigos antes que eles alcancem a parte inferior da tela ou colidam com a nave.

Na versão implementada, não foi feito o calculo de vidas do jogador, nem um aumento progressivo da dificuldade do jogo e as naves inimigas descem diretamente na vertical, ao contrario do jogo original em que a formação de inimigos posicionada na parte superior da tela, se move horizontalmente e, ao atingir a borda, desce um nível e inverte sua direção, aproximando-se da nave do jogador. Essas características podem ser incluídas futuramente.

5.2.1 Controle

O controle da nave foi implementado utilizando três botões conectados às entradas digitais da Raspberry Pi Pico. Dois botões são responsáveis pela movimentação lateral (esquerda e direita), enquanto o terceiro aciona o disparo. A leitura dos estados dos botões

é feita continuamente para atualizar a posição da nave e realizar os disparos conforme a interação do jogador.

5.2.2 Mecânica do Space Invader

A nave pode se mover lateralmente para esquivar-se e realizar disparos. Quando um projétil atinge um inimigo, ele é removido da tela. O jogo termina quando todos os inimigos são eliminados ou quando a formação alcança a posição da nave.

5.2.3 Interface Gráfica

O display OLED é utilizado para exibir a interface gráfica do jogo. A tela apresenta:

- A nave do jogador, representada por um sprite simples.
- Os projéteis disparados pela nave.
- A formação de inimigos, movimentando-se conforme a lógica do jogo.

Todos os elementos são desenhados e atualizados a cada ciclo de execução, garantindo uma jogabilidade fluida e responsiva.

5.2.4 Blocos Criados e Organizacao

Ao contrario do Pong, o **Space Invader** em blocos já foi desenvolvido com o foco em simplificar o máximo possível a sua programação, não passando pelo estagio de se utilizar blocos já existentes com a complementação de alguns poucos.

Desta forma, foram desenvolvidos novos blocos mais abrangentes, que além de suprirem lacunas, também encapsulam e simplificam funções anteriormente dispersas. Para melhor organização, os blocos foram classificados em quatro categorias, seguindo a estrutura lógica do código original: **Configuração**, **Funções**, **Execução**.

5.2.4.1 Configurações

Os blocos desta categoria são responsáveis pela configuração inicial do hardware e das variáveis necessárias para o funcionamento do jogo *Space Invader*, conforme ilustrado na Figura 27.

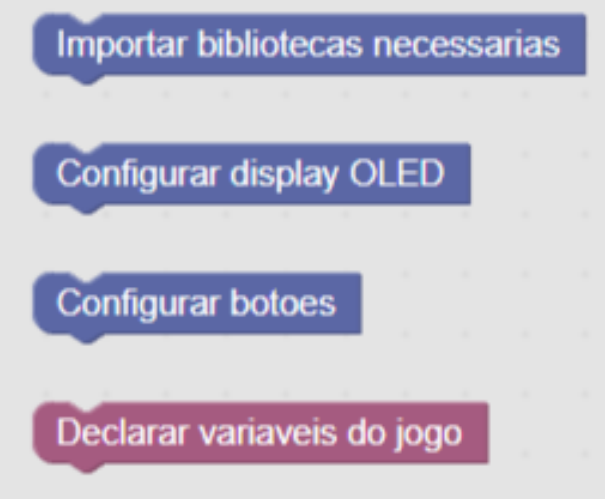


Figura 27 – Blocos de ConFiguração do Space Invader.

As principais configurações incluem:

- **Inicialização do display OLED**: configuração do barramento I2C, utilizando os pinos `GP0` e `GP1` da Raspberry Pi Pico para comunicação com o display, assegurando a correta exibição gráfica do jogo.

- **Configuração dos botões**: definição dos pinos digitais para a movimentação da nave e o disparo. Foram utilizados três botões: um para mover a nave para a esquerda, outro para a direita e um terceiro para realizar os disparos.

- **Importação das bibliotecas**: inclusão dos módulos necessários para o funcionamento do código, como `machine` para manipulação dos pinos, `ssd1306` para a interface gráfica, `utime` para controle de tempo e `random` para gerar comportamentos aleatórios dos inimigos.

- **Inicialização das variáveis**: definição das variáveis globais para armazenar as informações essenciais do jogo, como posições da nave, estado dos inimigos, projéteis ativos e demais parâmetros relacionados à lógica do jogo.

5.2.4.2 Funções

Esta categoria agrupa os blocos responsáveis pela implementação das principais funcionalidades do jogo, conforme mostrado na Figura 28.

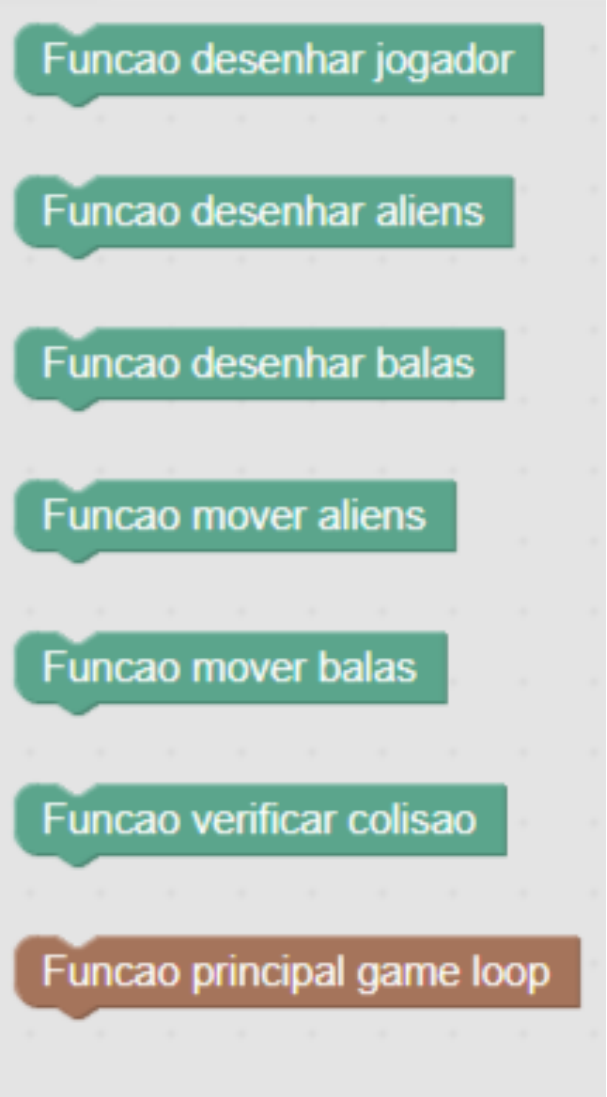


Figura 28 – Blocos de Funções do Space Invader.

As funções encapsulam comportamentos fundamentais e são chamadas durante a execução do jogo:

- **Leitura dos botões**: funções que realizam a leitura do estado dos botões de movimentação e disparo, determinando as ações do jogador.
- **Atualização da posição da nave**: ajuste da posição horizontal da nave conforme a leitura dos botões de movimentação, assegurando que ela permaneça dentro dos limites da tela.
- **Controle do disparo**: verificação do acionamento do botão de disparo e atualização da posição do projétil enquanto estiver ativo.
- **Movimentação dos inimigos**: função que implementa o padrão clássico de movimentação da formação inimiga, deslocando-se lateralmente e, ao atingir a borda, descendo uma linha e invertendo a direção.

- **Detecção de colisões**: função que verifica se há colisão entre projéteis e inimigos, removendo ambos da tela quando necessário e, opcionalmente, atualizando a pontuação do jogador.
- **Verificação do fim de jogo**: análise das condições de término da partida, seja pela eliminação completa dos inimigos (vitória) ou pela aproximação da formação inimiga até a posição da nave (derrota).

5.2.4.3 Execução

Esta categoria contém os blocos responsáveis pela execução contínua do jogo, garantindo a interação dinâmica entre o jogador e o ambiente, como ilustrado na Figura 29.

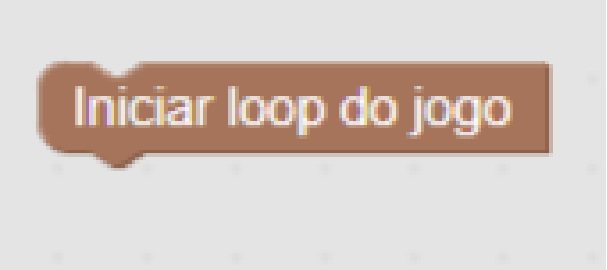


Figura 29 – Blocos de Execução do Space Invader.

Os blocos de execução estruturam o ciclo principal do jogo, organizando a sequência de chamadas às funções e a atualização da interface gráfica:

- **Chamada das funções de leitura e atualização**: a cada ciclo, são realizadas leituras dos botões e atualizações das posições da nave, projéteis e inimigos.
- **Renderização dos elementos gráficos**: após as atualizações, são desenhados na tela a nave, os inimigos remanescentes e os projéteis ativos.
- **Atualização do display**: efetiva todas as alterações gráficas no display OLED, assegurando que a interface reflita as mudanças ocorridas durante o ciclo de execução.
- **Controle de tempo**: utilização de pausas adequadas entre os ciclos, implementadas com `utime.sleep()`, para garantir uma velocidade de jogo adequada e fluidez na movimentação dos elementos.

Essa estrutura modular, organizada em **Configurações**, **Funções** e **Execução**, promove clareza e facilita o processo de desenvolvimento e compreensão do jogo, sendo especialmente útil para novos usuários que estão se familiarizando com o ambiente BIPES e com a lógica da programação de sistemas embarcados.

Assim, foi obtido o codigo mostrado na Figura 30

Figura 30 – Space Invader em Blocos.

Através deste condigo, foi possível obter o resultado mostrado na Figura 31 para o *Space Invader*:

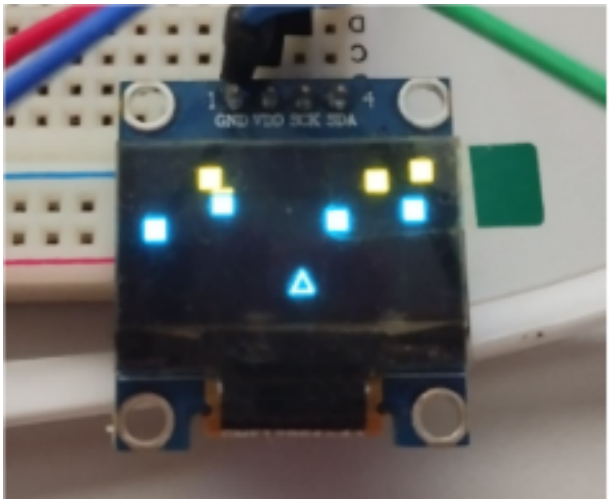


Figura 31 – Jogo Space Invader implementado no display OLED em Blocos.

Por fim, cabe destacar que, diferentemente do Pong, nesta implementação do *Space Invader* optou-se desde o início por não incluir elementos sonoros, visando simplificar o circuito e o desenvolvimento do jogo. Além disso, a estrutura da implementação foi

concebida desde o começo com foco na simplicidade e na acessibilidade, de modo a tornar o processo de programação mais intuitivo e adequado para novos usuários.

6 . CONCLUSÃO

Atrair estudantes para as áreas STEM é crucial para o avanço da sociedade e da inovação tecnológica. Conforme destacado no relatório *"Reasserting US Leadership in Microelectronics: The Role of Universities"*, o futuro da economia e da competitividade global depende do desenvolvimento de talentos nessas áreas, essenciais para impulsionar a inovação, superar desafios complexos e promover o progresso científico. Investir na formação e no engajamento de estudantes nessas disciplinas é fundamental para cultivar uma força de trabalho capacitada, preparada para enfrentar os desafios do mundo contemporâneo.

Em consonância com essa necessidade, a proposta deste trabalho visa não apenas atrair, mas também engajar estudantes do ensino fundamental e médio nas áreas STEM. Busca-se oferecer uma abordagem inovadora e prática para o aprendizado, através do desenvolvimento de blocos, ferramentas e recursos que permitam aos alunos criar seus próprios jogos físicos e interativos. A intenção é estimular o interesse e a participação ativa dos estudantes nas disciplinas STEM ao possibilitar que eles construam jogos semelhantes aos apresentados nas seções anteriores.

Neste contexto, destaca-se a importância do ambiente BIPES como uma ferramenta facilitadora para o ensino de computação física e sistemas embarcados. O BIPES, ao adotar uma abordagem de programação visual baseada em blocos, reduz significativamente a complexidade normalmente associada ao desenvolvimento de aplicações embarcadas. Essa característica torna a plataforma mais acessível para iniciantes, especialmente estudantes de níveis básico e médio, permitindo que explorem conceitos fundamentais de programação, lógica computacional e eletrônica sem a necessidade de dominar linguagens de programação tradicionais. Assim, o BIPES contribui para democratizar o acesso à educação tecnológica e fomentar o desenvolvimento de competências alinhadas às demandas da sociedade contemporânea.

A partir da criação de novos blocos para o ambiente BIPES, foi possível viabilizar a implementação do clássico jogo *Pong*, utilizando uma Raspberry Pi Pico e um display OLED. A modularização do código em blocos visuais facilitou a construção do jogo, permitindo encapsular funções complexas em comandos simples e intuitivos. Esse processo demonstrou que a programação em blocos pode ser uma estratégia eficaz não apenas para introduzir estudantes à lógica da programação, mas também para possibilitar a criação de projetos práticos e interativos, como jogos físicos. A implementação do Pong exemplifica como é possível transpor conceitos abstratos para uma aplicação concreta, promovendo um aprendizado mais significativo.

Além disso, o desenvolvimento deste trabalho resultou na elaboração de dois roteiros completos, um para o jogo *Pong* e outro para o *Space Invader*. Ambos os roteiros contemplam detalhadamente todas as etapas necessárias para a implementação dos jogos,

abrangendo desde a montagem física dos circuitos, com a conexão dos componentes eletrônicos, até a programação utilizando os blocos desenvolvidos no ambiente BIPES. Esses materiais foram organizados com o objetivo de proporcionar uma experiência didática e acessível, permitindo que estudantes e educadores possam replicar as atividades de forma autônoma, favorecendo a aprendizagem prática de sistemas embarcados e de conceitos fundamentais de programação.

Como parte das ações de divulgação e validação prática dos recursos desenvolvidos, foi realizada uma participação no **InovaSpace**, por meio da condução de um workshop com alunos do Ensino Fundamental II. Durante essa atividade, foram promovidos exercícios de programação utilizando o ambiente BIPES e as *CanSats*, permitindo que os participantes aplicassem na prática os conceitos trabalhados, explorando a programação por blocos e o funcionamento de sistemas embarcados. O workshop possibilitou ainda o compartilhamento de experiências, desafios e soluções relacionadas ao uso do BIPES em contextos educacionais, evidenciando seu potencial como ferramenta de apoio ao ensino de computação física e de estímulo ao interesse dos estudantes pelas áreas STEM. Além disso, com a experiencia do workshop, foi escrito um artigo sobre a experiencia e um guia para o BIPES, visando facilitar o primeiro contato com a ferramenta.

Diante dos resultados alcançados, considera-se que este trabalho cumpriu seus objetivos ao demonstrar a viabilidade e a importância de desenvolver novos blocos para o BIPES, com foco na criação de jogos físicos interativos. Embora não tenha sido realizada a aplicação pedagógica direta com estudantes, a base tecnológica e metodológica estabelecida poderá subsidiar futuras pesquisas e práticas educacionais que explorem o potencial do BIPES e da programação em blocos para o ensino de STEM. Assim, este trabalho representa uma contribuição relevante para o fortalecimento de estratégias que visam democratizar o acesso ao conhecimento tecnológico e estimular o protagonismo dos estudantes na construção de soluções criativas e inovadoras.

REFERÊNCIAS

AROCA, R. et al. **Uma Introdução à Internet das Coisas e Sistemas Embarcados utilizando Programação por Blocos com BIPES e ESP8266 / ESP32**. [s.n.], 2021. ISBN: 978-65-00-36921-2. Disponível em: <https://bipes.net.br/wp/book-livro/>.

AROCA, R. V. **Building MicroPython for the Raspberry Pi PICO with Ethernet and WebREPL Support**. 2022. Disponível em: <https://rafaelaroca.wordpress.com/2022/02/13/building-micropython-for-the-raspberry-pi-pico-with-ethernet-and-webrepl-support/>.

ARZTMANN, M. et al. Effects of games in stem education: a meta-analysis on the moderating role of student background characteristics. **Studies in Science Education**, v. 59, p. 1–37, 03 2022.

BARCELOS, T.; SILVEIRA, I. **Pensamento Computacional e Educação Matemática: Relações para o Ensino de Computação na Educação Básica**. [S.l.: s.n.], 2012.

BIPES Project. **Criando novos blocos no BIPES**. 2025. Acessado em: 26 maio 2025. Disponível em: <https://bipes.net.br/docs/get-started/create-block.html>.

BLOCKLY, G. **Blockly Developer Tools and Samples**. 2025. Acessado em: 26 maio 2025. Disponível em: <https://google.github.io/blockly-samples/examples/developer-tools/index.html>.

CHURSIN, G.; SEMENOV, M. Learning game development with unity3d engine and arduino microcontroller. **Journal of Physics: Conference Series**, v. 1488, p. 012023, 03 2020.

DAVIES, T. L. **Programming Embedded Systems: With C and MicroPython**. [S.l.]: O'Reilly Media, 2021. ISBN 978-1492080170.

DUCH, P.; JAWORSKI, T. Enriching computer science programming classes with arduino game development. p. 148–154, 07 2018.

FRASER, N. **Blockly: A visual programming editor**. 2013. <https://developers.google.com/blockly>. Acessado em: 26 maio 2025.

GEORGE, D. P. **MicroPython: Python for microcontrollers**. 2025. <https://micropython.org/>. Acessado em: 3 junho 2025.

GOOGLEDEVELOPERS. **Blockly Games**. 2019. <https://blockly.games>. Acessado em: 12 de fevereiro de 2024.

GUZDIAL, M.; KAY, A. Programming environments for novices. In: FINCHER, S.; PETRE, M. (Ed.). **Computer Science Education Research**. [S.l.]: Routledge, 2010. p. 127–154.

HAVA, K.; GüYER, T.; CAKIR, H. Gifted students' learning experiences in systematic game development process in after-school activities. **Educational Technology Research and Development**, v. 68, 02 2020.

JUNIOR, A. G. D. S. et al. Bipes: Block based integrated platform for embedded systems. **IEEE Access**, IEEE, v. 8, p. 197955–197968, 2020.

KONG, S.-c.; ABELSON, H. **Computational Thinking Education**. [S.l.: s.n.], 2019. ISBN 978-981-13-6527-0.

MARWEDEL, P. **Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things**. [S.l.: s.n.], 2021. ISBN 978-3-030-60909-2.

MATA, W.; TOBON, S. Analysis of factors associated to the enrollment and demand of computing-related careers. **Social Sciences**, v. 8, p. 1, 12 2018.

PAPERT, S. **Mindstorms: Children, Computers, and Powerful Ideas / Seymour Papert.** [S.l.: s.n.], 1980.

RASHID, M.; ANWAR, M.; KHAN, A. **Towards the Tools Selection in Model Based System Engineering for Embedded Systems - A Systematic Literature Review**. 2015.

RESNICK, M. et al. Scratch: Programming for all. **Communications of the ACM**, ACM, v. 52, n. 11, p. 60–67, 2009.

SILVA, V.; SILVA, L.; FRANçA, R. Pensamento computacional na formação de professores: experiências e desafios encontrados no ensino da computação em escolas públicas. p. 805, 10 2017.

STAUS, N. et al. Interested, disinterested, or neutral: Exploring stem interest profiles and pathways in a low-income urban community. **Eurasia Journal of Mathematics, Science and Technology Education**, v. 16, 04 2020.

STEVIESTL. **Raspberry Pi Pico running Pong**. 2021. <https://youtu.be/W6Yr9gv2dTQ?si=wn5i0Bs_YAsCMO56>. Acessado em: 12 de fevereiro de 2024.

WANG, N. et al. Gender differences in high school students' interest in stem careers: a multi-group comparison based on structural equation model. **International Journal of STEM Education**, v. 10, 10 2023.

WESTER, E. et al. Student engagement declines in stem undergraduates during covid-19-driven remote learning. **Journal of microbiology biology education**, v. 22, 03 2021.

XU, X.; JIN, W. Game development workshops designed and delivered by peer mentors to increase student curiosity and interest in an introductory programming course. p. 87–92, 04 2021.

ZHU, M.; WANG, A. Model-driven game development: A literature review. **ACM Computing Surveys**, v. 52, p. 1–32, 11 2019.

A Apêndice I - Roteiro Pong

Universidade Federal de São Carlos - UFSCar

Departamento de Ciência da Computação - DC

Programa de Pós-Graduação em Ciência da Computação - PPGCC

Ricardo Henrique da Silva Assis

Tutorial de montagem do jogo de pong com raspberry pi pico

São Carlos - SP

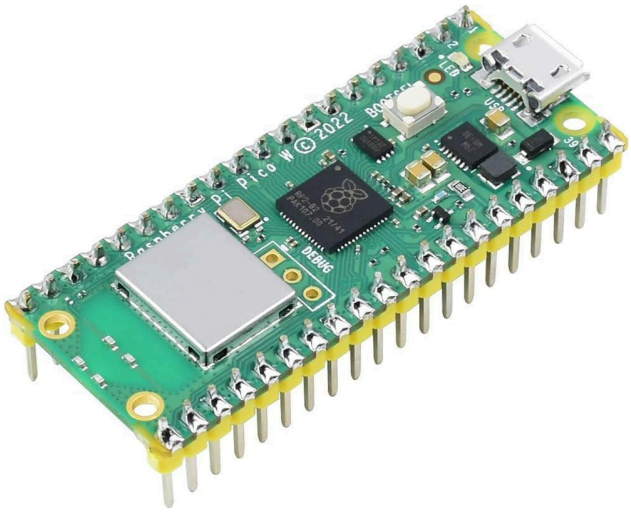
2025

ROTEIRO PONG

Neste roteiro, será mostrado passo-a-passo como realizar a montagem do jogo de Pong utilizando uma Raspberry Pi Pico e outros componentes simples. O código do jogo e as bibliotecas necessárias estão disponíveis no arquivo winrar.

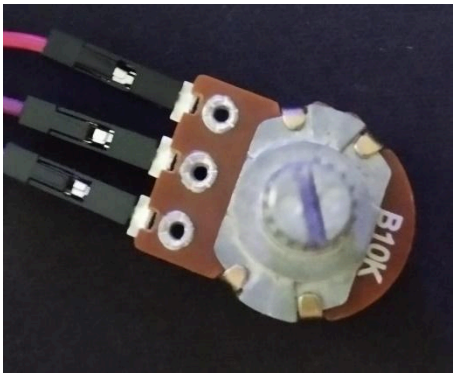
Materiais:

1. Raspberry Pi Pico

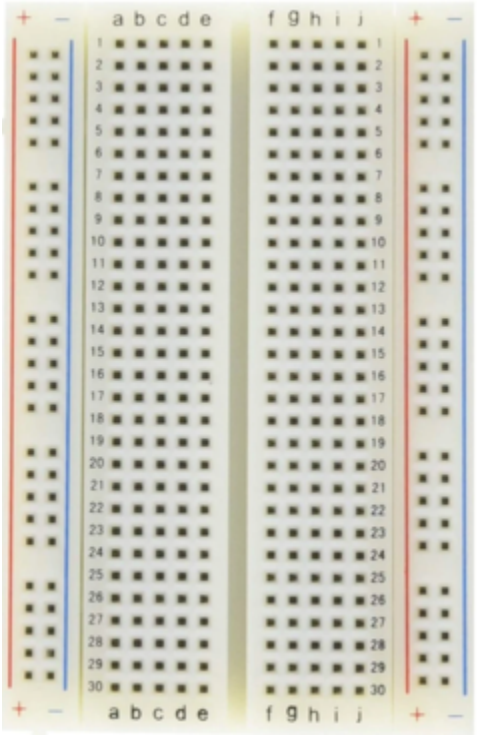


2. Display OLED

3. 2x Potenciômetros



4. Protoboard

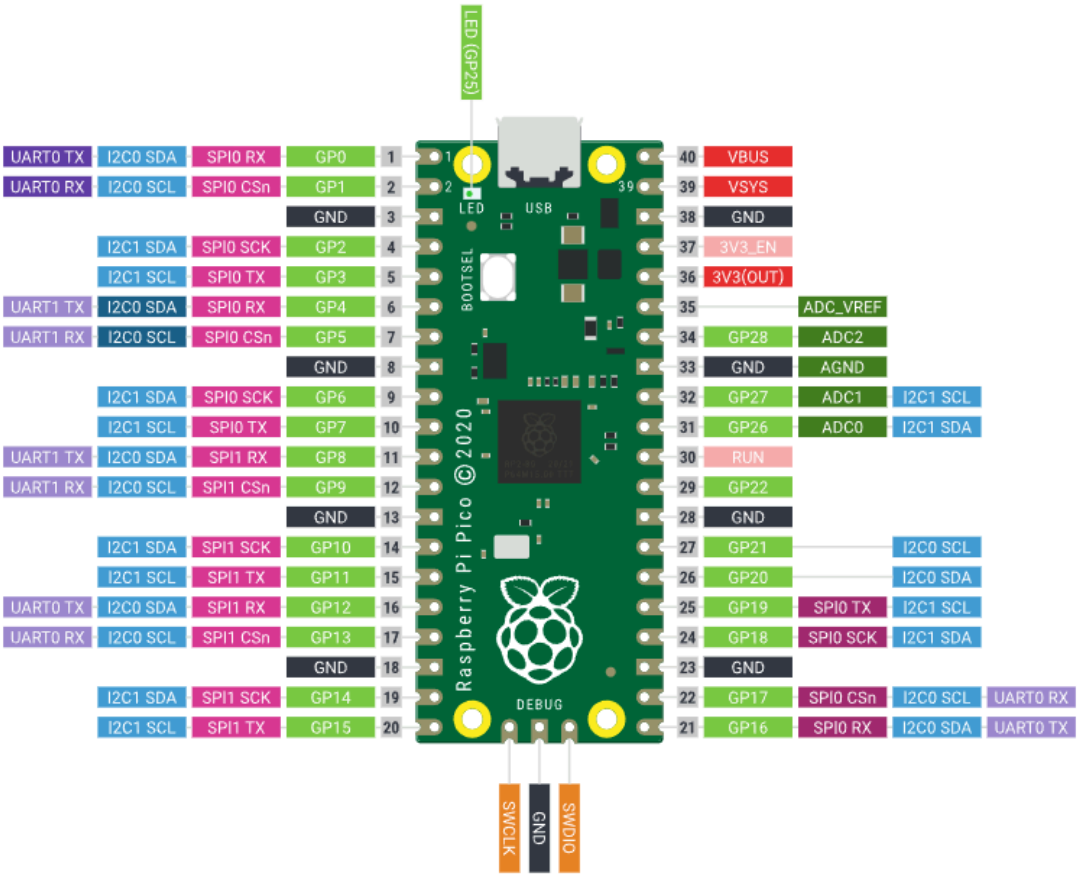


5. Buzzer

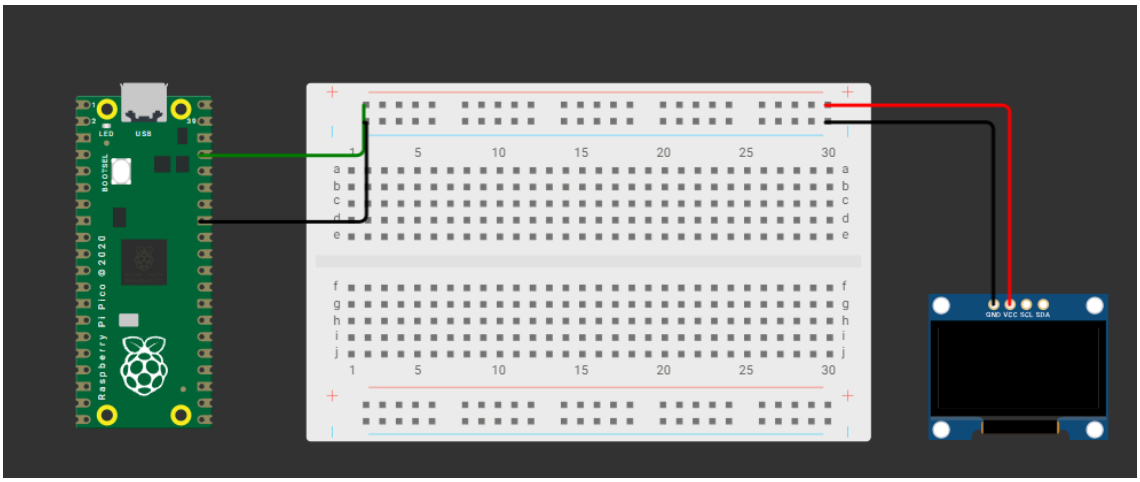
Montagem Física:

- Conecte a raspbarry pi pico e o display LED na protoboard. Obs.: conferir se está bem encaixado e que nenhuma trilha é compartilhada entre a raspbarry e o display

Para os próximos passos será necessário saber quais são os pinos da pi pico, para isso, basta usar a imagem a seguir:



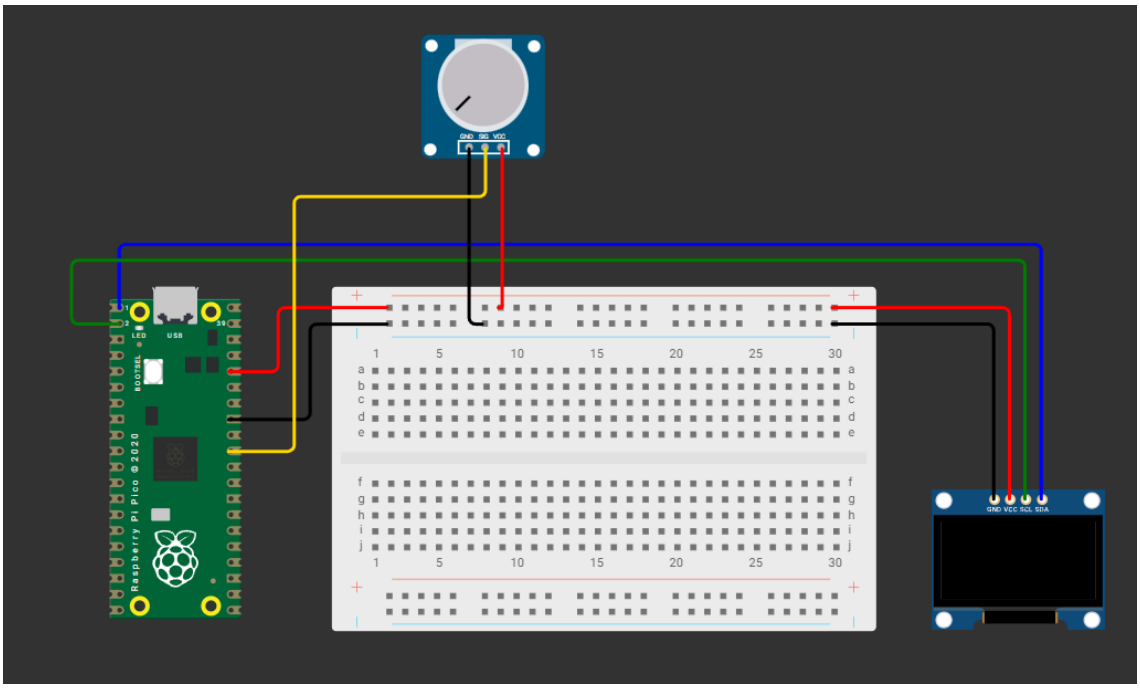
- Agora, conecte o GND e o 3V3(OUT) da placa nas entradas mais externas da protoboard, assim como é mostrado na figura a seguir:



- Conecte o GND e o VCC do OLED nas mesmas trilhas que você conectou o GND e o 3V3(OUT) da pi pico.

- Conecte o SDA do OLED no pino físico 1 (GP0) da placa e o SCL no pino 2(GP1).

- Da mesma forma que foi feita para o display OLED, conecte um dos pinos externos do potenciômetro no GND e o outro no 3V3(OUT). Já o pino do meio deve ser conectado ao ADC0 (GP26 - pino 31).



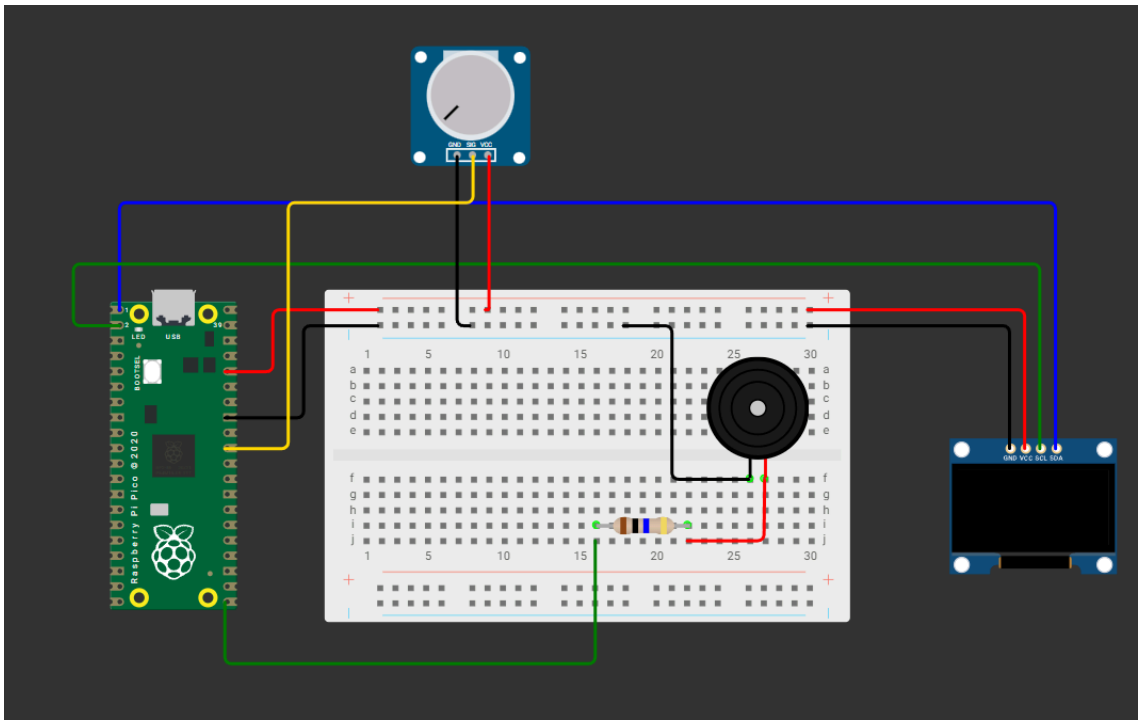
- Caso queira configurar o pong para dois jogadores, basta conectar outro potenciômetro da mesma maneira feita anteriormente, mas colocando no pino DC1 (GP27 - pino 32) da placa. Por fim, é necessário também alterar a seguinte linha do código do jogo:
    - 1 jogador

```
pot_pin_1 = machine.ADC(26)
pot_pin_2 = machine.ADC(26)
```

    - 2 jogadores

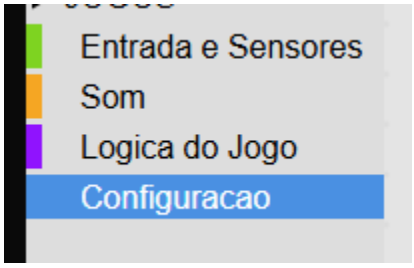
```
pot_pin_1 = machine.ADC(26)
pot_pin_2 = machine.ADC(27)
```

- Por fim, conecte o buzzer no circuito. Para isso, basta encaixá-lo na protoboard, no seu lado positivo deve-se colocar uma resistência e no outro lado da resistência, é necessário ligar no pino 21 - GP16. Já o lado negativo do buzzer deve ser conectado ao GND.



Montagem BIPES:

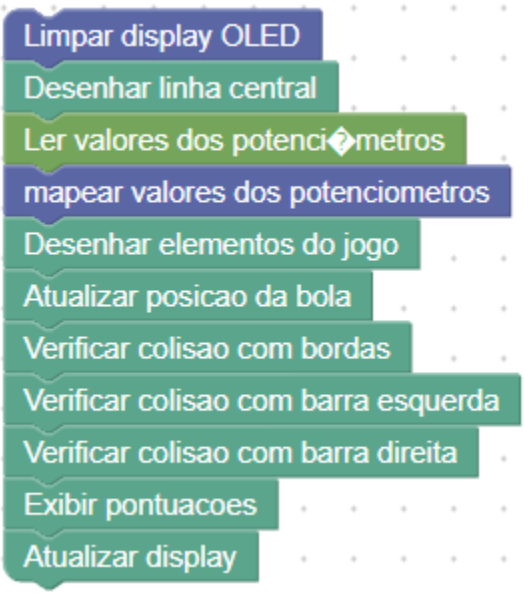
Os blocos necessários para se montar o jogo no BIPES estão localizados nas categorias mostradas a seguir:



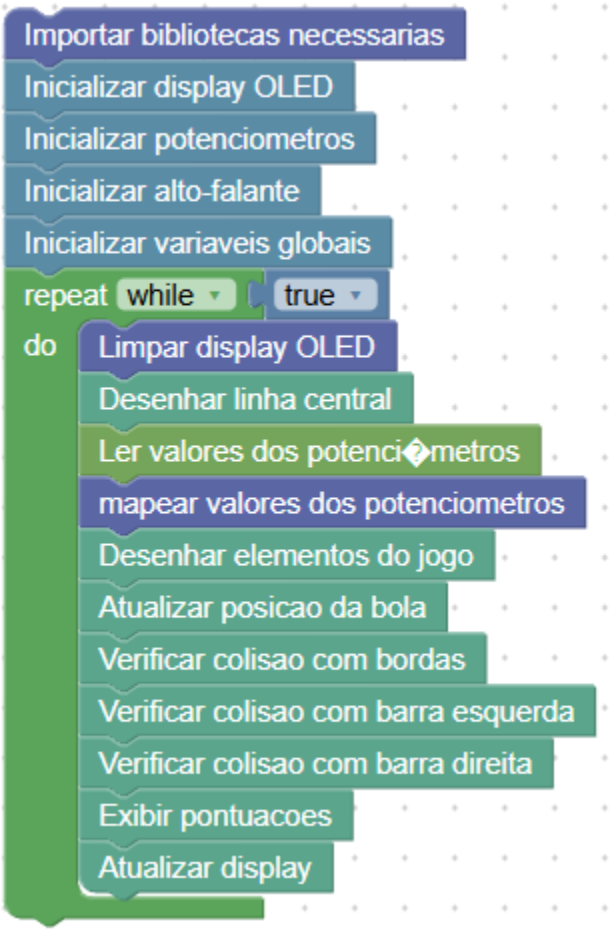
Primeiramente é preciso colocar os blocos referentes às configurações e declarações de funções do jogo, sendo que o primeiro deve ser o que faz a importação das bibliotecas

O segundo passo é criar um loop usando o bloco de repetição while e colocar como entrada um bloco lógico true:

Agora, você deve usar os blocos das categorias “Entradas e sensores” e “Lógica do jogo” e depois colocá-los dentro do bloco While usado anteriormente conforme a figura a seguir :



Após seguir esses passos, você deve ter um código em blocos igual ao que é mostrado a seguir:



Agora é só conectar a sua Raspberry Pi Pico, dar play e se divertir.

B Apêndice II - Roteiro Space Invader

Universidade Federal de São Carlos - UFSCar

Departamento de Ciência da Computação - DC

Programa de Pós-Graduação em Ciência da Computação - PPGCC

Ricardo Henrique da Silva Assis

Tutorial de montagem do jogo de Space Invader com raspberry pi pico

São Carlos - SP

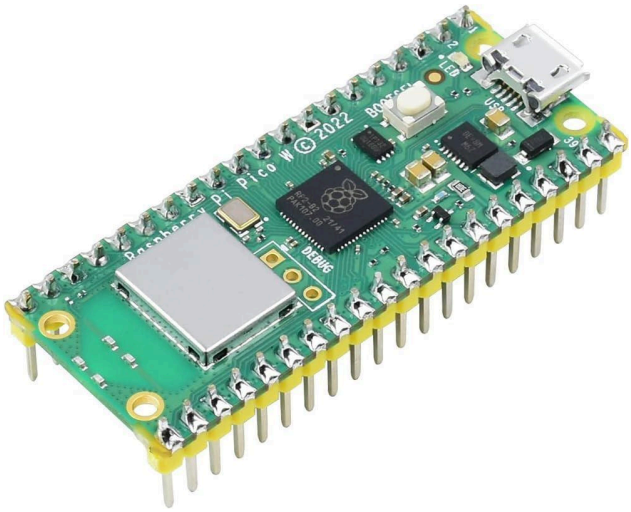
2025

ROTEIRO SPACE INVADER

Neste roteiro, será mostrado passo-a-passo como realizar a montagem do jogo space invader utilizando uma Raspberry Pi Pico e outros componentes simples. O código do jogo e as bibliotecas necessárias estão disponíveis no arquivo winrar.

Materiais:

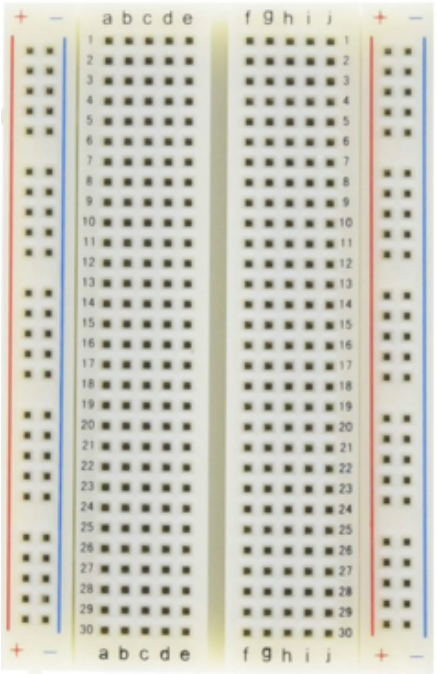
1. Raspberry Pi Pico



2. Display OLED

3. 3x botões

4. Protoboard

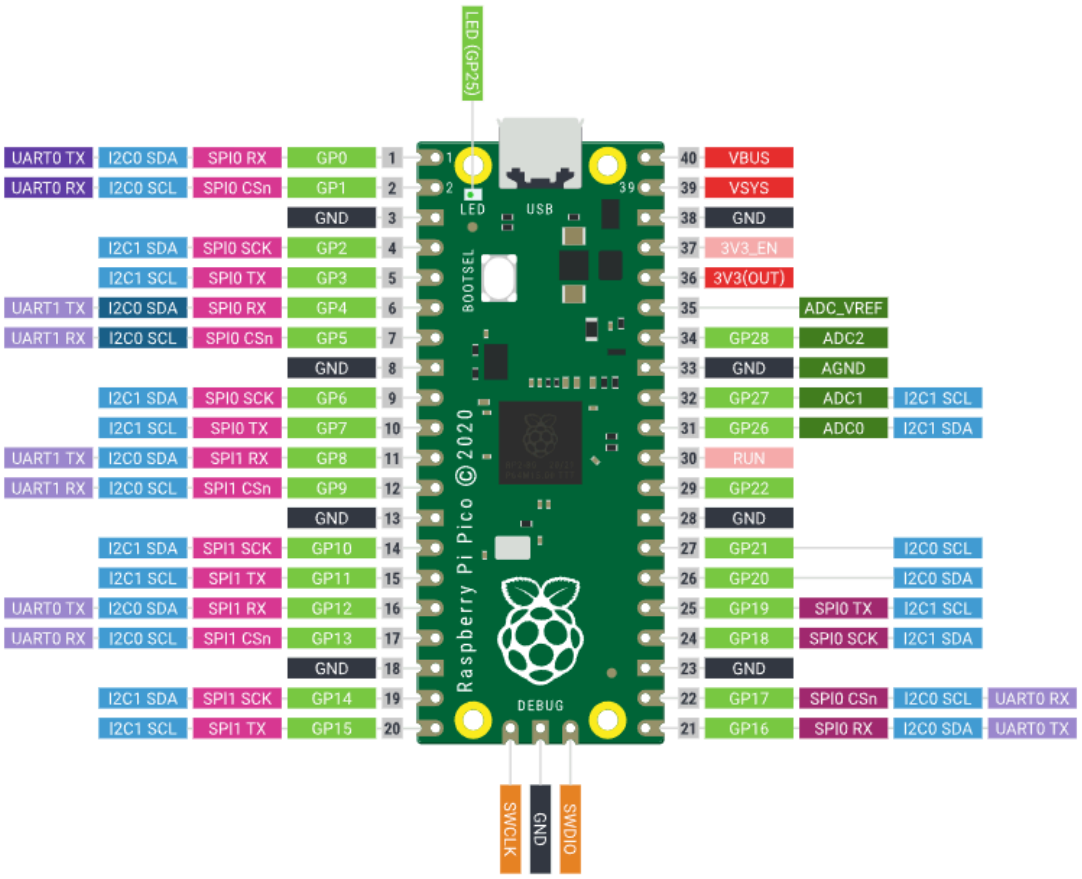


5. 3x Resistor 10k Ω

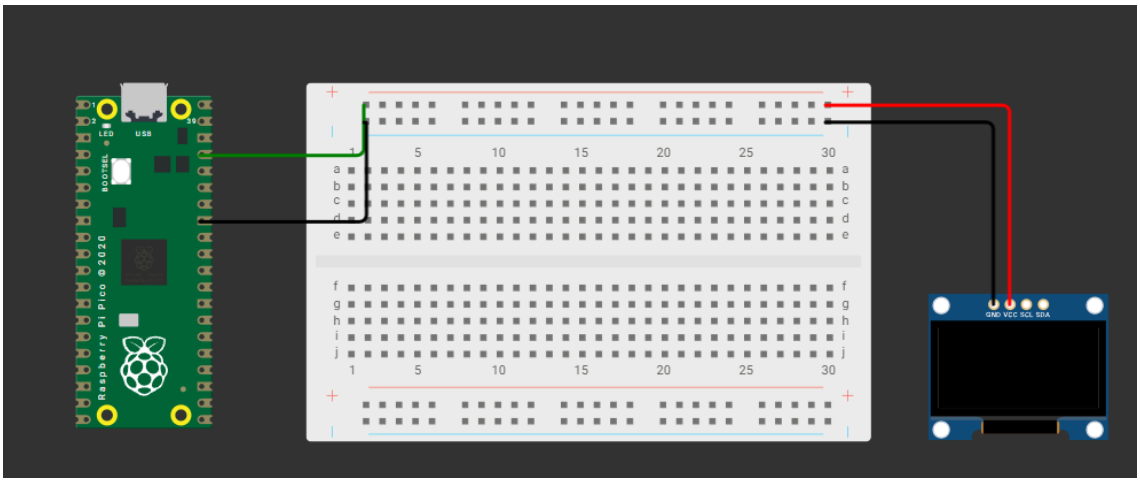
Montagem Física:

- Conecte a raspbarry pi pico e o display LED na protoboard. Obs.: conferir se está bem encaixado e que nenhuma trilha é compartilhada entre a raspbarry e o display

Para os próximos passos será necessário saber quais são os pinos da pi pico, para isso, basta usar a imagem a seguir:



- Agora, conecte o GND e o 3V3(OUT) da placa nas entradas mais externas da protoboard, assim como é mostrado na figura a seguir:



- Conecte o GND e o VCC do OLED nas mesmas trilhas que você conectou o GND e o 3V3(OUT) da pi pico.

- Conecte o SDA do OLED no pino físico 1 (GP0) da placa e o SCL no pino 2(GP1).

Da mesma forma que foi feita a conexão do display OLED, conecte os botões de controle diretamente aos pinos GPIO da Raspberry Pi Pico.

Conecte um dos terminais de cada botão ao GND e o outro terminal ao pino correspondente da Pico:

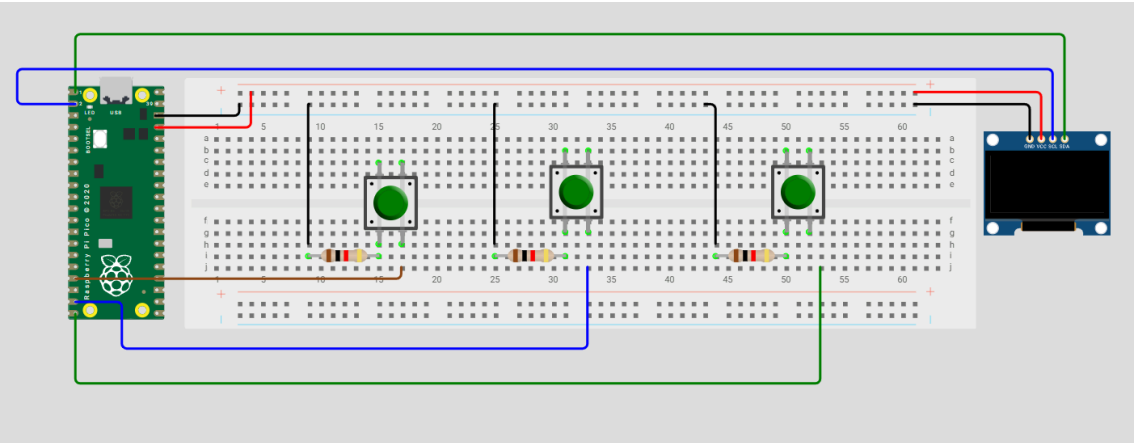
- GP14 (pino 19) para mover o jogador para a esquerda

- GP15 (pino 20) para mover o jogador para a direita

- GP13 (pino 17) para disparar

Esses pinos já estão configurados no código para utilizar resistores de pull-up internos, ou seja, não é obrigatório adicionar resistores externos. No entanto, recomenda-se adicionar resistores de 10kΩ como pull-down externos para garantir maior estabilidade na leitura do botão.

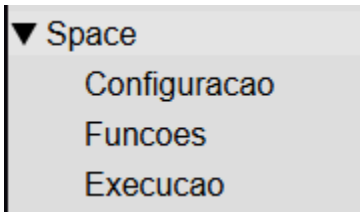
Os resistores devem ser conectados uma “perna” no GND e a outra em um dos terminais do botão, como na imagem a seguir:



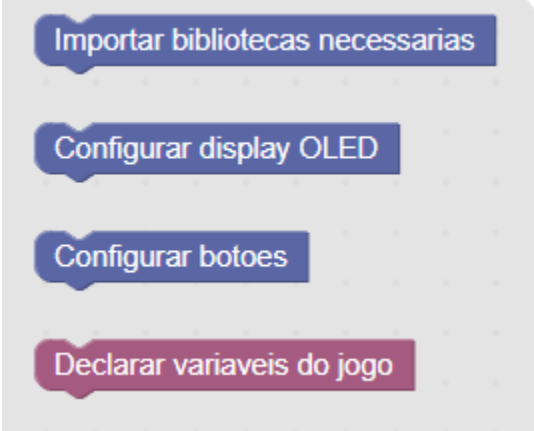
Após isso, o circuito está pronto para rodar o Space Invader

Montagem BIPES:

Os blocos necessários para se montar o jogo no BIPES estão localizados na categoria Space:

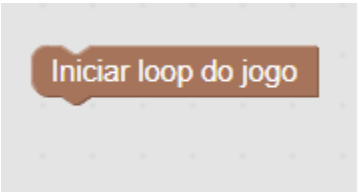


Primeiramente é preciso colocar os blocos referentes às configurações do jogo.

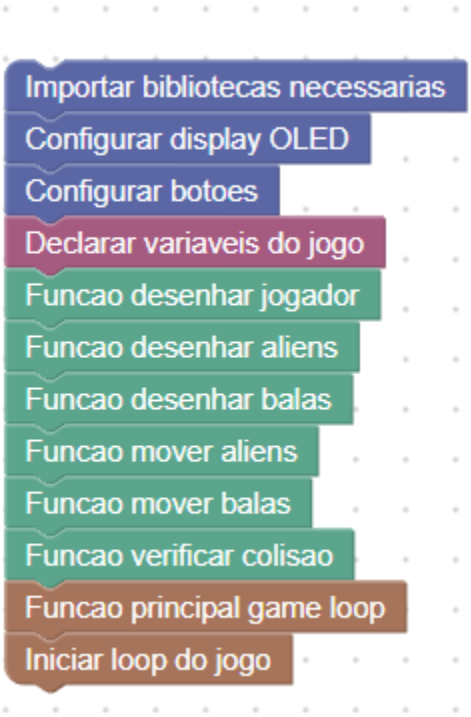


O segundo passo é declarar as funções usadas:

Por fim, basta chamar a função principal usando o bloco que se encontra na categoria execução:



Após seguir esses passos, você deve ter um código em blocos igual ao que é mostrado a seguir:



Agora é só conectar a sua Raspberry Pi Pico, dar play e se divertir.

C Apêndice III - Guia BIPES Para Ensino

Guia: BIPES — Blocos Integrados de Programação para Ensino de Sistemas

4 de junho de 2025

Sumário

1 Introdução ao BIPES

O **BIPES** (Blocos Integrados de Programação para Ensino de Sistemas) é uma plataforma de programação visual baseada em blocos, desenvolvida para facilitar o ensino e a aprendizagem de conceitos de eletrônica, robótica e programação de sistemas embarcados.

Utilizando uma interface gráfica intuitiva, o BIPES permite criar programas conectando blocos lógicos, reduzindo a barreira de entrada para quem está começando a programar.

Principais objetivos:

- Democratizar o ensino de programação e eletrônica.

- Facilitar o desenvolvimento de aplicações em plataformas como ESP32, Raspberry Pi Pico, entre outras.
- Permitir a experimentação direta com hardware.

2 Estrutura da Plataforma BIPES

2.1 Área de Blocos

Espaço central onde os usuários arrastam e soltam blocos para construir programas.

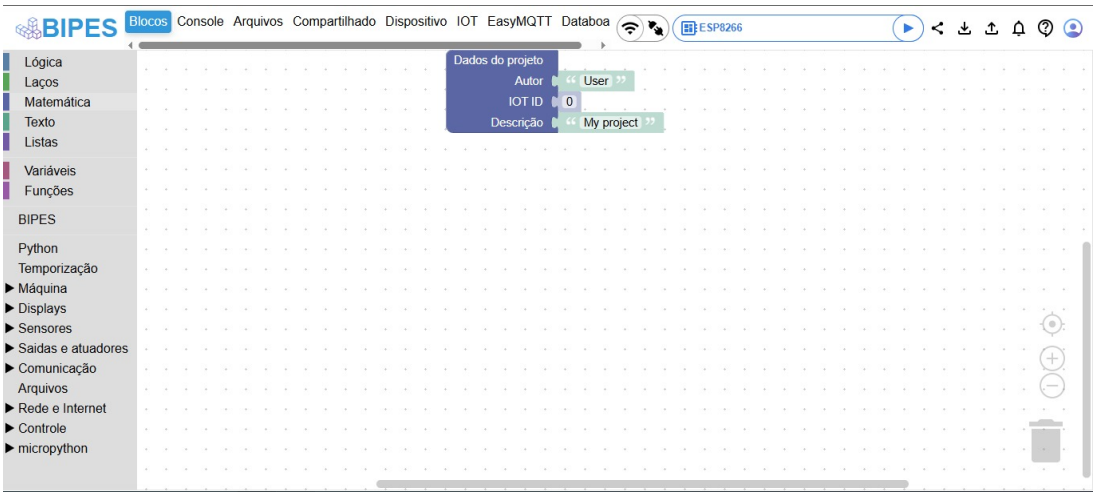


Figura 1: Area de trabalho do BIPES.

Função: substitui a codificação textual tradicional.

Exemplo:

Criar um programa que imprima “Olá CanSat e CubeSat” 10 vezes.

2.2 Menu de Blocos

Local onde se encontram todas as categorias de blocos.

Figura 2: Menu de categorias de blocos.

Funções e Categorias principais:

- Lógica: Blocos lógicos e estruturas condicionais.
- Laços: Blocos de repetição.
- Matemática: Operações aritméticas e lógicas.
- Texto: Blocos para imprimir textos.
- Variáveis: Blocos para criar variáveis para armazenar e manipular dados.

Além dessas categorias, também temos outras para projetos que usem componentes especificos, como displays, atuadores, sensores, entre outros.

2.3 Área de Código Gerado

Exibe automaticamente o código equivalente ao programa criado com blocos.

Figura 3: Área de Código Gerado

Exemplo:

Os blocos:

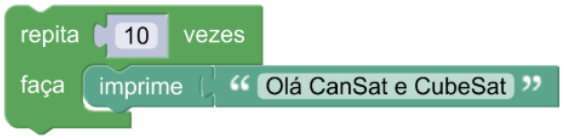


Figura 4: Código para criação de um laço simples

Geram o seguinte código em micropython:

Figura 5: Codificação do bloco de print

2.4 Área de Execução

Console de execução onde o usuário pode:

- Monitorar saídas.

- Testar o comportamento do programa e debugga-lo.

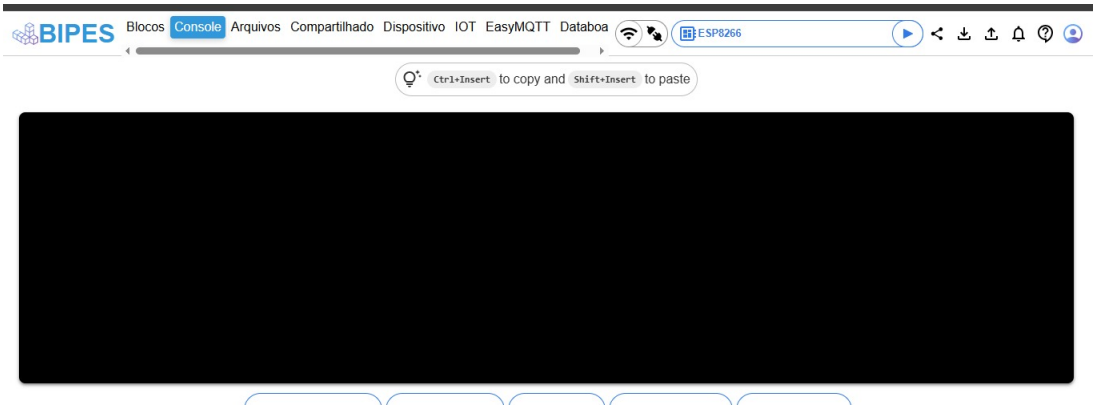


Figura 6: Terminal do BIPES

2.5 Conexão com Hardware

Interface para conectar a plataforma BIPES com o hardware.

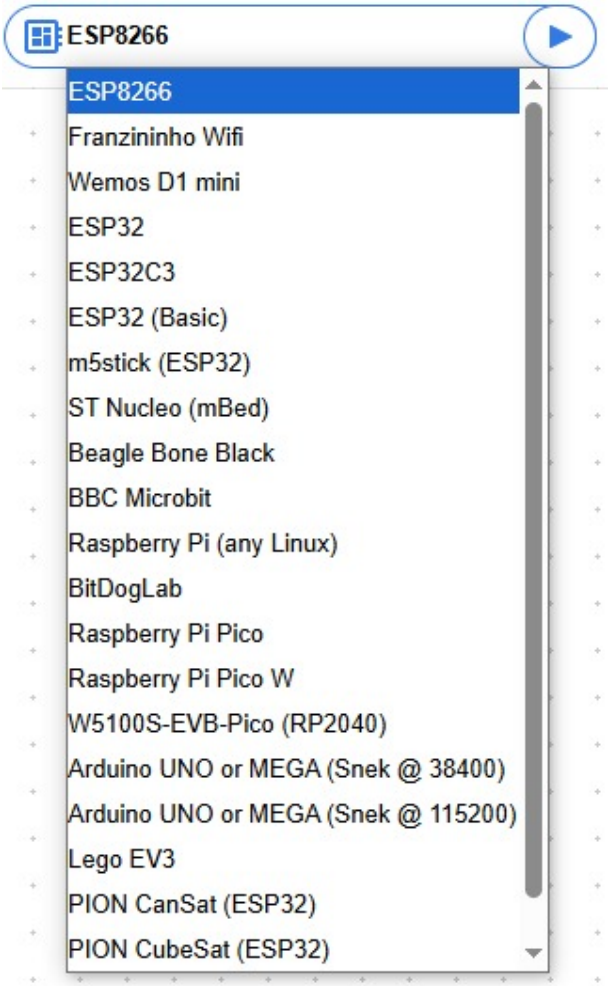


Figura 7: Conexões com microcontroladores

Passos:

1. Definir a placa.
2. Selecionar a porta serial.
3. Conectar.
4. Enviar o programa.

3 Area do Usuário

Clicando na foto no canto superior direito, abrirá a area do usuário. Nela você pode criar, baixar, apagar e navegar entre os seus projetos. Além disso, é nela que fica a configuração de linguagem.

3.1 Gerenciador de Projetos

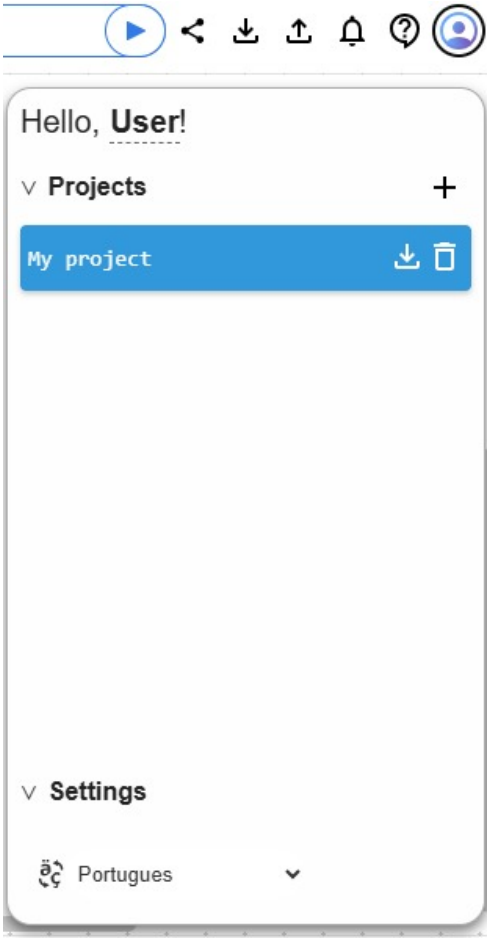


Figura 8: Gerenciador de projetos salvos.

4 Exemplos de Programas no BIPES

4.1 Exemplo 1: Laços de Repetição:

Cria um laço de repetição mais complexo, contendo condições de finalização.

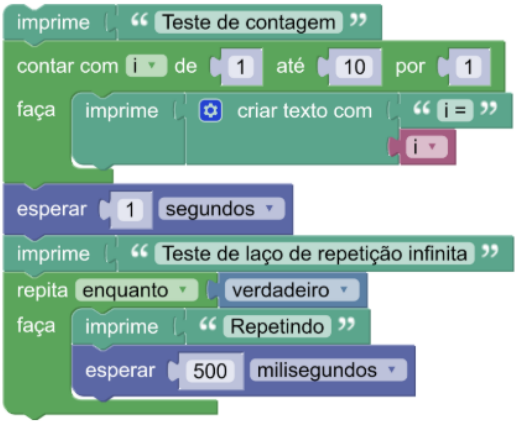


Figura 9: Código para criar um laço de repetição com condições.

4.2 Exemplo 2: Sensor de Temperatura e Umidade:

Mede a temperatura e umidade e imprime na aba console os valores lidos.

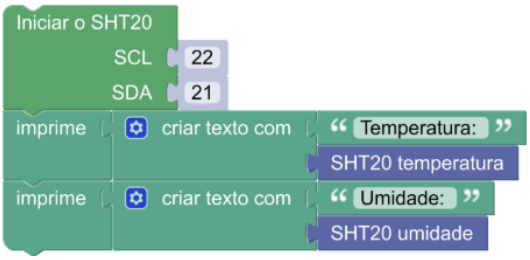


Figura 10: Código para medir a temperatura e umidade.

5 Exercícios Propostos

1. Crie um programa que pisque dois LEDs alternadamente.
2. Faça um sistema que leia um sensor de luz e acione um LED se o valor for abaixo de um limite.
3. Programe o BIPES para reproduzir uma melodia e, após terminar, exibir a mensagem “Fim da música”.
4. Desenvolva um programa que meça a temperatura a cada 10 segundos e envie o valor para o console por 1 minuto.
5. Monte um sistema de alarme com sensor de movimento: se detectar movimento, toque um som e acenda um LED.

6 Dicas e Boas Práticas no BIPES

- Teste blocos isoladamente antes de montar o programa completo.
- Observe o código gerado para aprender MicroPython.

- Use comentários para manter o projeto organizado.
- Salve frequentemente seus projetos.
- Explore diferentes sensores e atuadores.

7 Recursos Adicionais

- Site oficial do BIPES
- Documentação e tutoriais na própria plataforma.
- Comunidades e fóruns online.

D Apêndice IV - Generator_stubs

```
Blockly.Python['play_startup_sound'] = function(block) {
  return `play_startup_sound()\n`;
};

Blockly.Python['play_bounce_sound'] = function(block) {
  return `play_bounce_sound()\n`;
};

Blockly.Python['play_score_sound'] = function(block) {
  return `play_score_sound()\n`;
};


Blockly.Python['update_ball_position'] = function(block) {
  return `ball_x = ball_x + ball_x_dir\nball_y = ball_y + ball_y_dir\n`;
};

Blockly.Python['check_wall_collision'] = function(block) {
  return `if ball_y < 0:\n    ball_y_dir = 1\nif ball_y > HEIGHT - 3:\n    ball_y_dir = -1\n`;
};

Blockly.Python['check_left_paddle_collision'] = function(block) {
  return `if ball_x < 1:\n    top_paddle = left_paddle - HALF_PAD_HEIGHT\n    bottom_paddle
= left_paddle + HALF_PAD_HEIGHT\n    if ball_y > top_paddle and ball_y <
bottom_paddle:\n        ball_x_dir = 1\n        ball_x = 2\n        print('paddle hit on left edge',
pot_val_1, top_paddle, bottom_paddle)\n    else:\n        r_score += 1\n        ball_x =
int(WIDTH / 2)\n        ball_y = int(HEIGHT / 2)\n        ball_x_dir = random.randint(-1, 2)\n
if ball_x_dir == 0:\n            ball_x_dir = 1\n        ball_y_dir = random.randint(-1, 2)\n
print('score on left edge', pot_val_1, top_paddle, bottom_paddle)\n        sleep(.25)\n`;
};

Blockly.Python['check_right_paddle_collision'] = function(block) {
  return `if ball_x > WIDTH - 3:\n    ball_x = WIDTH - 4\n    top_paddle = right_paddle -
HALF_PAD_HEIGHT\n    bottom_paddle = right_paddle + HALF_PAD_HEIGHT\n    if ball_y
> top_paddle and ball_y < bottom_paddle:\n        ball_x_dir = -1\n        print('bounce on right
paddle', pot_val_1, top_paddle, bottom_paddle)\n    else:\n        l_score += 1\n        ball_x =
int(WIDTH / 2)\n        ball_y = int(HEIGHT / 2)\n        ball_x_dir = random.randint(-1, 2)\n
if ball_x_dir == 0:\n            ball_x_dir = 1\n        ball_y_dir = random.randint(-1, 2)\n
print('score on right edge', pot_val_1, top_paddle, bottom_paddle)\n        sleep(.25)\n`;
};

Blockly.Python['init_display'] = function(block) {
  return 'sda=machine.Pin(0)\nscl=machine.Pin(1)\ni2c=machine.I2C(0,sda=sda,
scl=scl)\noled = SSD1306_I2C(128, 64, i2c)\n';
};

Blockly.Python['init_pots'] = function(block) {
```

```
  return 'pot_pin_1 = machine.ADC(26)\npot_pin_2 = machine.ADC(26)\n';
};

Blockly.Python['init_speaker'] = function(block) {
  return 'SPEAKER_PIN = 16\nspeaker = PWM(Pin(SPEAKER_PIN))\n';
};

Blockly.Python['init_globals'] = function(block) {
  return `WIDTH = 128
HALF_WIDTH = int(WIDTH / 2)
HEIGHT = 64
HALF_HEIGHT = HEIGHT
BALL_SIZE = 3
PAD_WIDTH = 2
PAD_HEIGHT = 16
HALF_PAD_WIDTH = int(PAD_WIDTH / 2)
HALF_PAD_HEIGHT = int(PAD_HEIGHT / 2)
POT_MIN = 3000
POT_MAX = 65534
MAX_ADC_VALUE = 65534
paddle1_vel = 0
paddle2_vel = 0
l_score = 0
r_score = 0
ball_x = int(WIDTH / 2)
ball_y = int(HEIGHT / 2)
ball_x_dir = 1
ball_y_dir = 1\n`;
};

Blockly.Python['draw_center_line'] = function(block) {
  return 'oled.vline(int(WIDTH / 2), 0, HEIGHT, 1)\n';
};

Blockly.Python['read_pot_values'] = function(block) {
  return 'pot_val_1 = pot_pin_1.read_u16()\npot_val_2 = pot_pin_2.read_u16()\n';
};

Blockly.Python['display_scores'] = function(block) {
  return 'oled.text(str(l_score), HALF_WIDTH - 20, 5, 1)\noled.text(str(r_score),
HALF_WIDTH + 5, 5, 1)\n';
};

Blockly.Python['update_display'] = function(block) {
  return 'oled.show()\n';
};
Blockly.Python['valmap'] = function(block) {
  const code = `
```

```
def valmap(value, istart, istop, ostart, ostop):
    return int(ostart + (ostop - ostart) * ((value - istart) / (istop - istart)))\n`;
  return code;
};


Blockly.Python['map_pot_values'] = function(block) {
  var code = `
left_paddle = valmap(pot_val_1, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT -
HALF_PAD_HEIGHT - 2)
right_paddle = valmap(pot_val_1, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT -
HALF_PAD_HEIGHT - 2)\n`;
  return code;
};

Blockly.Python['init_imports'] = function(block) {
  return 'from machine import PWM, SPI\n' +
         'from machine import Pin, I2C\n' +
         'from ssd1306 import SSD1306_I2C, SSD1306_SPI\n' +
            'import random\n';
};

Blockly.Python['draw_elements'] = function(block) {
  return `draw_paddle(1, left_paddle)\ndraw_paddle(2, right_paddle)\ndraw_ball()\n`;
};

Blockly.Python['draw_paddle'] = function(block) {
  const code = `
def draw_paddle(paddle_no, paddle_center):
    if paddle_no == 1:
        x = 0
    else:
        x = WIDTH - 2
    y = paddle_center - HALF_PAD_HEIGHT
    oled.fill_rect(x, y, PAD_WIDTH, PAD_HEIGHT, 1)\n`;
  return code;
};

Blockly.Python['draw_ball'] = function(block) {
  const code = `
def draw_ball():
    oled.fill_rect(ball_x, ball_y, BALL_SIZE, BALL_SIZE, 1)\n`;
  return code;
};

// Space Invader -------------------------------------------------------------

Blockly.Python['update_display'] = function(block) {
```

```
  const code = `
oled.show()\n`;
  return code;
};

Blockly.Python['init_imports'] = function(block) {
  const code = `
import utime
from machine import PWM, SPI, Pin, I2C
from ssd1306 import SSD1306_I2C, SSD1306_SPI
import random\n`;
  return code;
};

Blockly.Python['config_display'] = function(block) {
  const code = `
sda = Pin(0)
scl = Pin(1)
i2c = I2C(0, sda=sda, scl=scl)
oled = SSD1306_I2C(128, 64, i2c)
oled_width = 128
oled_height = 64\n`;
  return code;
};

Blockly.Python['config_botoes'] = function(block) {
  const code = `
button_left = Pin(14, Pin.IN, Pin.PULL_UP)
button_right = Pin(15, Pin.IN, Pin.PULL_UP)
button_fire = Pin(13, Pin.IN, Pin.PULL_UP)\n`;
  return code;
};

Blockly.Python['variaveis_jogo'] = function(block) {
  const code = `
player_x = 64
player_y = 56
bullets = []
aliens = []
score = 0
aliens_destroyed = 0
MAX_ALIENS = 7\n`;
  return code;
};

Blockly.Python['funcao_draw_player'] = function(block) {
  const code = `
def draw_player(x, y):
```

```
    oled.line(x, y, x + 4, y - 8, 1)
    oled.line(x + 4, y - 8, x + 8, y, 1)
    oled.line(x, y, x + 8, y, 1)\n`;
  return code;
};

Blockly.Python['funcao_draw_aliens'] = function(block) {
  const code = `
def draw_aliens():
    for alien in aliens:
        oled.fill_rect(alien[0], alien[1], 8, 8, 1)\n`;
  return code;
};

Blockly.Python['funcao_draw_bullets'] = function(block) {
  const code = `
def draw_bullets():
    for bullet in bullets:
        oled.fill_rect(bullet[0], bullet[1], 2, 4, 1)\n`;
  return code;
};

Blockly.Python['funcao_move_aliens'] = function(block) {
  const code = `
def move_aliens():
    global aliens
    new_aliens = []
    for alien in aliens:
        if alien[1] < 64:
            new_aliens.append([alien[0], alien[1] + 1])
    aliens = new_aliens\n`;
  return code;
};

Blockly.Python['funcao_move_bullets'] = function(block) {
  const code = `
def move_bullets():
    global bullets
    new_bullets = []
    for bullet in bullets:
        if bullet[1] > 0:
            new_bullets.append([bullet[0], bullet[1] - 2])
    bullets = new_bullets\n`;
  return code;
};

Blockly.Python['funcao_check_collision'] = function(block) {
  const code = `
```

```
def check_collision():
    global bullets, aliens, score, aliens_destroyed
    new_bullets = []
    for bullet in bullets:
        hit = False
        for alien in aliens:
            if bullet[0] in range(alien[0], alien[0] + 8) and bullet[1] in range(alien[1], alien[1] + 8):
                score += 1
                aliens_destroyed += 1
                hit = True
                break
        if not hit:
            new_bullets.append(bullet)
    bullets = new_bullets\n`;
  return code;
};

Blockly.Python['funcao_game_loop'] = function(block) {
  var code =
'def game_loop():\n' +
'    global player_x, player_y, bullets, aliens\n' +
'    while True:\n' +
'        oled.fill(0)\n' +
'        draw_player(player_x, player_y)\n' +
'        draw_aliens()\n' +
'        draw_bullets()\n' +
'        if button_left.value() == 0 and player_x > 0:\n' +
'            player_x -= 2\n' +
'        if button_right.value() == 0 and player_x < oled_width - 8:\n' +
'            player_x += 2\n' +
'        if button_fire.value() == 0:\n' +
'            bullets.append([player_x + 3, player_y])\n' +
'        move_bullets()\n' +
'        move_aliens()\n' +
'        check_collision()\n' +
'        if len(aliens) < MAX_ALIENS:\n' +
'            if random.getrandbits(1) == 1:\n' +
'                aliens.append([random.randint(0, oled_width - 8), 0])\n' +
'        oled.show()\n' +
'        utime.sleep_ms(50)\n';
  return code;
};
Blockly.Python['chamar_game_loop'] = function(block) {
  const code = `
game_loop()\n`;
  return code;
};
```

E Apêndice V - Block_definitions

```
Blockly.Blocks['init_display'] = {
  init: function() {
    this.appendDummyInput().appendField("Inicializar display OLED");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(200);
    this.setTooltip("Inicializa o display OLED via I2C.");
  }
};

Blockly.Blocks['init_pots'] = {
  init: function() {
    this.appendDummyInput().appendField("Inicializar potenciometros");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(200);
    this.setTooltip("Inicializa os pinos ADC conectados aos potenciômetros.");
  }
};

Blockly.Blocks['init_speaker'] = {
  init: function() {
    this.appendDummyInput().appendField("Inicializar alto-falante");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(200);
    this.setTooltip("Inicializa o pino PWM do alto-falante.");
  }
};

Blockly.Blocks['init_globals'] = {
  init: function() {
    this.appendDummyInput().appendField("Inicializar variaveis globais");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(200);
    this.setTooltip("Define variaveis globais e constantes utilizadas no jogo.");
  }
};

Blockly.Blocks['play_startup_sound'] = {
  init: function() {
    this.appendDummyInput().appendField("Tocar som de inicializacao");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(30);
    this.setTooltip("Reproduz o som de inicializacao.");
  }
```

```
};

Blockly.Blocks['play_bounce_sound'] = {
  init: function() {
    this.appendDummyInput().appendField("Tocar som de quique");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(30);
    this.setTooltip("Reproduz o som quando a bola quica.");
  }
};

Blockly.Blocks['play_score_sound'] = {
  init: function() {
    this.appendDummyInput().appendField("Tocar som de ponto");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(30);
    this.setTooltip("Reproduz o som ao marcar um ponto.");
  }
};

Blockly.Blocks['draw_center_line'] = {
  init: function() {
    this.appendDummyInput().appendField("Desenhar linha central");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Desenha a linha central na tela.");
  }
};

Blockly.Blocks['read_pot_values'] = {
  init: function() {
    this.appendDummyInput().appendField("Ler valores dos potenciometros");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(100);
    this.setTooltip("Lê os valores dos potenciometros conectados.");
  }
};

Blockly.Blocks['draw_paddle'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("draw_paddle funcao");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
```

```
    this.setColour(125);
    this.setTooltip("Desenha as palhetas.");
    this.setHelpUrl("");
  }
};

Blockly.Blocks['draw_ball'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("draw_ball funcao");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(125);
    this.setTooltip("Desenha a bola.");
    this.setHelpUrl("");
  }
};

Blockly.Blocks['valmap'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Valmap funcao");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(125);
    this.setTooltip("Mapeia os valores lidos dos potenciometros para a faixa da tela.");
    this.setHelpUrl("");
  }
};

Blockly.Blocks['map_pot_values'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("mapear valores dos potenciometros");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Atualiza as variaveis dos potenciometros com valores mapeados.");
    this.setHelpUrl("");
  }
};

Blockly.Blocks['draw_elements'] = {
  init: function() {
    this.appendDummyInput().appendField("Desenhar elementos do jogo");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
```

```
    this.setTooltip("Desenha as barras e a bola na tela.");
  }
};

Blockly.Blocks['update_ball_position'] = {
  init: function() {
    this.appendDummyInput().appendField("Atualizar posicao da bola");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Atualiza a posição da bola com base na direcao atual.");
  }
};

Blockly.Blocks['check_wall_collision'] = {
  init: function() {
    this.appendDummyInput().appendField("Verificar colisao com bordas");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Inverte a direcao da bola se colidir com as bordas superior ou inferior.");
  }
};

Blockly.Blocks['check_left_paddle_collision'] = {
  init: function() {
    this.appendDummyInput().appendField("Verificar colisao com barra esquerda");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Verifica colisao com a barra esquerda e atualiza o jogo.");
  }
};

Blockly.Blocks['check_right_paddle_collision'] = {
  init: function() {
    this.appendDummyInput().appendField("Verificar colisao com barra direita");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Verifica colisao com a barra direita e atualiza o jogo.");
  }
};

Blockly.Blocks['display_scores'] = {
  init: function() {
    this.appendDummyInput().appendField("Exibir pontuacoes");
    this.setPreviousStatement(true, null);
```

```
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Mostra os placares na tela.");
  }
};

Blockly.Blocks['update_display'] = {
  init: function() {
    this.appendDummyInput().appendField("Atualizar display");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Atualiza o conteudo do display OLED.");
  }
};



Blockly.Blocks['init_imports'] = {
  init: function () {
    this.appendDummyInput()
        .appendField("Importar bibliotecas necessarias");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Importa bibliotecas para GPIO, I2C e display OLED.");
  }
};


// Space Invader ---------------------------------------------




Blockly.Blocks['update_display'] = {
  init: function() {
    this.appendDummyInput().appendField("Atualizar display");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Atualiza o conteúdo do display OLED.");
  }
};

Blockly.Blocks['init_imports'] = {
  init: function () {
```

```
    this.appendDummyInput()
        .appendField("Importar bibliotecas necessarias");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Importa bibliotecas para GPIO, I2C e display OLED.");
  }
};

Blockly.Blocks['config_display'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Configurar display OLED");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Configura o display OLED utilizando I2C.");
  }
};

Blockly.Blocks['config_botoes'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Configurar botoes");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Configura os botoes de controle do jogo.");
  }
};

Blockly.Blocks['variaveis_jogo'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Declarar variaveis do jogo");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(330);
    this.setTooltip("Declara as variaveis globais do jogo.");
  }
};

Blockly.Blocks['funcao_draw_player'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao desenhar jogador");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
```

```
    this.setColour(160);
    this.setTooltip("Define a funcao para desenhar o jogador.");
  }
};

Blockly.Blocks['funcao_draw_aliens'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao desenhar aliens");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Define a funcao para desenhar os aliens.");
  }
};

Blockly.Blocks['funcao_draw_bullets'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao desenhar balas");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Define a funcao para desenhar as balas.");
  }
};

Blockly.Blocks['funcao_move_aliens'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao mover aliens");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Define a funcao para mover os aliens.");
  }
};

Blockly.Blocks['funcao_move_bullets'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao mover balas");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Define a funcao para mover as balas.");
  }
};
```

```
Blockly.Blocks['funcao_check_collision'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao verificar colisao");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Define a funcao para verificar colisões e atualizar o score e a contagem de aliens destruidos.");
  }
};

Blockly.Blocks['funcao_game_loop'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Funcao principal game loop");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(20);
    this.setTooltip("Define a funcao principal do jogo com limite de aliens e contagem de destruidos.");
  }
};

Blockly.Blocks['chamar_game_loop'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Iniciar loop do jogo");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(20);
    this.setTooltip("Chama a funcao principal do jogo.");
  }
};
```

F Apêndice VI - Arquivo XML do BIPES

```
<category name="Entrada e Sensores" colour="#7ED321">
  <block type="valmap"></block>
  <block type="map_pot_values"></block>
  <block type="read_pot_values"></block>
</category>

  <category name="Som" colour="#F5A623">
    <block type="play_startup_sound"></block>
    <block type="play_bounce_sound"></block>
    <block type="play_score_sound"></block>
  </category>

  <category name="Logica do Jogo" colour="#9013FE">
    <block type="draw_center_line"></block>
    <block type="draw_elements"></block>
    <block type="draw_ball"></block>
    <block type="draw_paddle"></block>
    <block type="update_ball_position"></block>
    <block type="check_wall_collision"></block>
    <block type="check_left_paddle_collision"></block>
    <block type="check_right_paddle_collision"></block>
    <block type="display_scores"></block>
    <block type="update_display"></block>
    <block type="draw_player"></block>
    <block type="draw_aliens"></block>
    <block type="draw_bullets"></block>
    <block type="move_aliens"></block>
    <block type="move_bullets"></block>
    <block type="check_collision"></block>
    <block type="spawn_alien"></block>
    <block type="game_loop"></block>
  </category>

  <category name="Configuracao" colour="#4A90E2">
    <block type="init_imports"></block>
    <block type="init_display"></block>
    <block type="init_pots"></block>
    <block type="init_speaker"></block>
    <block type="init_display"></block>
    <block type="init_pots"></block>
    <block type="init_globals"></block>
</category>


// sssssssssssssssssssssssss


<category name="Space">
```

```
<category name="Configuracao">
  <block type="init_imports"></block>
  <block type="config_display"></block>
  <block type="config_botoes"></block>
  <block type="variaveis_jogo"></block>
</category>
<category name="Funcoes">
  <block type="funcao_draw_player"></block>
  <block type="funcao_draw_aliens"></block>
  <block type="funcao_draw_bullets"></block>
  <block type="funcao_move_aliens"></block>
  <block type="funcao_move_bullets"></block>
  <block type="funcao_check_collision"></block>
  <block type="funcao_game_loop"></block>
</category>
<category name="Execucao">
  <block type="update_display"></block>
  <block type="chamar_game_loop"></block>
</category>  </category>

<category name="Configuracao" colour="#4A90E2">
  <block type="init_imports"></block>
  <block type="init_display"></block>
  <block type="init_buttons"></block>
  <block type="init_globals"></block>
</category>

<category name="Logica do Jogo" colour="#9013FE">
  <block type="declare_draw_player"></block>
  <block type="call_draw_player"></block>

  <block type="declare_draw_aliens"></block>
  <block type="call_draw_aliens"></block>

  <block type="declare_draw_bullets"></block>
  <block type="call_draw_bullets"></block>

  <block type="declare_move_aliens"></block>
  <block type="call_move_aliens"></block>

  <block type="declare_move_bullets"></block>
  <block type="call_move_bullets"></block>

  <block type="declare_check_collision"></block>
  <block type="call_check_collision"></block>

  <block type="declare_game_loop"></block>
  <block type="call_game_loop"></block>
```

</category>

G Apêndice VII - Pong em MicroPython

```
# Pong game on Raspberry Pi Pico with a OLED and two Potentimeters
from machine import PWM, SPI
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C, SSD1306_SPI
from utime import sleep
import random # random direction for new ball

# spi_sck=machine.Pin(2)
# spi_tx=machine.Pin(3)
# spi=machine.SPI(0,baudrate=100000,sck=spi_sck, mosi=spi_tx)
# CS = machine.Pin(1)
# DC = machine.Pin(4)
# RES = machine.Pin(5)
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl)

oled = SSD1306_I2C(128, 64, i2c)
# oled = SSD1306_SPI(128, 64, spi, DC, RES, CS)
# connect the center tops of the potentiometers to ADC0 and ADC1
pot_pin_1 = machine.ADC(26)
pot_pin_2 = machine.ADC(26) # make them the same for testing

# lower right corner with USB connector on top
SPEAKER_PIN = 16
# create a Pulse Width Modulation Object on this pin
speaker = PWM(Pin(SPEAKER_PIN))



# globals variables
```

```
# static variables are constants are uppercase variable names
WIDTH = 128
HALF_WIDTH = int(WIDTH / 2)
HEIGHT = 64
HALF_HEIGHT = HEIGHT
BALL_SIZE = 3 # 2X2 pixels
PAD_WIDTH = 2
PAD_HEIGHT = 16
HALF_PAD_WIDTH = int(PAD_WIDTH / 2)
HALF_PAD_HEIGHT = int(PAD_HEIGHT / 2)
POT_MIN = 3000
POT_MAX = 65534
MAX_ADC_VALUE = 65534 # Maximum value from the Analog to Digital Converter is 2^16 -
1
# dynamic global variables use lowercase
paddle1_vel = 0
paddle2_vel = 0
l_score = 0
r_score = 0
# continiuous update of the paddle and ball

# start with the ball in the center
ball_x = int(WIDTH / 2)
ball_y = int(HEIGHT / 2)
# set the initial directinon to down to the right
ball_x_dir = 1
ball_y_dir = 1

def play_startup_sound():
    speaker.duty_u16(5000)
    speaker.freq(800)
```

```
    sleep(.25)
    speaker.freq(1200)
    sleep(.25)
    speaker.freq(1600)
    sleep(.25)
    speaker.duty_u16(0)

def play_bounce_sound():
    speaker.duty_u16(1000)
    speaker.freq(900)
    sleep(.25)
    speaker.duty_u16(0)

def play_score_sound():
    speaker.duty_u16(5000)
    speaker.freq(600)
    sleep(.25)
    speaker.freq(800)
    sleep(.25)
    speaker.duty_u16(0)

# note that OLEDs have problems with screen burn it - don't leave this on too long!
def border(WIDTH, HEIGHT):
    oled.rect(0, 0, WIDTH, HEIGHT, 1)



# Takes an input number vale and a range between high-and-low and returns it scaled to the new range
# This is similar to the Arduino map() function
def valmap(value, istart, istop, ostart, ostop):
```

```
  return int(ostart + (ostop - ostart) * ((value - istart) / (istop - istart)))


# draw a vertical bar
def draw_paddle(paddle_no, paddle_center):
    if paddle_no == 1:
        x = 0
    else:
        x = WIDTH - 2
    y = paddle_center - HALF_PAD_HEIGHT
    oled.fill_rect(x,  y, PAD_WIDTH, PAD_HEIGHT, 1) # fill with 1s


def draw_ball():
    oled.fill_rect(ball_x, ball_y, BALL_SIZE, BALL_SIZE, 1) # square balls for now


play_startup_sound()
# The main event loop
while True:
    oled.fill(0) # clear screen
    oled.vline(int(WIDTH / 2), 0, HEIGHT,  1)
    # border(WIDTH, HEIGHT)
    # read both the pot values
    pot_val_1 = pot_pin_1.read_u16()
    pot_val_2 = pot_pin_1.read_u16()
    # print(pot_val_1)


    # scale the values from the max value of the input is a 2^16 or 65536 to 0 to HEIGHT - PAD_HEIGHT
    # ideally, it should range from 5 to 58
    pot_val_1 = valmap(pot_val_1, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT - HALF_PAD_HEIGHT - 2)
    pot_val_2 = valmap(pot_val_2, POT_MIN, POT_MAX, HALF_PAD_HEIGHT, HEIGHT - HALF_PAD_HEIGHT - 2)
```

```
    # print(pot_val, pot_scaled)
    draw_paddle(1, pot_val_1 + HALF_PAD_HEIGHT)
    draw_paddle(2, pot_val_2 + HALF_PAD_HEIGHT)
    draw_ball()

    #update ball position with the current directions
    ball_x = ball_x + ball_x_dir
    ball_y = ball_y + ball_y_dir

    # update the ball direction if we are at the top or bottom edge
    if ball_y < 0:
        ball_y_dir = 1
#         play_bounce_sound()
    if ball_y > HEIGHT - 3:
        ball_y_dir = -1
#         play_bounce_sound()

    # if it hits the paddle bounce else score
    if ball_x < 1:
        top_paddle = pot_val_1 - HALF_PAD_HEIGHT
        bottom_paddle = pot_val_1 + HALF_PAD_HEIGHT
        if ball_y > top_paddle and ball_y < bottom_paddle:
            # we have a hit
            ball_x_dir = 1
            ball_x = 2
#             play_bounce_sound()
            print('paddle hit on left edge', pot_val_1, top_paddle, bottom_paddle)
        else:
            # we have a score for the right player
```

```
            play_score_sound()
            r_score += 1
            ball_x = int(WIDTH / 2)
            ball_y = int(HEIGHT / 2)
            ball_x_dir = random.randint(-1, 2)
            if ball_x_dir == 0:
                ball_x_dir = 1
            ball_y_dir = random.randint(-1, 2)
            print('score on left edge', pot_val_1, top_paddle, bottom_paddle)
            sleep(.25)

    if ball_x > WIDTH - 3:
        ball_x = WIDTH - 4
        top_paddle = pot_val_2 - HALF_PAD_HEIGHT
        bottom_paddle = pot_val_2 + HALF_PAD_HEIGHT
        if ball_y > top_paddle and ball_y < bottom_paddle:
            ball_x_dir = -1
            print('bounce on right paddle', pot_val_1, top_paddle, bottom_paddle)
        else:
            l_score += 1
            play_score_sound()
            ball_x = int(WIDTH / 2)
            ball_y = int(HEIGHT / 2)
            ball_x_dir = random.randint(-1, 2)
            if ball_x_dir == 0:
                ball_x_dir = 1
            ball_y_dir = random.randint(-1, 2)
#           play_bounce_sound()
            print('score on right edge', pot_val_1, top_paddle, bottom_paddle)
            sleep(.25)
```

```
oled.text(str(l_score), HALF_WIDTH - 20, 5, 1)

oled.text(str(r_score), HALF_WIDTH + 5, 5, 1)

oled.show()
```

H Apêndice VIII - Space Invader em MicroPython

```
import utime
from machine import PWM, SPI
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C, SSD1306_SPI
from utime import sleep
import random

# Configuração do display
# Pinos I2C na Raspberry Pi Pico
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl)

oled_width = 128
oled_height = 64

oled = SSD1306_I2C(128, 64, i2c)

# Configuração dos botões
button_left = Pin(14, Pin.IN, Pin.PULL_UP)
button_right = Pin(15, Pin.IN, Pin.PULL_UP)
button_fire = Pin(13, Pin.IN, Pin.PULL_UP)

# Variáveis do jogo
player_x = 64
player_y = 56
bullets = []
aliens = []
```

```
score = 0

# Função para desenhar o jogador
def draw_player(x, y):
    oled.line(x, y, x + 4, y - 8, 1)  # Diagonal esquerda
    oled.line(x + 4, y - 8, x + 8, y, 1)  # Diagonal direita
    oled.line(x, y, x + 8, y, 1)  # Base

# Função para desenhar os aliens
def draw_aliens():
    for alien in aliens:
        oled.fill_rect(alien[0], alien[1], 8, 8, 1)

# Função para desenhar as balas
def draw_bullets():
    for bullet in bullets:
        oled.fill_rect(bullet[0], bullet[1], 2, 4, 1)

# Função para mover os aliens
def move_aliens():
    global aliens
    new_aliens = []
    for alien in aliens:
        if alien[1] < 64:
            new_aliens.append([alien[0], alien[1] + 1])
    aliens = new_aliens
```

```
# Função para mover as balas
def move_bullets():
    global bullets
    new_bullets = []
    for bullet in bullets:
        if bullet[1] > 0:
            new_bullets.append([bullet[0], bullet[1] - 2])
    bullets = new_bullets

# Função para detectar colisão
def check_collision():
    global bullets, aliens, score
    new_bullets = []
    new_aliens = []
    for bullet in bullets:
        hit = False
        for alien in aliens:
            if bullet[0] in range(alien[0], alien[0] + 8) and bullet[1] in range(alien[1], alien[1] + 8):
                score += 1
                hit = True
                break
        if not hit:
            new_bullets.append(bullet)
    bullets = new_bullets

# Função principal do jogo
def game_loop():
    global player_x, player_y, bullets, aliens
```

```
    while True:
        oled.fill(0)
        draw_player(player_x, player_y)
        draw_aliens()
        draw_bullets()

        if button_left.value() == 0 and player_x > 0:
            player_x -= 2
        if button_right.value() == 0 and player_x < oled_width - 8:
            player_x += 2
        if button_fire.value() == 0:
            bullets.append([player_x + 3, player_y])

        move_bullets()
        move_aliens()
        check_collision()

        if random.getrandbits(1) == 1:
            aliens.append([random.randint(0, oled_width - 8), 0])

        #oled.text(f'Score: {score}', 0, 0)
        oled.show()
        utime.sleep_ms(50)

# Inicia o loop do jogo
game_loop()
```