UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
MESTRADO PROFISSIONAL EM MATEMÁTICA
EM REDE NACIONAL - PROFMAT

Gamificação na Educação Matemática:

Desenvolvimento de Jogos 2D no Construct 3 para Engajar Nativos Digitais

por

Pedro Paulo Cavalcante Pimentel

2025

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
MESTRADO PROFISSIONAL EM MATEMÁTICA
EM REDE NACIONAL - PROFMAT

Gamificação na Educação Matemática:
Desenvolvimento de Jogos 2D no Construct 3 para Engajar Nativos Digitais

por

Pedro Paulo Cavalcante Pimentel

sob a orientação do

Prof. Dr. José Laudelino de Menezes Neto

Dissertação apresentada ao Corpo Docente do Mestrado Profissional em Matemática em Rede Nacional - PROFMAT/CCEN/UFPB, como requisito parcial para a obtenção do título de Mestre em Matemática.

Fevereiro/ 2025
João Pessoa - PB

Catalogação na publicação
Seção de Catalogação e Classificação

P644g Pimentel, Pedro Paulo Cavalcante.
Gamificação na educação matemática : desenvolvimento de jogos 2D no Construct 3 para engajar nativos digitais / Pedro Paulo Cavalcante Pimentel. - João Pessoa, 2025.
137 f. : il.

Orientação: José Laudelino de Menezes Neto.
Dissertação (Mestrado) - UFPB/CCEN.

1. Jogos matemáticos. 2. Jogos didáticos - matemática. 3. Ensino da matemática - metodologia. 4. Plano cartesiano. 5. Funções afim. 6. Construct 3 - criação de jogos. I. Menezes Neto, José Laudelino de. II. Título.

UFPB/BC CDU 51-8(043)

Elaborado por RUSTON SAMMEVILLE ALEXANDRE MARQUES DA SILVA - CRB-15/0386

Gamificação na Educação Matemática:
Desenvolvimento de Jogos 2D no Construct 3 para Engajar Nativos Digitais

por

Pedro Paulo Cavalcante Pimentel

Dissertação apresentada ao Corpo Docente do Mestrado Profissional em Matemática em Rede Nacional - PROFMAT/CCEN/UFPB, como requisito parcial para a obtenção do título de Mestre em Matemática.

Área de Concentração: Matemática

Aprovada por:

Prof. Dr. José Laudelino de Menezes Neto - UFPB (Orientadora)

Prof. Dr. Adriano Alves De Medeiros - UFPB

Prof. Dr. Eudes Naziazeno Galvão - UFPE

Prof. Dr. Flank David Morais Bezerra - UFPB

Fevereiro/ 2025

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA EM REDE NACIONAL
Fone/Ramal: (83) 3216-7563 http://www.ufpb.br/pos/profmat

ATA DA SESSÃO PÚBLICA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DE MESTRADO PROFISSIONAL REALIZADA NO DEPARTAMENTO DE MATEMÁTICA DO CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA DA UNIVERSIDADE FEDERAL DA PARAÍBA

No dia vinte de fevereiro de dois mil e vinte e cinco (20/02/2025), às 16:00 horas, por meio da plataforma virtual *Google Meet*, através do link: **https://meet.google.com/msk-wmbe-mxv,** em conformidade com o parágrafo único do Art. 83 da Resolução CONSEPE nº 54/2024, em sessão pública, teve início a defesa de trabalho de conclusão de curso intitulado *"**Gamificação na Educação Matemática: Desenvolvimento de Jogos 2D no Construct 3 para Engajar Nativos Digitais**",* do aluno **PEDRO PAULO CAVALCANTE PIMENTEL**, que havia cumprido, anteriormente, todos os requisitos para a obtenção do grau de Mestre em Matemática, sob a orientação do professor José Laudelino de Menezes Neto. A Banca Examinadora, aprovada pelo Coordenador do Programa de Pós-Graduação em Matemática em Rede Nacional – PROFMAT, foi composta pelos professores José Laudelino de Menezes Neto (presidente), Adriano Alves de Medeiros (membro interno), Flank David Morais Bezerra (membro interno) e Eudes Naziazeno Galvão (membro externo/UFPE). O professor José Laudelino de Menezes Neto, em virtude da sua condição de presidente, iniciou os trabalhos e depois das formalidades de apresentação, convidou o aluno a discorrer sobre o conteúdo do seu trabalho de conclusão. Concluída a explanação, o candidato foi arguido pela Banca Examinadora, que em seguida, sem a presença do aluno, finalizando os trabalhos, reuniu-se para deliberar, tendo concedido a menção: **APROVADO**. Face à aprovação, declarou o presidente achar-se o avaliado legalmente habilitado a receber o Grau de **Mestre** em Matemática, cabendo à Universidade Federal da Paraíba, providências como, de direito, a expedição do Diploma a que o mesmo fez jus. Nada mais havendo a tratar, foi lavrada a presente ata que será assinada pelos membros da Banca Examinadora.

João Pessoa, 20 de fevereiro de 2025.

Banca Examinadora

José Laudelino de Menezes Neto _____
gov.br Documento assinado digitalmente
JOSE LAUDELINO DE MENEZES NETO
Data: 20/02/2025 17:04:06-0300
Verifique em https://validar.iti.gov.br

Adriano Alves de Medeiros _____
gov.br Documento assinado digitalmente
ADRIANO ALVES DE MEDEIROS
Data: 21/02/2025 10:33:24-0300
Verifique em https://validar.iti.gov.br

Flank David Morais Bezerra _____
gov.br Documento assinado digitalmente
FLANK DAVID MORAIS BEZERRA
Data: 21/02/2025 13:04:05-0300
Verifique em https://validar.iti.gov.br

Eudes Naziazeno Galvão _____
gov.br Documento assinado digitalmente
EUDES NAZIAZENO GALVAO
Data: 25/02/2025 06:18:32-0300
Verifique em https://validar.iti.gov.br

Agradecimentos

Ao longo desta jornada de mestrado, fui agraciado com o apoio de pessoas e instituições que, de diferentes formas, contribuíram para a realização deste trabalho e para o meu crescimento pessoal e acadêmico. A elas, dedico minha eterna gratidão.

Aos meus pais, em especial à minha mãe, cuja dedicação incansável e amor incondicional garantiram que nunca me faltasse nada, e ao meu pai, que sempre acreditou no meu potencial e me incentivou a buscar meus sonhos com coragem e perseverança. Foi graças à força que recebi de vocês que pude alcançar o que alcancei até hoje.

À minha companheira, Priscilla Trindade, que esteve ao meu lado em cada etapa dessa caminhada. Sou imensamente grato por sua paciência, compreensão e apoio constante, que foram fundamentais para que eu pudesse seguir em frente, mesmo nos momentos mais desafiadores.

Aos amigos que fiz na Escola Cônego Francisco Gomes de Lima, que estiveram sempre dispostos a oferecer sua ajuda e incentivo ao longo do meu percurso no Profmat, meu sincero agradecimento. Em especial, agradeço ao meu amigo Eduardo Bispo, professor de Língua Inglesa, por sua presença e apoio durante a aplicação de parte deste projeto, e à Greiciane Frazão, que sempre me incentivou.

Ao meu amigo de graduação João Paulo de Araujo Souza, por sua amizade, generosidade e por sempre compartilhar seus vastos conhecimentos de análise real, que foram fundamentais para meu amadurecimento acadêmico e para o sucesso deste curso.

Aos meus colegas do Profmat, com quem compartilhei tantos momentos de aprendizado e desafios, expresso minha gratidão. Em especial, agradeço ao meu amigo Marconi Ferreira (O Bispo), que sempre se mostrou disposto a ajudar nos problemas mais difíceis. Sua paciência e seu entusiasmo foram inspiradores e fizeram toda a diferença ao longo desse percurso.

Aos professores que, com sua sabedoria e dedicação, enriqueceram a minha formação. Destaco o meu orientador, o Professor Dr. José Laudelino de Menezes Neto, cuja orientação criteriosa, somada à sua disponibilidade em rigorosos encontros semanais, foi essencial para a realização deste trabalho. Sua paciência e rigor acadêmico foram fundamentais para que eu pudesse concretizar minhas ideias e superar os desafios que surgiram ao longo do processo.

Aos programas e instituições que, por meio de suas iniciativas, proporcionaram a oportunidade de me qualificar e me aperfeiçoar academicamente. Agradeço ao Programa

de Pós-Graduação Stricto Sensu para a Qualificação de Professores da Rede Pública de Educação Básica (PROEB), à Sociedade Brasileira de Matemática (SBM), ao Instituto de Matemática Pura e Aplicada (IMPA) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio e pela possibilidade de participar de um programa tão enriquecedor.

A todos que de alguma forma contribuíram para que eu chegasse até aqui, deixo meu mais sincero agradecimento. Sem o apoio de cada um de vocês, este trabalho não teria sido possível.

Dedicatória

Aos amigos que, unidos por um sonho comum, fundaram o "Cambridge Vest". Uma ideia efêmera, mas que se consolidou como o ponto de partida para momentos que mudaram minha vida para sempre.

Resumo

Este trabalho investiga o uso da plataforma Construct 3 como ferramenta para o desenvolvimento de jogos didáticos voltados ao ensino de Matemática, com ênfase na exploração do plano cartesiano e de funções afim. A proposta visa a criação de ambientes interativos e lúdicos que possibilitem ao estudante exercitar conceitos fundamentais de Geometria Analítica e Álgebra, tais como localização de pontos, translação, escalas e inclinação de retas, estabelecendo conexões práticas com a teoria. A dissertação descreve, inicialmente, a construção de um jogo simplificado chamado Carteiro Aventureiro, no qual o jogador insere coordenadas para deslocar um objeto no plano. Posteriormente, apresenta-se o Jogo da Joaninha, composto por duas fases que introduzem e exploram a taxa de variação de uma função e o ponto de interceptação com o eixo Y de forma intuitiva. A fundamentação teórica deste trabalho baseia-se nas ideias de Seymour Papert sobre a aprendizagem construtivista e na perspectiva dos nativos digitais proposta por Marc Prensky. A partir dessas abordagens, discute-se a importância de metodologias ativas que integrem recursos tecnológicos com abordagens teóricas no ensino de Matemática. Em alinhamento com a Base Nacional Comum Curricular (BNCC), destaca-se a necessidade de conectar o conhecimento formal a experiências lúdicas e significativas, fortalecendo a motivação dos alunos. Além disso, com base nas propostas de Mattar, enfatiza-se a importância de integrar jogos educativos e ferramentas digitais no ambiente pedagógico, contribuindo para a inovação curricular e o desenvolvimento de aprendizes mais autônomos e críticos.

Palavras-chaves: Construct 3; Metodologias ativas; Plano cartesiano; Funções afim; Jogos didáticos.

Abstract

This work investigates the use of the Construct 3 platform as a tool for developing educational games aimed at Mathematics teaching, with an emphasis on the exploration of the Cartesian plane and affine functions. The proposal seeks to create interactive and playful environments that enable students to exercise fundamental concepts of Analytical Geometry and Algebra, such as point location, translation, scales, and the slope of lines, establishing practical connections with theoretical content. The dissertation initially describes the construction of a simplified game called "Carteiro Aventureiro" ("Adventurous Postman"), in which the player enters coordinates to move an object on the plane. Subsequently, the "Jogo da Joaninha" ("Ladybug Game") is presented, comprising two phases that intuitively introduce and explore the rate of change of a function and the point of intercept with the Y-axis. The theoretical foundation of this work is based on Seymour Papert's ideas about constructivist learning and on Marc Prensky's perspective of digital natives. Building upon these approaches, the importance of active methodologies that integrate technological resources with theoretical approaches in Mathematics teaching is discussed. Aligned with the Base Nacional Comum Curricular (BNCC), the need to connect formal knowledge with playful and meaningful experiences is underscored, thereby strengthening student motivation. Furthermore, drawing on Mattar's proposals, the relevance of integrating educational games and digital tools into the pedagogical environment is highlighted, contributing to curricular innovation and the development of more autonomous and critical learners.

Key-words: Construct 3; Active methodologies; Cartesian plane; Linear functions; Educational games.

Sumário

Lista de Figuras

Introdução

Nas últimas décadas, a educação tem passado por transformações profundas, resultantes dos avanços tecnológicos que alteraram as ferramentas disponíveis para ensinar e as formas de aprender. Esse fenômeno é amplificado pelo surgimento dos chamados nativos digitais, uma geração que cresceu imersa em tecnologia e, consequentemente, desenvolveu modos distintos de processar informações e resolver problemas. Segundo Prensky (2001) [22], os estudantes de hoje já não correspondem ao perfil para o qual o sistema educacional tradicional foi projetado, pois tendem a preferir abordagens mais visuais, interativas e não lineares, valorizando experiências que façam uso de tecnologia e permitam multitarefas. Dessa forma, métodos convencionais precisam ser complementados por práticas pedagógicas dinâmicas, colaborativas e digitais, a fim de engajar e fortalecer o aprendizado dessa nova geração.

A Base Nacional Comum Curricular (BNCC) [3] também reconhece a importância de preparar os estudantes para as demandas tecnológicas contemporâneas, indo além da simples aquisição de conteúdos tradicionais. O pensamento computacional, conforme definido na BNCC, envolve a habilidade de compreender, modelar e resolver problemas de maneira lógica e estruturada, utilizando princípios da computação para automatizar e sistematizar soluções. Nesse sentido, a Matemática é uma área privilegiada para o desenvolvimento dessas competências, pois viabiliza a tradução de situações-problema em representações diversas (fórmulas, gráficos, tabelas), além de incorporar estratégias algorítmicas para decompor problemas complexos em tarefas mais simples. De fato, a BNCC enfatiza que a aprendizagem de Álgebra, Números, Geometria, Probabilidade e Estatística pode contribuir diretamente para o desenvolvimento do pensamento computacional:

> "[...] a aprendizagem de Álgebra, como também aquelas relacionadas a Números, Geometria e Probabilidade e estatística, podem contribuir para o desenvolvimento do pensamento computacional dos alunos, tendo em vista que eles precisam ser capazes de traduzir uma situação dada em outras linguagens, como transformar situações-problema, apresentadas em língua materna, em fórmulas, tabelas e gráficos e vice-versa." (BRASIL, 2018) [3]

Nesse panorama, a criação de jogos digitais desponta como uma estratégia didática que possibilita desenvolver tanto a matemática quanto o pensamento computacional,

aliando experimentação, erro construtivo e engajamento. As ideias de Papert (1996) [20] reforçam essa visão ao propor micromundos nos quais os estudantes constroem e interagem livremente, aprendendo de maneira significativa. Paralelamente, Mattar (2013) [15] destaca como o erro, nos ambientes de jogo, é encarado como parte natural do processo, incentivando os alunos a persistir na busca de soluções criativas:

> "O papel do fracasso em videogames é muito diferente do que na escola, que não integra a colaboração e a competição como nos games. [...] O fracasso ao matar um mestre, por exemplo, é em geral encarado como uma maneira de aprender e, numa próxima oportunidade, tentar vencer." (MATTAR, 2013) [15]

No cenário atual, outro desafio enfrentado pelos professores é competir pela atenção dos estudantes em meio a uma abundância de informações rápidas e fragmentadas, acessadas principalmente por meio das redes sociais. Carr (2010) [4] chama a atenção para como esse excesso de estímulos pode reduzir a capacidade de concentração em temas complexos. Entretanto, quando bem direcionado, o uso de dispositivos como celulares pode se tornar um aliado, canalizando a familiaridade dos alunos com a tecnologia para atividades educacionais (ETCUBAN; PANTINOPLE, 2018) [7].

Para explorar esse potencial, o presente trabalho apoia-se na plataforma Construct 3, que possibilita criar jogos tanto para computadores quanto para *smartphones* de forma relativamente simples. Essa ferramenta se destaca por:

- Interface intuitiva baseada em eventos, o que dispensa conhecimentos avançados de programação;
- Compatibilidade multiplataforma, ampliando a difusão dos jogos criados;
- Enfoque no raciocínio lógico, o que favorece a conexão com conteúdos matemáticos.

Dessa forma, o desenvolvimento de jogos não se resume ao uso passivo de softwares educacionais prontos: os estudantes tornam-se criadores, construindo personagens, programando mecânicas e elaborando narrativas que podem envolver múltiplas áreas do conhecimento. Além de Arte, aplicada à criação de elementos visuais (pixel art, design de personagens, cenários), pode-se incorporar Língua Portuguesa (na elaboração de roteiros e diálogos), Física (em leis que regem gravidade e movimentação), Inglês (ao trabalhar com vocabulário técnico de programação e design de jogos, além da diversidade de materiais, como tutoriais, manuais e fóruns na língua inglesa) e Matemática (ao exigir conceitos como plano cartesiano, escalas e funções). Tais processos estimulam o pensamento computacional, fortalecem o raciocínio lógico e tornam o aprendizado mais envolvente e contextualizado.

Delimitação do Estudo

Este trabalho se caracteriza como um relato de prática e de desenvolvimento de recursos para o ensino de Matemática, tendo o Construct 3 como ferramenta principal. A proposta foi aplicada em uma disciplina eletiva ofertada a turmas de Ensino Médio da Escola Cidadã Integral Cônego Francisco Gomes de Lima, sem realização de questionários formais ou coleta de dados identificáveis dos discentes. Assim, o escopo deste trabalho limita-se a demonstrar como projetos de jogos podem ser estruturados e inseridos no contexto escolar, oferecendo observações pontuais sobre engajamento e percepção docente, sem constituir um experimento empírico pautado em análises quantitativas.

Objetivos

Para organizar as metas do trabalho, definiram-se um objetivo geral e alguns objetivos específicos:

Objetivo Geral

Demonstrar como o Construct 3 pode ser empregado para criar e implementar jogos digitais que auxiliem na compreensão de conceitos matemáticos, em especial aspectos do plano cartesiano e de funções.

Objetivos Específicos

1. Justificar a escolha do Construct 3 face a outras ferramentas, destacando sua acessibilidade e adequação às demandas pedagógicas.

2. Descrever o processo de elaboração de protótipos de jogos que envolvam noções de Geometria Analítica, tais como coordenadas e transformações.

3. Relatar a experiência de aplicação dos protótipos em uma disciplina eletiva, discutindo de forma informal o engajamento estudantil e as possibilidades de interdisciplinaridade.

4. Apontar as limitações e os potenciais dessa abordagem, indicando como futuras pesquisas podem investigar seus efeitos quantitativos na aprendizagem.

5. Prover orientações para professores que desejem replicar ou adaptar os jogos, bem como incentivar a criação autoral em sala de aula.

Encaminhamento do Trabalho

Para atingir esses objetivos, o texto se organiza da seguinte forma:

- **Capítulo 1** aborda a fundamentação teórica, apresentando as bases de metodologias ativas, nativos digitais e pensamento computacional na BNCC.
- **Capítulos 2 a 8** abordam detalhadamente o Construct 3, explicando as razões de sua escolha para o desenvolvimento do jogo Carteiro Aventureiro e apresentando suas principais funcionalidades. Ao longo desses capítulos, inicia-se o desenvolvimento do jogo, com foco na aplicação prática das ferramentas e recursos oferecidos pela plataforma. A medida que o texto evolui, conceitos matemáticos são introduzidos progressivamente, contextualizando a aplicação desses conteúdos no processo de criação do jogo. Além disso, em alguns capítulos, são feitas sugestões de como o professor pode trabalhar de forma prática e didática, incentivando os estudantes a explorarem a matemática enquanto desenvolvem suas habilidades de programação e design de jogos.
- **Capítulo 9** apresenta duas sugestões de como o professor pode aplicar esse projeto na escola, além de um breve relato de experiência obtido durante sua implementação em uma disciplina eletiva.
- **Capítulo 10** aborda a construção do jogo da Joaninha, oferecendo um processo mais direto e objetivo. O capítulo introduz o uso de mosaicos (tilesets), destacando a criação e aplicação de recursos gráficos, como a arte do jogo, para compor cenários e elementos visuais. Além disso, são apresentadas estratégias didáticas para que o professor possa introduzir, de forma prática, os conceitos de função linear e função afim, preparando o terreno para um estudo mais aprofundado e rigoroso desses conceitos matemáticos.
- **Conclusão** retoma as contribuições do trabalho, aponta limitações e sugere novas possibilidades de aplicação e aprofundamento.

Assim, espera-se mostrar que a criação de jogos no Construct 3 pode potencializar o ensino de Matemática ao conectar conceitos abstratos a práticas concretas, promovendo maior protagonismo dos alunos e incentivando a interdisciplinaridade de forma contínua e criativa.

Capítulo 1

Fundamentação Teórica

1.1 Metodologias Ativas no Ensino de Matemática

O conceito de metodologias ativas envolve estratégias didáticas em que os estudantes se tornam sujeitos centrais do processo de aprendizagem, deixando de ser meros receptores de conteúdo para assumirem um papel de cocriadores do próprio conhecimento [15, 20]. Em vez de ouvintes passivos, os alunos são convidados a experimentar, colaborar e resolver problemas de forma autônoma. Na educação matemática, essa abordagem tem o potencial de favorecer a compreensão dos conceitos, pois exige que o estudante estabeleça conexões entre teoria e prática, associando o abstrato ao contexto concreto [24].

Uma das vantagens das metodologias ativas é a valorização do erro como parte natural do processo. Em ambientes tradicionais, o receio de falhar inibe a criatividade e a busca por soluções alternativas. Já em estratégias ativas, o erro se torna um ponto de partida para a autorreflexão, incentivando a persistência e a autonomia [20]. Quando se agrega a gamificação ou a criação de jogos, esse efeito pode ser intensificado, pois a dinâmica dos ambientes de jogo encoraja múltiplas tentativas e exploração de caminhos criativos, como destacam autores como Mattar (2013) e Papert (1996).

1.2 Nativos Digitais e a Geração Conectada

O termo nativos digitais foi cunhado por Prensky (2001) para descrever a geração que cresceu em um mundo permeado por dispositivos eletrônicos, internet e redes sociais [22]. Esses estudantes, segundo o autor, desenvolvem uma relação diferenciada com a tecnologia, processando informações de maneira mais rápida, multitarefa e multissensorial.

Para a prática docente, compreender o perfil dos nativos digitais implica repensar as estratégias pedagógicas. Métodos exclusivamente expositivos tendem a não satisfazer a necessidade de imediatismo, interatividade e experimentação que caracterizam essa geração. Em contrapartida, a adoção de ambientes digitais, jogos e criação de projetos

mobiliza o interesse desses alunos, agregando o referencial cultural em que estão inseridos.

Porém, é importante salientar que nem todos os jovens têm acesso pleno aos recursos tecnológicos, evidenciando disparidades de ordem econômica e social. Ainda assim, possibilitar espaços e oportunidades de desenvolvimento digital na escola permite que a familiaridade dos estudantes com o uso de smartphones e computadores seja canalizada para a aprendizagem autônoma e criativa, transformando o uso cotidiano da tecnologia em uma ferramenta de construção de conhecimento [22].

1.3 BNCC e Pensamento Computacional

A Base Nacional Comum Curricular (BNCC) [3] estabelece diretrizes para a educação básica no Brasil, indicando, entre outras competências, a necessidade de desenvolver o pensamento computacional. Essa competência envolve habilidades como decomposição de problemas, reconhecimento de padrões, abstração e criação de algoritmos.

Dentro da área de Matemática, tais processos se manifestam de modo particularmente claro em temas como Álgebra, Geometria Analítica e Probabilidade, onde o raciocínio lógico e a representação simbólica são elementos fundamentais. O pensamento computacional, no entanto, não se restringe à programação formal: trata-se de uma forma de pensar e resolver problemas de maneira sistemática, aplicável a diversas situações [3].

A BNCC enfatiza que seu desenvolvimento deve ocorrer de maneira transversal, permeando diferentes atividades escolares e disciplinas. A criação de jogos no Construct 3 exemplifica essa transversalidade ao requerer a elaboração de eventos, o uso de variáveis e condições, além do planejamento de estruturas lógicas para fases e pontuações. Tais elementos favorecem o contato com o pensamento computacional ao mesmo tempo em que aprofundam conceitos matemáticos (posicionamento de objetos, manipulação de coordenadas, funções simples, etc.).

Adicionalmente, a BNCC reforça a importância de formar estudantes autônomos, colaborativos e criativos, aptos a atuarem na sociedade da informação de forma ética e crítica [3]. Ao se propor a construção de jogos em sala de aula, cria-se um ambiente que converge múltiplas áreas: os discentes mobilizam conhecimentos de Arte (design de personagens e cenários), Língua Portuguesa (roteiro, narrativas) e Matemática (cálculos, planos cartesianos, escalas) para desenvolver projetos que encorajam a autoria e o protagonismo.

1.4 Seymour Papert e o Construcionismo

Seymour Papert (1928–2016) foi um matemático e educador que aprofundou e expandiu as ideias de Jean Piaget. Enquanto Piaget formulou o construtivismo, no qual o conhecimento é construído ativamente pelo sujeito a partir de suas experiências, Papert elaborou o construcionismo, uma vertente que defende a aprendizagem por meio

da construção de objetos externos que podem ser compartilhados e analisados socialmente [18, 19].

O construtivismo parte do princípio de que o sujeito não é um mero receptor de informações, mas sim um agente que constrói esquemas de conhecimento a partir de interações com o meio. Papert, alinhado a essa perspectiva, propõe o construcionismo como um passo adiante: aprender se torna mais significativo quando o indivíduo é estimulado a fazer algo, isto é, a construir artefatos tangíveis (ou simbólicos) que representem ou operacionalizem suas ideias [18]. Dessa forma, a interação prática com ferramentas, materiais e tecnologias estimula reflexões e descobertas que podem potencializar a compreensão de conceitos.

Papert foi pioneiro na defesa do uso de computadores em sala de aula, acreditando que eles poderiam funcionar como micromundos de exploração e experimentação [19]. É importante enfatizar, porém, que Papert não considerava a tecnologia uma solução absoluta para todos os problemas educacionais. Ele via os dispositivos tecnológicos como instrumentos de mediação, cuja efetividade depende de práticas pedagógicas que estimulem a criatividade, a curiosidade e a autorregulação do aprendiz. Em outras palavras, o uso de computadores deve ser inserido em contextos de aprendizagem bem planejados, nos quais os estudantes sejam desafiados a criar e investigar.

Nos ambientes construcionistas, o erro é tratado como parte essencial do processo de aprendizagem. Ao explorar um micromundo digital, por exemplo, o estudante tem a oportunidade de errar, depurar suas hipóteses e reformular estratégias, desenvolvendo pensamento crítico e autonomia [18, 19]. No entanto, ressalta-se que o simples estímulo ao erro, desprovido de orientação, pode levar a concepções equivocadas que permanecem sem correção [8]. Assim, o professor exerce papel fundamental ao orientar a reflexão, propor desafios e promover a discussão dos conceitos por trás das situações-problema.

O construcionismo vem sendo adotado em diferentes propostas pedagógicas, mas não está isento de críticas. Algumas limitações e desafios incluem:

- Falta de recursos e formação docente: Muitos sistemas de ensino carecem de infraestrutura adequada ou de programas de formação que capacitem os professores a utilizar ferramentas digitais de maneira eficaz.
- Sustentação teórica: A criação de ambientes digitais ou de micromundos, por si só, não garante aprendizagens mais profundas; é necessário investigar em que medida esses ambientes promovem reflexões consistentes e duradouras [11].
- Desigualdades socioeconômicas: O acesso limitado a tecnologias pode reforçar disparidades educacionais, exigindo políticas públicas que ampliem e democratizem o uso de recursos digitais.

Embora esses pontos não desacreditem o construcionismo, evidenciam a importância de pesquisas, planejamento e adaptações para cada realidade.

A criação de jogos no Construct 3, proposta neste trabalho, inspira-se nos princípios do construcionismo ao incentivar os alunos a "aprender fazendo", por meio do desenvolvimento prático de protótipos digitais. Reconhecemos o desafio apontado na literatura sobre a falta de preparo dos educadores em usar ferramentas digitais; por isso, esta dissertação busca sanar parte dessa lacuna ao fornecer um suporte didático-metodológico para que professores possam se apropriar do Construct 3. Acreditamos que, ao apresentar um passo a passo de criação de jogos e ao exemplificar conteúdos matemáticos trabalhados na plataforma, os docentes encontrarão meios de incorporar estratégias construcionistas em sala de aula sem exigir conhecimentos avançados de programação ou de design.

Assim, embora a adoção do construcionismo de Papert não seja garantia automática de sucesso, há indícios de que seu enfoque no fazer e na exploração de micromundos digitais pode contribuir significativamente para a motivação, o engajamento e o entendimento de conceitos matemáticos e computacionais. No contexto desta pesquisa, a mediação docente e o planejamento pedagógico adequados procuram unir criatividade, tecnologia e fundamentação teórica para potencializar a aprendizagem de forma ativa e integrada.

Apresentamos quatro pilares fundamentais para a proposta de criação de jogos educativos usando o Construct 3:

1. **Metodologias Ativas**: Respaldam a participação central do aluno na criação de conteúdo, valorizando a experimentação e a valorização do erro como motor de aprendizagem.

2. **Nativos Digitais**: Iluminam a necessidade de propor ferramentas tecnológicas e interativas, que dialoguem com a geração atual e sua familiaridade com o mundo digital.

3. **BNCC e Pensamento Computacional**: Fornecem um arcabouço legal e pedagógico para a adoção de práticas inovadoras, nas quais a programação e a resolução lógica de problemas enriquecem a aprendizagem de Matemática.

4. **Seymour Papert e o Construcionismo**: Ressaltam a importância de um ambiente em que o estudante seja o protagonista, construindo o próprio conhecimento por meio de micromundos que estimulam a experimentação, a criatividade e a autonomia. O erro, nesse contexto, deixa de ser obstáculo e se torna parte essencial do processo de aprendizagem, incentivando a busca de soluções inovadoras e a persistência.

Com esses elementos teóricos, fundamenta-se a proposta de desenvolver e aplicar jogos digitais que não só ensinam conceitos matemáticos, mas que também envolvem criatividade e autoria dos estudantes. No próximo capítulo, serão detalhadas as características do Construct 3 e a forma como essa plataforma pode ser aproveitada para colocar em prática a abordagem discutida.

Capítulo 2

Introdução ao Construct 3

Este capítulo oferece uma visão abrangente sobre o Construct 3, uma plataforma de desenvolvimento de jogos 2D baseada em navegador, que se destaca por permitir a criação de jogos por meio de uma interface visual intuitiva, sem exigir conhecimento em programação. Ao longo das seções a seguir, exploraremos os principais componentes da interface, como a barra de ferramentas, o layout, a folha de eventos e as opções de personalização do ambiente. Também veremos o processo de registro e a criação de um novo projeto, abordando as configurações iniciais recomendadas.

Vale ressaltar que, embora existam outras ferramentas de criação de jogos como *Scratch*, *GameMaker* ou mesmo *engines*[1] de maior complexidade (*Unity*, *Godot*, entre outras), o Construct 3 se diferencia por combinar recursos avançados de desenvolvimento com uma curva de aprendizado relativamente baixa. Como ressaltado por Prensky (2001) [22], os nativos digitais tendem a preferir abordagens práticas e dinâmicas, o que torna este tipo de software ainda mais atraente no contexto educacional. Além disso, a proposta de *learning by doing*[2] (Papert, 1996) [20] é perfeitamente alinhada às práticas interativas e visuais oferecidas pelo Construct 3, favorecendo a experimentação e o protagonismo dos estudantes durante o processo de criação de jogos.

Mesmo que você já tenha alguma familiaridade com o Construct 3, recomendamos a leitura deste capítulo, pois os termos e conceitos apresentados serão frequentemente referenciados ao longo do texto.

2.1 O que é o Construct 3?

O Construct 3 é uma plataforma de desenvolvimento de jogos baseada em navegador, desenvolvida pela Scirra Ltd., que permite criar, testar e exportar jogos de forma rápida

[1] Em desenvolvimento de jogos, uma *engine* é um software que fornece as ferramentas e os recursos necessários para criar e rodar jogos.

[2] *Learning by doing* defende que os alunos aprendem de forma mais efetiva quando estão ativamente envolvidos na construção de projetos ou artefatos que representem suas ideias.

e eficiente. Como sucessor do Construct 2, foi projetado para simplificar o processo de desenvolvimento sem exigir conhecimento em programação. Com uma interface visual intuitiva, os usuários podem arrastar objetos, configurar comportamentos e criar interações por meio de eventos, sem a necessidade de aprender linguagens de programação tradicionais, tornando a ferramenta mais acessível e amigável.

Além disso, o Construct 3 oferece:

- Suporte a extensões e plugins, possibilitando integrações com diversos sistemas e serviços externos, o que amplia suas funcionalidades;
- Exportação multiplataforma, permitindo que os jogos desenvolvidos sejam publicados em formatos HTML5, Android, iOS, Windows, macOS e outros;
- Facilidade para prototipagem, já que a programação visual torna o fluxo de experimentação mais dinâmico, atendendo tanto iniciantes quanto desenvolvedores experientes.

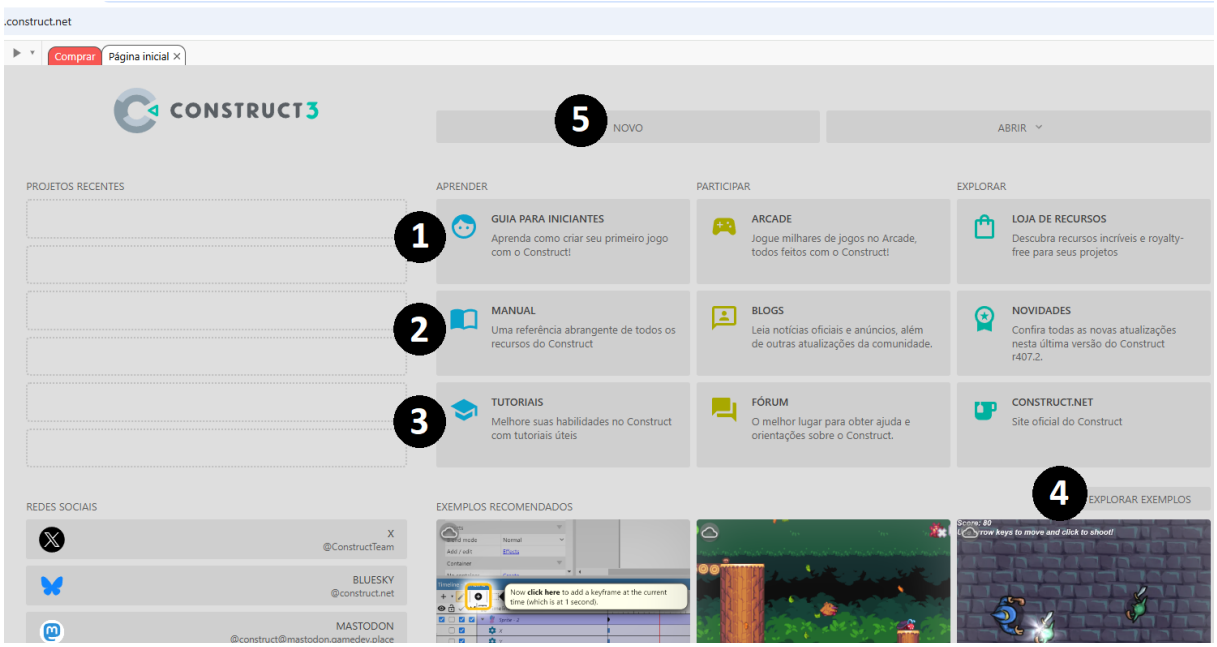
Dessa forma, o Construct 3 se configura como uma solução poderosa e acessível, promovendo uma experiência criativa no desenvolvimento de jogos 2D e tornando-se especialmente relevante no contexto educacional.

2.2 Primeiro acesso ao Construct 3

Para iniciar o desenvolvimento de um projeto na plataforma Construct 3, o primeiro passo é acessar o ambiente de edição online, disponível no endereço oficial: https://editor.construct.net/. Ao acessar a plataforma pela primeira vez, será exibida uma janela de boas-vindas, oferecendo um tour guiado que ajuda a criar um jogo simples do tipo "corra e pule". Recomenda-se aproveitar este tutorial, pois ele proporciona uma visão geral das funcionalidades básicas da ferramenta e facilita o entendimento do restante deste texto.

Na página inicial do Construct 3 (Figura 1), há várias opções que podem ser úteis na construção de diferentes tipos de jogos. Entre as principais, destaca-se o **Guia para iniciantes** ①, que ensina a criar o jogo *Spell Caster*, no qual o jogador controla um mago que precisa lançar feitiços para derrotar inimigos e superar desafios. Ao seguir este guia, o usuário aprenderá a utilizar diversas ferramentas do editor, como a adição de objetos, comportamentos, eventos, lógica de jogo, variáveis e pontuação. Essa opção é especialmente recomendada para quem está começando, além de servir como um bom ponto de partida para professores que queiram usar o Construct 3 em uma disciplina eletiva, por exemplo.

Figura 1: Página inicial do Construct 3.



Fonte: Elaboração própria.

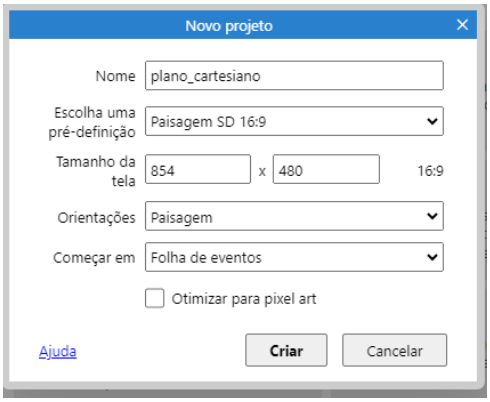
Outras opções importantes incluem o **Manual** (2) e os **Tutoriais** (3), voltados para usuários mais avançados. Além disso, há a opção **Explorar exemplos** (4), onde é possível encontrar diversos jogos prontos. Esses exemplos permitem analisar como os desenvolvedores estruturaram a arquitetura do jogo e oferecem outros tours guiados que ajudam a aprofundar o conhecimento sobre o processo de desenvolvimento.

Para criar um projeto do zero, o usuário deve selecionar a opção **Novo** (5), onde iniciará o processo de configuração do projeto. Será apresentada uma interface para a definição das propriedades básicas do projeto (Figura 2). Nesse momento devemos nos preocupar em preencher apenas o nome do projeto. Outras configurações disponíveis na mesma janela, tais como a pré-definição do projeto, o tamanho da tela, orientações e a opção começar em, não são de interesse neste estágio inicial e devem ser mantidas em suas configurações padrão.

Recomenda-se utilizar letras minúsculas e evitar espaços entre as palavras. Em substituição aos espaços, pode-se utilizar *underscore* "_" ou hífens "-". Além disso, é aconselhável evitar caracteres especiais e números no início do nome.

Após a inserção do nome, o usuário deve proceder clicando no botão `Criar`. Isto fará com que o espaço de trabalho no Construct 3 esteja pronto para uso.

Figura 2: Configurações padrões de um novo projeto.

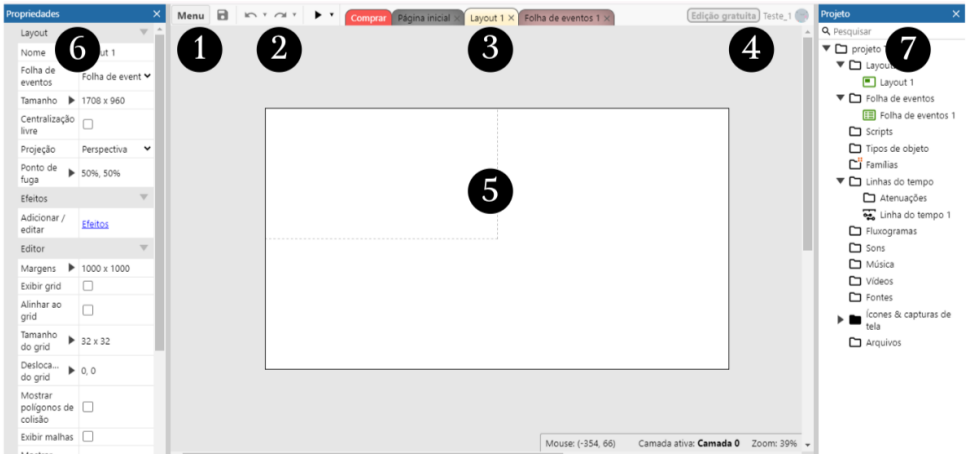


Fonte: Elaboração própria.

2.3 Visão Geral das Principais Áreas do Construct 3

Dentre as diversas disposições e janelas apresentadas pelo software (Figura 3), destacaremos as que serão utilizadas para a elaboração dos produtos finais apresentados neste trabalho.

Figura 3: Interface do Construct 3.



Fonte: Elaboração própria.

1. **Botão de menu principal:** A partir dele, o usuário pode realizar tarefas básicas como abrir e fechar projetos, exportar o jogo, alterar configurações e acessar outras ferramentas importantes do ambiente de desenvolvimento. Ele é essencial para a navegação entre as funções principais da plataforma. Vale ressaltar que, ao abrir

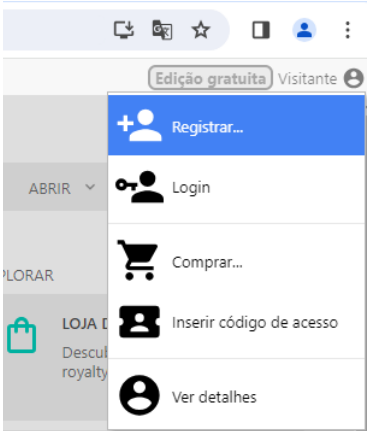
o Construct 3 pela primeira vez, a interface pode variar ligeiramente em relação à que o usuário espera, dependendo de como as barras e painéis estão organizados. No entanto, isso pode ser facilmente ajustado através do botão de `menu`, acessando a opção `exibir` e depois `barras`. Esse menu permite ao usuário personalizar a interface, exibindo ou ocultando diferentes barras de ferramentas e painéis, de modo a adaptar o ambiente de desenvolvimento conforme sua preferência ou necessidade.

2. **Barra de ferramentas principal:** Esta barra proporciona atalhos para as funcionalidades mais utilizadas, como salvar, desfazer, refazer e visualizar a prévia do projeto.

3. **Abas de visualização:** Estas abas permitem que o desenvolvedor alterne entre diferentes *layouts* (onde os elementos que compõe o cenário são posicionados), folhas de eventos (onde a parte lógica do jogo é definida) e página inicial.

4. **Ícone da conta:** O ícone da conta no Construct 3 exibe o status do usuário e, ao ser clicado, abre o menu de gerenciamento. Nele, é possível atualizar informações pessoais, administrar licenças e configurar preferências relacionadas ao uso da plataforma. Embora o software possa ser utilizado sem cadastro, o registro é recomendado para liberar algumas funcionalidades. Usuários não registrados estão limitados a 25 eventos por projeto, uma restrição que pode comprometer o desenvolvimento de jogos mais complexos. Ao criar uma conta, o usuário ganha acesso a funcionalidades adicionais da plataforma, mas o acesso completo a todas as funcionalidades só estará disponível por meio de uma assinatura paga. No entanto, a versão paga não será necessária para o escopo deste projeto. Para proceder com o cadastro, o usuário deve selecionar a opção `Edição Gratuita`, localizada no canto superior direito da interface inicial (Figura 4), seguida pela escolha de `Registrar`. Esta ação direciona o usuário a uma página de registro intuitiva, que permite o uso de contas preexistentes do Gmail ou Facebook para uma inscrição mais ágil. Alternativamente, pode-se optar pelo preenchimento manual dos campos requisitados, que incluem um nome de usuário, uma senha e um endereço de e-mail. Após o preenchimento dos dados, é necessário aceitar os termos de uso da plataforma, finalizando o processo de registro ao clicar em `Let's Go`. Com o cadastro efetuado, o usuário passa a ter a disponibilidade de 40 eventos por projeto, permitindo expandir a complexidade dos jogos criados.

Figura 4: Botão de registro.



Fonte: Elaboração própria.

5. **Área principal de visualização:** Esta é a área central onde o *layout* ou a folha de eventos selecionada é exibida. É aqui que o desenvolvedor passa a maior parte do tempo, visualizando os objetos no layout ou criando a lógica do jogo nas folhas de eventos. O Layout serve como um "palco" onde os objetos são montados e ajustados visualmente, permitindo o arraste e a personalização de componentes em tempo real. Já as Folhas de Eventos funcionam como o "cérebro" do jogo, controlando as interações e a lógica por meio de regras definidas pelo desenvolvedor.

6. **Barra de propriedades:** Aqui o desenvolvedor pode ajustar e personalizar as propriedades dos objetos selecionados. Esta barra permite modificar atributos como tamanho, ângulo, coordenadas, comportamento e efeitos visuais, além de variáveis de instância. É uma ferramenta fundamental para refinar tanto a funcionalidade quanto a estética dos objetos dentro do jogo, oferecendo um controle completo sobre as características dos elementos no layout.

7. **Barra de projeto:** Esta barra lista todos os recursos e componentes do projeto, organizados de forma hierárquica em pastas. Ela oferece uma visão geral e permite navegar facilmente entre diferentes layouts, folhas de eventos e objetos. A pasta de `Tipos de Objetos`, por exemplo, contém personagens, inimigos e outros elementos gráficos essenciais para a jogabilidade, facilitando a gestão e modificação dos recursos do jogo. Com essa organização clara, é possível localizar rapidamente os elementos para edição ou ajustes conforme necessários.

Capítulo 3

Manipulação de Sprites e Fluxo de Trabalho no Construct 3

A partir daqui exploraremos gradualmente as ferramentas disponíveis para a construção do jogo “Carteiro Aventureiro", abordando os recursos necessários para dar vida aos objetos e definir suas interações na lógica do jogo. O objetivo deste jogo é ajudar o carteiro, que começa no centro de um plano cartesiano, a chegar aos pontos marcados no gráfico. Para isso, o jogador deverá inserir as coordenadas de cada destino, de forma que o carteiro siga a rota correta, entregue a encomenda e retorne ao ponto de origem.

Este processo será conduzido de forma gradual, à medida que desenvolvemos o nosso primeiro jogo. Cada recurso será introduzido e explicado detalhadamente conforme sua necessidade no decorrer do projeto. Vale ressaltar que, uma vez apresentado um determinado conceito ou funcionalidade, este não será detalhado novamente em explicações subsequentes. Em vez disso, sua aplicação será direta e progressiva, permitindo o avanço contínuo na construção do jogo sem repetições desnecessárias.

3.1 Objetos

No site oficial do Construct 3 [23], objetos são definidos como os blocos fundamentais que compõem o jogo. Eles representam tudo que pode ser visto ou interagido dentro do ambiente de jogo, desde elementos visuais, como personagens e itens (conhecidos como *sprites*), até objetos invisíveis, responsáveis por funcionalidades, como sons e variáveis.

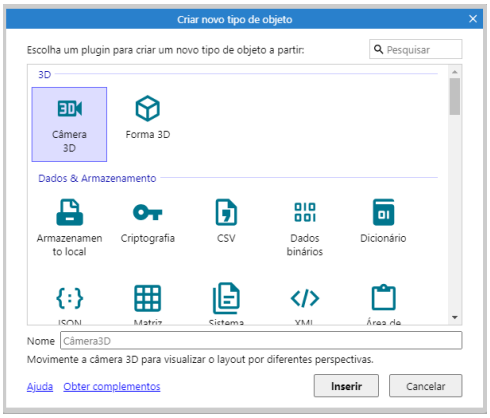
Esses objetos podem ser configurados para responder a diversas interações e comportamentos, como movimentação, colisão ou ações específicas quando um evento ocorre. No processo de desenvolvimento, os objetos definem a aparência do jogo e também sua lógica e comportamento. Portanto, quando adicionamos e manipulamos objetos na folha de layout, estamos construindo tanto a parte visual quanto a lógica do jogo. Cada objeto inserido possui propriedades específicas que podem ser configuradas (veremos mais sobre isso em 3.2.2).

3.1.1 Adicionando objetos na folha de layout

Para colocar um novo objeto na folha de layout do Construct 3, você pode usar um dos dois métodos simples: dê um clique duplo com o `botão esquerdo do mouse` em qualquer parte da área principal de visualização, ou clique com o `botão direito` e selecione a opção `Inserir Novo Objeto`.

Ao fazer isso, a janela `Criar novo tipo de objeto` será aberta mostrando vários tipos de objetos que você pode adicionar ao seu jogo (Figura 5). Esses objetos são organizados em categorias para facilitar a localização do que você precisa. Por exemplo, na categoria `3D`, você encontrará objetos como `câmera 3D` e `forma 3D`.

Figura 5: Janela de objetos disponíveis.



Fonte: Elaboração própria.

No entanto, para este projeto específico, os objetos que mais utilizaremos estão na categoria `Geral`. Dentro desta categoria, os objetos do tipo `sprite` serão amplamente utilizados.

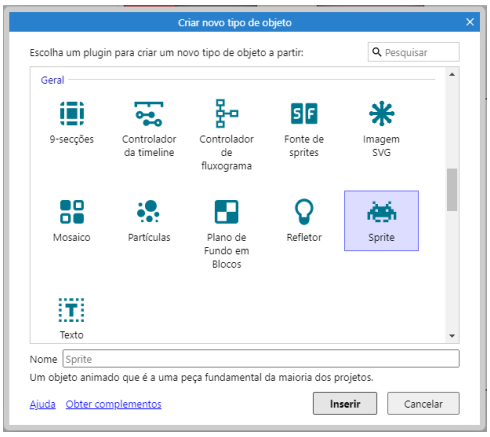
3.2 Sprites

Sprites são as imagens ou gráficos que você vê se movendo e interagindo no jogo. Eles podem ser personagens, inimigos, objetos que o jogador pode coletar, ou qualquer outra coisa visual no jogo. Cada sprite pode ter várias animações, que são diferentes séries de imagens usadas para mostrar movimento. Por exemplo, um personagem pode ter uma animação para correr, outra para pular, e outra para ficar parado. Porém se um Sprite tiver uma única animação com um único quadro, ele apenas mostra uma imagem sem animação.

3.2.1 Criando sprites

Na janela **Criar Novo Tipo de Objeto**, localize e selecione o objeto `sprite` (Figura 6). Após selecionar o sprite, clique no botão `Inserir`. Imediatamente após clicar, o cursor do mouse se transformará em uma cruz. Este cursor em forma de cruz é utilizado para definir onde o sprite será colocado na folha de layout. Com o cursor agora transformado, clique em qualquer local desejado na folha de layout para posicionar o sprite. Escolha um lugar que faça sentido para o design do seu jogo.

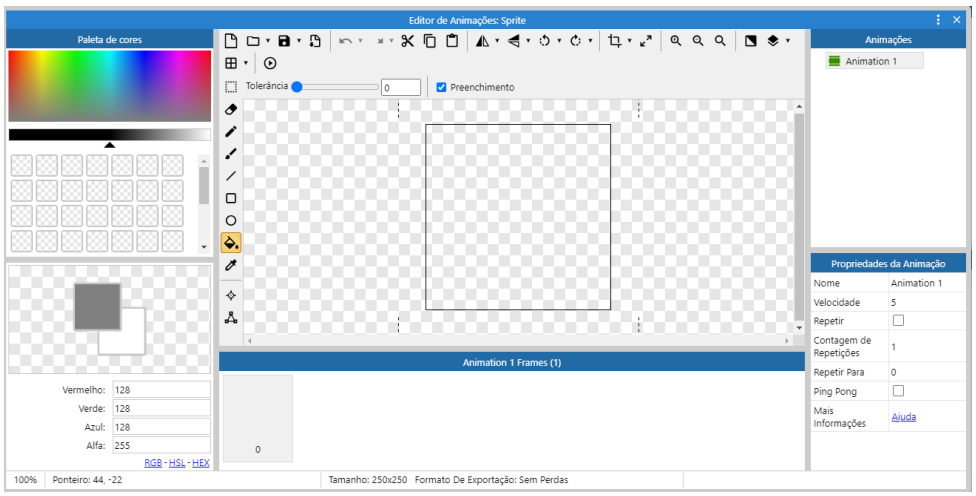
Figura 6: Objeto Sprite Selecionado.



Fonte: Elaboração própria.

Assim que você clicar no layout para posicionar o sprite, automaticamente será aberta uma nova janela chamada **Editor de Animações: Sprite** (Figura 7). Este é o ambiente onde você poderá editar e criar as animações para o seu sprite.

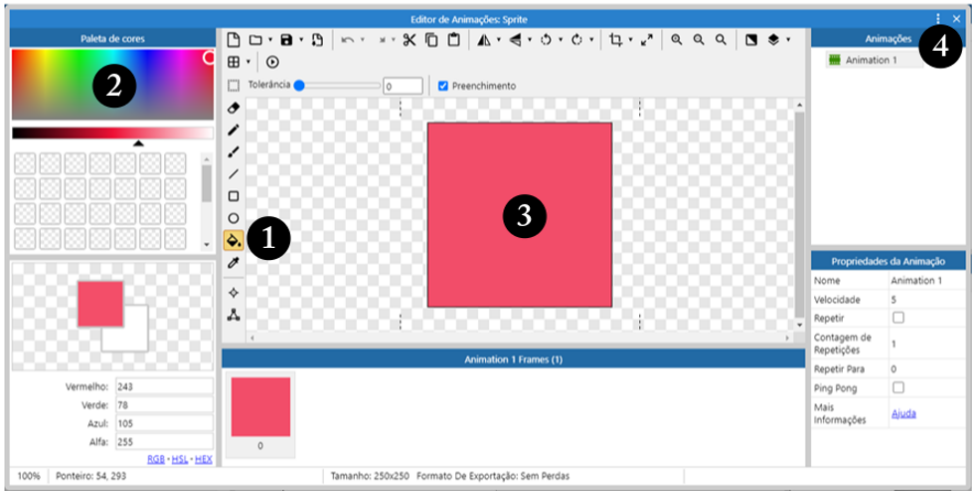
Figura 7: Janela do editor de animações do Sprite.



Fonte: Elaboração própria.

Para nos familiarizarmos com a criação de sprites no Construct 3, começaremos construindo um sprite simples em formato de quadrado que representará o carteiro do nosso jogo. Siga os passos numerados na Figura 8 para concluir a tarefa.

Figura 8: Criando um quadrado passo a passo.



Fonte: Elaboração própria.

1. **Selecionar a Ferramenta de Preenchimento**: Para começar a trabalhar com a coloração do seu sprite no editor de animações, você deve primeiro localizar e clicar no ícone que se parece com um balde de tinta. Este ícone, situado na barra de

ferramentas do editor, é a ferramenta `Preencher`. *Dica: A tecla F funciona como um atalho para selecionar diretamente a ferramenta Preencher.*

2. **Escolher a Cor**: Após ativar a ferramenta de preenchimento, será exibida uma paleta de cores. Selecione a cor de sua preferência para o quadrado.

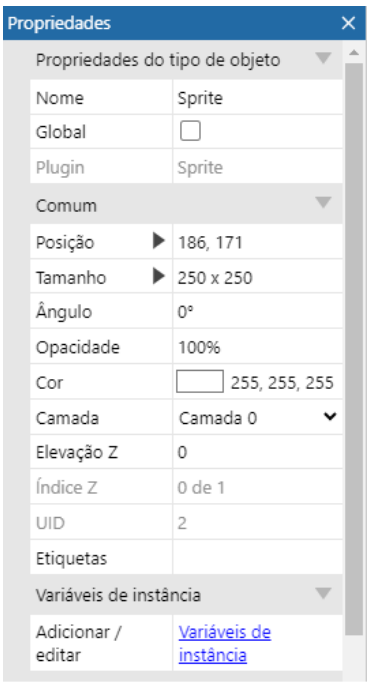
3. **Preencher o Quadrado**: Com a cor selecionada, mova o cursor para o quadrado que está visível no centro da tela do editor. Clique dentro deste quadrado para aplicar a cor. Você observará que o quadrado se preencherá totalmente com a cor escolhida.

4. **Salvar as Alterações**: Após preencher o quadrado, finalize clicando em `Fechar` para salvar as alterações e sair do editor de animações.

3.2.2 Propriedades do sprite

Observe que o sprite já está visível na tela de layout. Agora, podemos acessar a poderosa barra de propriedades deste objeto (Figura 9), onde encontramos uma variedade de opções para personalizar completamente seu comportamento e aparência. Entre essas opções, destacam-se a capacidade de ajustar com precisão o tamanho, a posição, o ângulo e a opacidade, além de adicionar comportamentos inteligentes, efeitos visuais dinâmicos e animações fluídas. Também é possível habilitar ou desabilitar a visibilidade inicial do objeto e configurar as colisões de acordo com as necessidades do projeto.

Figura 9: Parte da barra de propriedades do objeto.

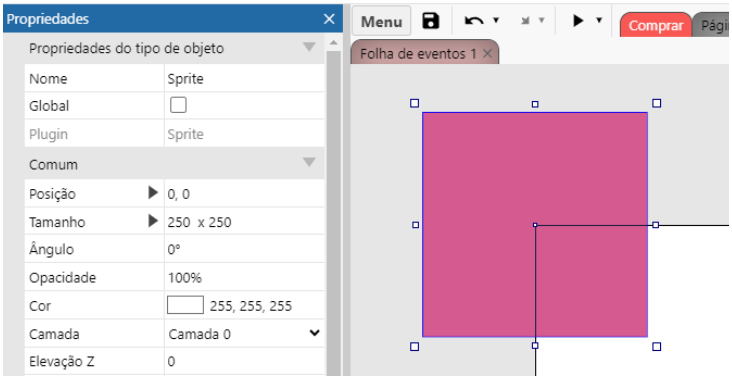


Fonte: Elaboração própria.

3.2.3 Modificando a coordenada de um objeto

Para modificar as coordenadas do objeto, primeiro acesse a **janela de propriedades** e localize a seção `Comum`. Dentro desta seção, você encontrará a opção `Posição`, que define as coordenadas do objeto no layout. A sintaxe para alterar essas coordenadas é simples: você precisa inserir dois valores numéricos separados por uma vírgula, onde o primeiro representa a posição no eixo X e o segundo no eixo Y. Por exemplo, para mover o objeto para a origem do layout, insira `0,0` e pressione a tecla `Enter` para aplicar a mudança.

Figura 10: Quadrado na posição $(0, 0)$.



Fonte: Elaboração própria.

Ao definir a posição de um objeto, como o quadrado do nosso exemplo, para as coordenadas $(0, 0)$, esse objeto será alinhado de modo que seu centro coincida com a origem do sistema de coordenadas do construct (Figura 10). Detalharemos mais sobre esse sistema no Capítulo 4.

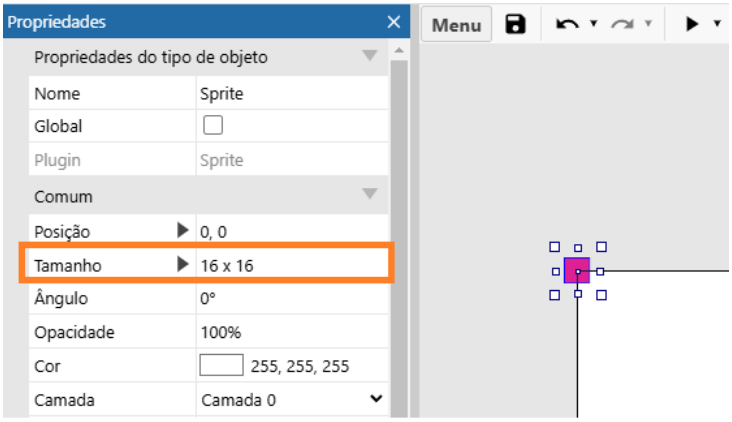
3.2.4 Modificando a o tamanho de um sprite

No desenvolvimento de jogos, é comum precisar ajustar o tamanho de um sprite para padronizar as dimensões dos objetos na tela. Para isso, acesse a barra de propriedades do objeto, localize a opção de tamanho e insira as novas dimensões, separando a largura e o comprimento por vírgula. As medidas devem ser especificadas em pixels. Para ilustrar, vamos definir o lado do nosso quadrado para 16 pixels. Digite `16,16` e pressione `Enter` para aplicar a alteração (Figura 11).

3.3 Executando o jogo

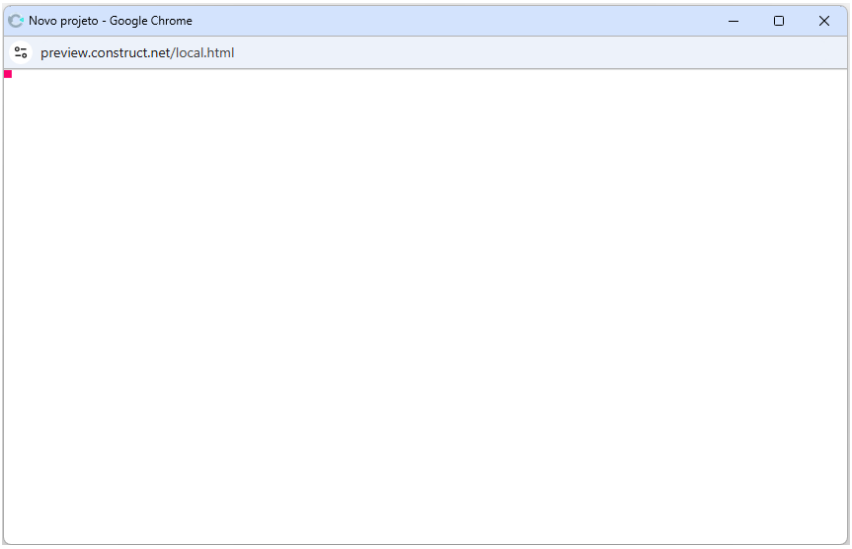
Para visualizar o jogo em funcionamento e observar o que foi construído até este ponto, basta clicar no botão `Play` (ícone ▶) localizado na barra de ferramentas (Figura 3). Esse processo irá rodar o projeto e abrir uma nova janela, onde o jogo poderá ser visualizado em tempo real, exatamente como o jogador final verá. Note que na tela de execução há um retângulo rosa na parte superior esquerda (Figura 12), que é o sprit criado anteriormente.

Figura 11: Quadrado com 16 pixels de lado



Fonte: Elaboração própria.

Figura 12: Janela de execução.



Fonte: Elaboração própria.

Essa etapa é importante para conferir a aparência dos objetos e cenários, e também para verificar se as interações e funcionalidades estão de acordo com o que foi planejado. É nesse momento que o código ganha forma visual e qualquer ajuste necessário pode ser identificado e implementado. Executar o jogo e realizar pequenos testes após cada mudança no projeto é uma boa prática de programação, garantindo que o desenvolvimento se mantenha no rumo certo e que problemas sejam detectados e resolvidos rapidamente.

3.4 Salvando o seu progresso no Construct 3

No Construct 3, salvar frequentemente é essencial para garantir a segurança do seu projeto e evitar a perda de horas de trabalho devido a falhas no sistema ou outros imprevistos. O desenvolvimento de jogos envolve múltiplas etapas de codificação, design e testes, e qualquer erro inesperado pode comprometer o progresso se as mudanças não forem salvas com regularidade. Ao salvar constantemente, você cria uma rede de segurança, permitindo retornar a uma versão estável do projeto sempre que necessário.

Salvar um arquivo no Construct 3 é simples e pode ser feito de diversas maneiras. Para salvar, basta seguir os seguintes passos:

1. Clique no botão de menu principal; (Figura 3).
2. Em seguida, selecione `Projeto` e depois `Salvar como`;
3. Escolha a opção desejada, como salvar na nuvem ou como arquivo local.

O Construct 3 oferece várias formas de salvar, dependendo da sua necessidade e do ambiente em que está trabalhando.

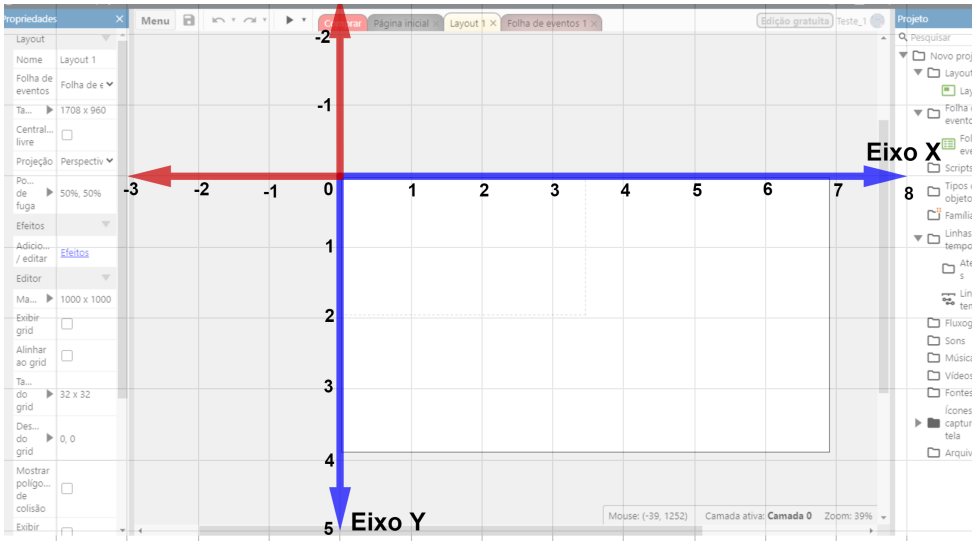
1. **Salvar na nuvem:** O Construct 3 permite salvar o projeto diretamente na nuvem, como no Google Drive, Dropbox ou OneDrive. Isso é útil para acessar o projeto de diferentes dispositivos, mantendo suas versões sincronizadas e seguras online.
2. **Salvar como arquivo único:** Nesta opção, o projeto é salvo em um único arquivo que contém todos os recursos, código e dados do jogo. Esse método é conveniente para manter todo o conteúdo do projeto em um só lugar, facilitando a transferência entre diferentes dispositivos ou colaboradores. Além disso, o formato `.c3p` garante que todas as partes do projeto permaneçam integradas, sem a necessidade de gerenciar arquivos separados. Durante toda a escrita e desenvolvimento do projeto, o salvamento será feito como um arquivo `.c3p` único.
3. **Salvar como pasta de projeto:** Diferente de salvar como um arquivo único, essa opção armazena o projeto em uma pasta, com cada recurso (imagens, sons, etc.) salvo em arquivos separados. É ideal para projetos mais complexos ou colaborativos, onde vários desenvolvedores podem trabalhar em diferentes partes do projeto simultaneamente.

Capítulo 4

Sistema de Coordenadas no Construct

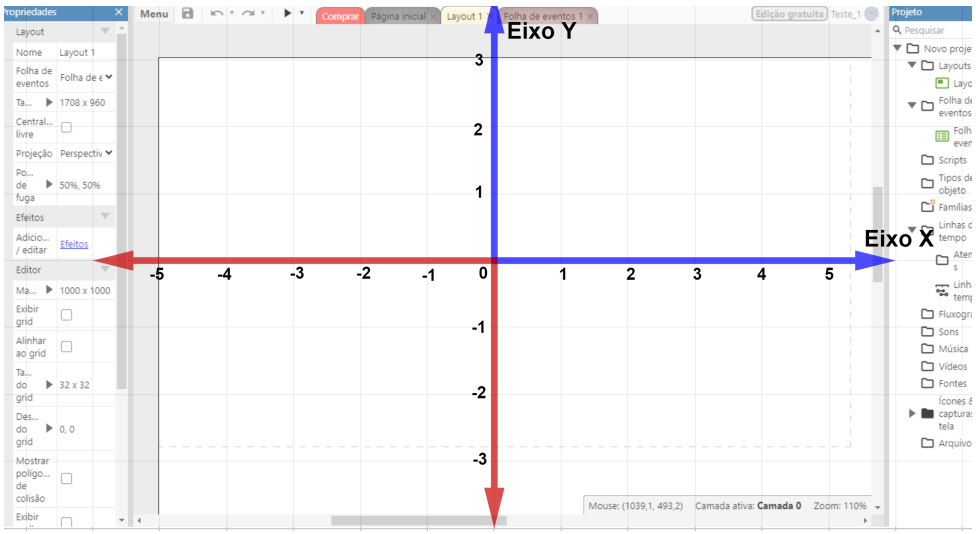
A manipulação da posição de objetos gráficos, como sprites, é uma operação fundamental que permite a configuração espacial desses elementos dentro do ambiente virtual. Entretanto, existe uma diferença entre o sistema de coordenadas adotado no Construct 3 (Figura 13) e o sistema de coordenadas que estamos habituados a utilizar na escola (Figura 14). No Construct 3, a origem está situada no canto superior esquerdo da tela visível, enquanto que no plano cartesiano tradicional, a origem está localizada no centro. Outra diferença notável refere-se ao eixo Y. No sistema de coordenadas do Construct 3, o eixo Y cresce para baixo, ao contrário do plano cartesiano tradicional, onde o eixo Y cresce para cima.

Figura 13: Configuração do sistema de coordenadas adotado pelo Construct 3.



Fonte: Elaboração própria.

Figura 14: Plano cartesiano que habitualmente os estudantes aprendem no ensino médio.



Fonte: Elaboração própria.

Segundo Hughes et al. (2013) [9], em gráficos computacionais, o sistema de coordenadas no plano da tela geralmente possui a origem no canto superior esquerdo, com o eixo X aumentando para a direita e o eixo Y aumentando para baixo. Essa convenção é adotada principalmente por questões práticas e históricas. Primeiro, a forma como textos são lidos e escritos na maioria das culturas ocidentais, da esquerda para a direita e de cima para baixo, influencia essa escolha, refletindo a naturalidade de desenhar e renderizar conteúdos na tela de uma forma semelhante. Além disso, as matrizes que armazenam dados de imagem digital são organizadas de forma que cada linha de pixels é representada sequencialmente na memória, começando do topo para baixo. Isso coincide com a maneira como as imagens são desenhadas nos monitores, que varrem a tela de cima para baixo.

Ao adotar um sistema de coordenadas com o eixo Y crescendo para baixo, simplifica-se a correlação entre a representação lógica dos dados gráficos no software e sua organização física na memória. Isso reduz a necessidade de transformações complexas ou ajustes na renderização de imagens, resultando em um processamento gráfico mais eficiente. A consistência dessa convenção em várias bibliotecas e plataformas gráficas também facilita o desenvolvimento e a portabilidade de aplicações gráficas, oferecendo um padrão amplamente reconhecido que minimiza a curva de aprendizado para desenvolvedores.

4.1 Layout

No Construct 3, o layout é a área onde você constrói visualmente os elementos do seu jogo, como personagens, cenários, botões e outros itens gráficos. Ele funciona como uma

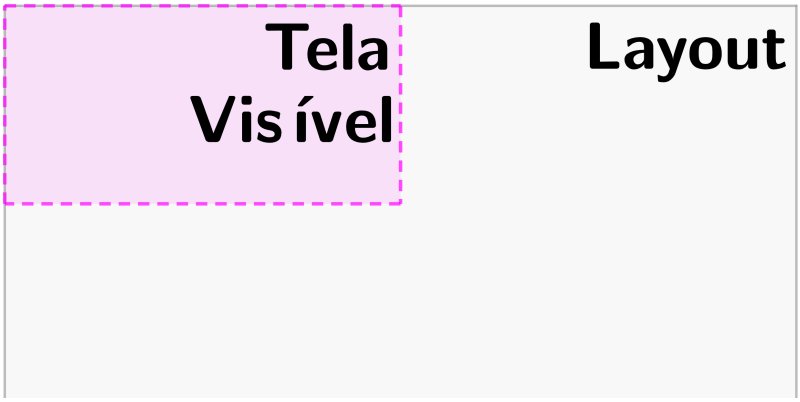
grande tela em branco na qual você posiciona e organiza tudo o que aparecerá no jogo, como se estivesse montando uma maquete digital. Essa área é onde o desenvolvimento visual acontece.

4.1.1 Distinção entre Layout e Tela Visível

No entanto, nem tudo que você colocar no layout será exibido de uma vez só para o jogador. O que o jogador realmente verá enquanto joga é apenas uma parte do layout, chamada de tela visível (Figura 15).

Uma boa analogia para entender isso é imaginar o layout como o cenário de um filme, e a tela visível como a lente de uma câmera. Embora o cenário completo esteja montado, a câmera só filma o que está dentro do seu campo de visão naquele momento. Tudo o que estiver fora dessa área não aparecerá na tela. No jogo, essa "câmera virtual" captura apenas uma porção do layout, que é exibida para o jogador, enquanto o restante do layout permanece oculto.

Figura 15: Layout e Tela visível.



Fonte: Elaboração própria.

Dessa forma, o layout é o espaço total onde você prepara tudo, e a tela visível é a janela pela qual o jogador vê e interage com o jogo. Ao planejar e organizar os elementos no layout, você precisa pensar onde eles estão e em como eles serão exibidos dentro dessa "câmera virtual". Na área principal de visualização do construct 3 é possível notar um retângulo formado por linhas pontilhadas. Esse retângulo está delimitando a área da tela visível.

4.1.2 Manipulando a Visualização do Layout

É importante saber como manipular a visualização do espaço de trabalho no Construct 3. Existem algumas ferramentas e atalhos que facilitam a navegação pelo layout,

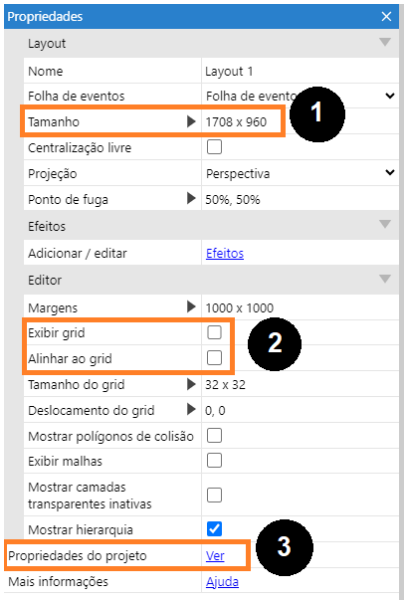
especialmente em projetos com grandes dimensões.

- **Zoom do Layout**: Para aumentar ou diminuir o zoom da área de trabalho, mantenha pressionada a tecla `CTRL` e utilize o `scroll` do mouse. Esse recurso permite uma visualização mais ampla ou mais detalhada de diferentes partes do layout.
- **Movimentação do Layout**: Para mover o layout e explorar outras áreas fora da visualização atual, segure a tecla `Espaço` enquanto move o mouse na direção desejada. Isso facilita a navegação sem alterar a posição dos objetos no layout.

4.1.3 Propriedades do Layout

Assim como os objetos no Construct 3, o layout também possui uma barra de propriedades poderosa, que funciona como um painel de controle central para ajustes fundamentais do projeto. Através dessa janela de propriedades, é possível acessar uma série de configurações que afetam diretamente a estrutura e a funcionalidade da área de trabalho onde os objetos do jogo são posicionados.

Figura 16: Barra de propriedades do Layout.



Fonte: Elaboração própria.

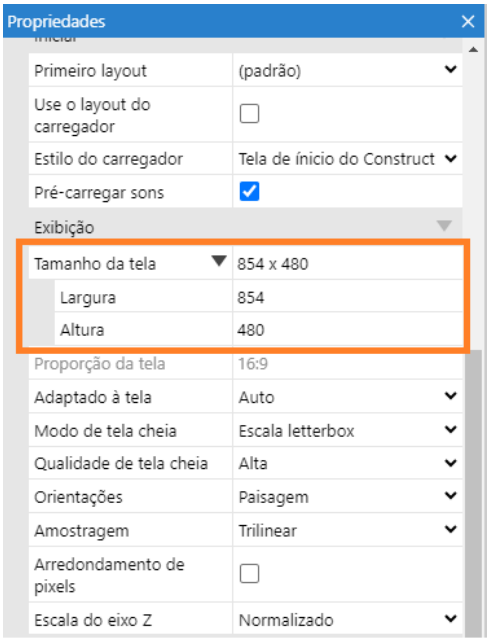
Ao selecionar uma área vazia do layout, a barra de propriedades exibe diversas opções que permitem personalizar aspectos essenciais do layout (Figura 16), como:

1. **Dimensões do layout**: Na aba de propriedades, você encontra as dimensões do layout, expressas em pixels. Estas medidas definem o espaço total disponível para a construção do seu jogo.

2. **Grid e Guias**: A barra de propriedades permite habilitar e personalizar a grade do layout, facilitando o alinhamento dos objetos. O espaçamento entre as linhas da grade pode ser ajustado, ajudando no posicionamento preciso dos elementos gráficos. Esse recurso será utilizado e detalhado em 4.2.2.

3. **Propriedades do projeto**: Nessa seção, ao clicar em `ver` você encontrará as configurações relacionadas ao tamanho da janela do jogo que será exibida ao jogador. Por padrão, o Construct 3 utiliza uma janela com dimensões definidas, mas você pode ajustá-las de acordo com as necessidades do seu projeto. Nesse projeto estamos com `854 pixels` de largura e `480 pixels` de altura (Figura 17).

Figura 17: Tamanho da tela do nosso projeto.



Fonte: Elaboração própria.

4.1.4 Centralizando um objeto na tela visível

Para centralizar um objeto na área visível, é necessário conhecer as dimensões dessa área. Com essas informações, podemos calcular as coordenadas do ponto central e, em seguida, posicionar o objeto nessas coordenadas.

1. **Cálculo do Centro do Layout**: Para calcular o centro da tela visível, basta dividir as dimensões de largura e altura dessa janela por dois. Sabemos que a largura é de `854 pixels` e a altura é de `480 pixels`, o centro será nas coordenadas:

$$X = \frac{854}{2} = 427 \quad \text{e} \quad Y = \frac{480}{2} = 240$$

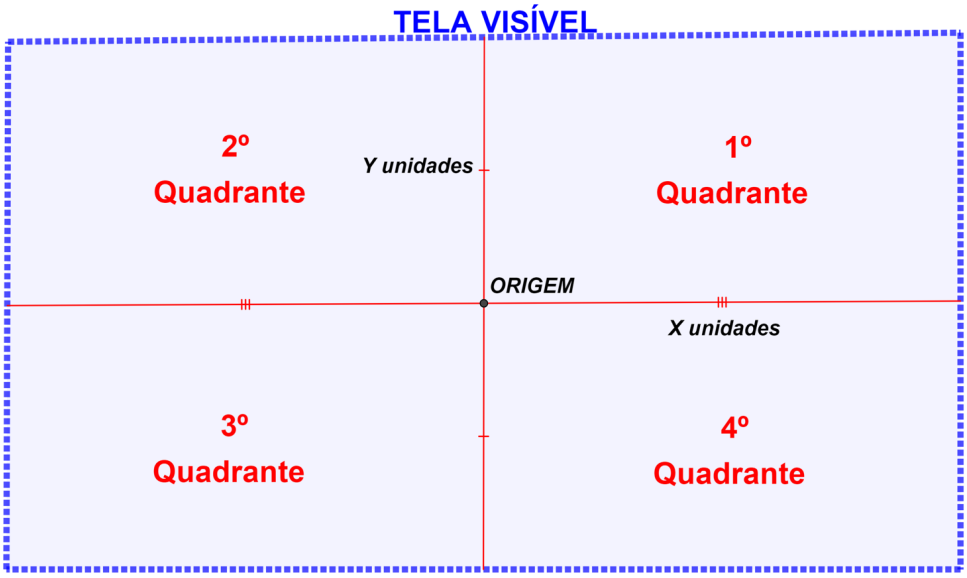
2. **Ajuste de Coordenadas**: Para centralizar o objeto dentro da janela visível, ajuste suas coordenadas (Figura 10) para `(427, 240)`. Isso garantirá que o objeto apareça exatamente no centro da tela durante a execução do jogo.

4.2 Construindo o plano cartesiano

Após esclarecermos as distinções entre o sistema de coordenadas utilizado no Construct 3 e o sistema de coordenadas tradicionalmente ensinado na educação básica, estamos devidamente preparados para iniciar a construção do plano cartesiano que servirá de base para o desenvolvimento do nosso jogo. Para isso, vamos criar dois eixos, X e Y, que servirão como base para o posicionamento dos objetos no ambiente de jogo e orientarão todas as movimentações e interações dentro desse espaço.

Uma decisão importante no desenvolvimento é a escolha de onde posicionar a origem $O(0, 0)$ do nosso plano cartesiano. Neste caso, posicionaremos a origem exatamente no centro da tela visível do jogo, representado pelo retângulo pontilhado azul da Figura 18. Esse posicionamento é estratégico, pois nos permite dividir de forma equilibrada o espaço visível entre os quatro quadrantes do plano cartesiano. Ao colocar a origem no centro da tela visível, garantimos que tanto os valores positivos quanto os negativos dos eixos X e Y tenham uma distribuição simétrica dentro da área jogável.

Figura 18: Esboço do posicionamento da origem do plano cartesiano na tela visível, representado pela borda pontilhada do retângulo azul.



Fonte: Elaboração própria.

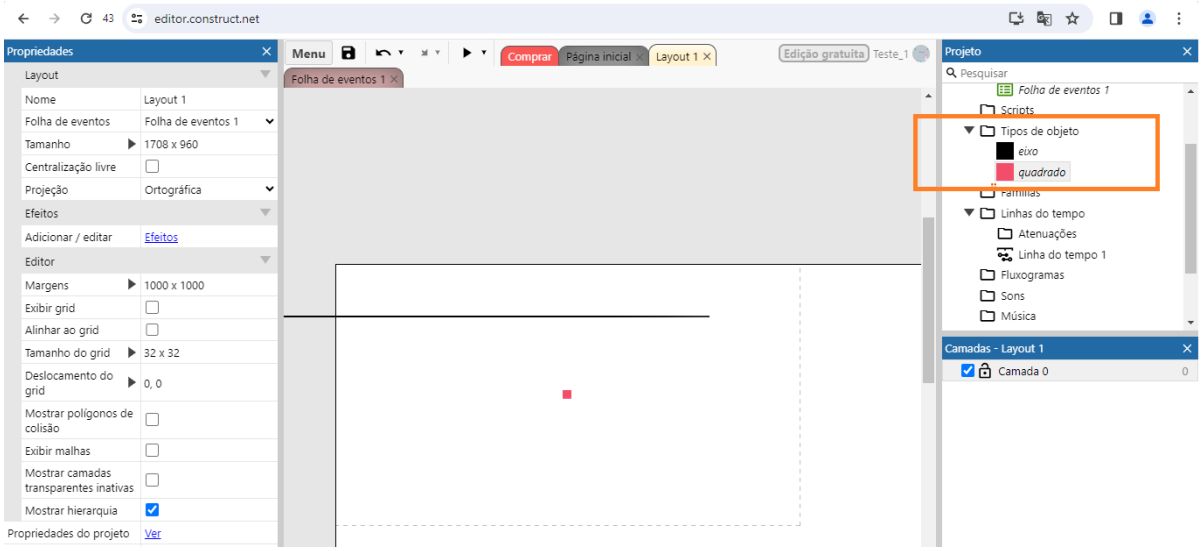
Para começar, adicione um novo sprite ao projeto. Este sprite será utilizado para representar um dos eixos, portanto, é interessante ser pintado com uma cor de destaque. Nesse caso, utilizaremos o preto. Modifique as dimensões deste novo sprite para 1000 pixels de largura e 4 pixels de altura. Essa configuração visa criar uma linha que ultrapasse as dimensões do retângulo visível pelo jogador e mantenha uma espessura discreta e adequada para representar um eixo coordenado.

Na barra de projetos, localizada geralmente no lado direito da interface, você encontrará uma pasta denominada `Tipos de Objetos`. Dentro desta pasta, identifique os sprites adicionados até esse momento. É sempre interessante renomear os sprites de modo que melhor reflitam a sua função. Para isso, clique com o botão direito sobre o objeto e selecione `Renomear`. No contexto do nosso jogo, chamaremos um deles de eixo e outro de quadrado, conforme a Figura 19. Isso facilitará a identificação e manipulação futura dentro do projeto.

Agora, vamos posicionar o eixo no centro da tela visível, cujo centro deverá coincidir com o centro do quadrado. Para isso, basta definir as mesmas coordenadas do quadrado para o eixo (Ver 2.4.4).

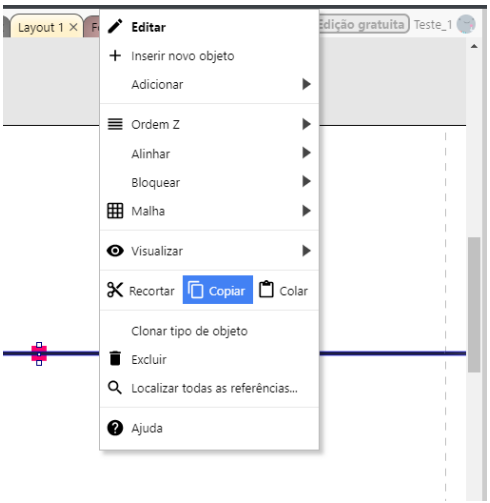
Para adicionar o eixo Y ao nosso projeto, faremos uma cópia do objeto eixo. Portanto, clique com o `botão direito do mouse` no objeto eixo já presente no seu layout, que servirá como base para o eixo Y. No menu que aparece, selecione `copiar` (Figura 20). Isso cria uma cópia exata do sprite.

Figura 19: Localização da pasta tipos de objetos e seus arquivos renomeados.



Fonte: Elaboração própria.

Figura 20: Copiando um objeto que está no Layout.

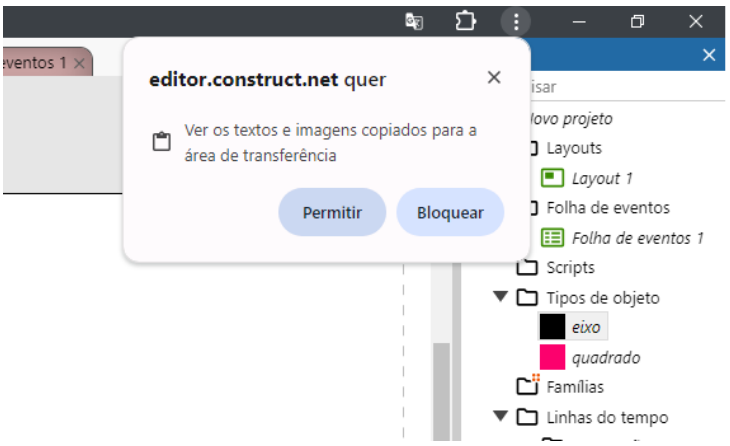


Fonte: Elaboração própria.

Clique com o `botão direito do mouse` em uma área vazia do layout, fora de qualquer outro objeto, e selecione `colar`. Isso insere um novo sprite no layout. Nesse momento pode aparecer uma janela pop-up solicitando permissão para ver textos e imagens copiadas para a área de transferência (Figura 21). Caso isso aconteça, clique no botão permitir e tente colar novamente.

Selecione o sprite recém-colado. Na janela de propriedades, localize a opção para

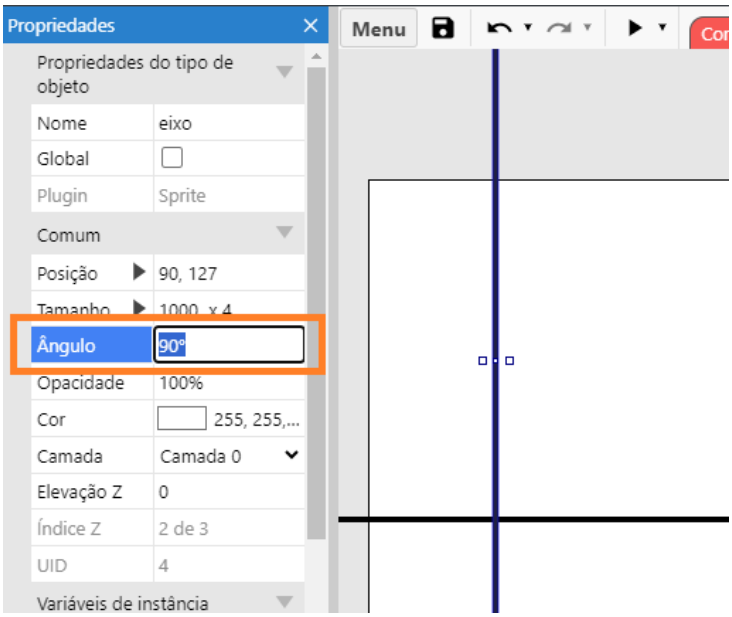
Figura 21: Janela Pop-up.



Fonte: Elaboração própria.

ajustar o ângulo e modifique-o para 90 graus (Figura 22). Isso rotaciona o sprite, transformando a linha horizontal em uma linha vertical, adequada para representar o eixo Y.

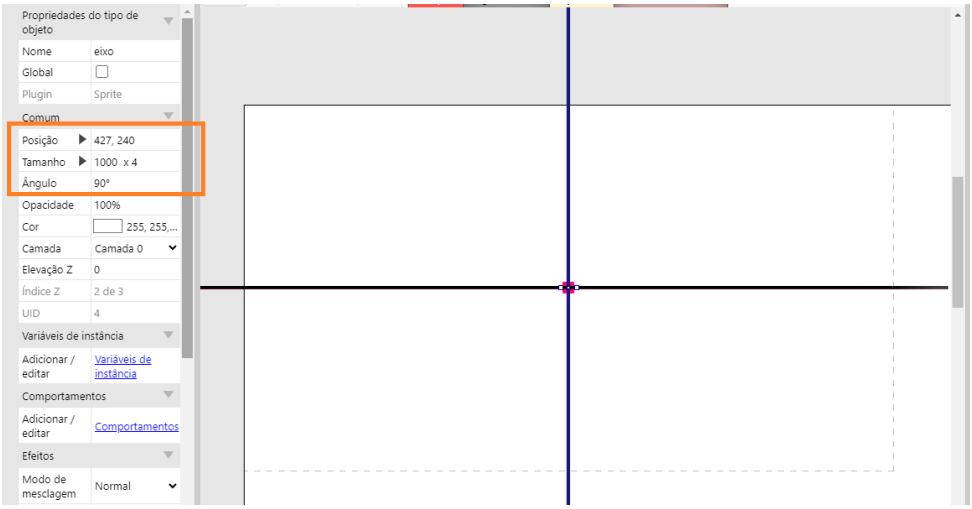
Figura 22: Modificando o ângulo de um objeto.



Fonte: Elaboração própria.

Por fim, repita o procedimento de centralização usado anteriormente para centralizar o eixo Y (Ver 2.4.4) para obtermos o resultado exibido na Figura 23.

Figura 23: Detalhes da posição, tamanho e ângulo da linha que representa o eixo Y.



Fonte: Elaboração própria.

4.2.1 Definindo o tamanho da unidade no plano cartesiano

No estudo do plano cartesiano, a unidade de medida é fundamental e pode ser visualizada como a distância entre a origem e o ponto marcado como 1 no eixo X. Uma vez estabelecida essa unidade, todos os pontos subsequentes no eixo são definidos em múltiplos dessa unidade inicial. Por exemplo, o ponto marcado como 3 no eixo está exatamente três vezes a distância da unidade do ponto de origem.

Em nosso projeto, é crucial definir claramente qual valor será usado como a unidade base. O construct utiliza como unidade a medida de 1 pixel. Esse valor é demasiadamente pequeno para ser utilizado como unidade base. O nosso quadrado, por exemplo, que em relação ao plano deveria ter tamanho desprezível, tem 16 pixels de lado. Logo a manutenção desta escala inicial impossibilitaria a determinação precisa das coordenadas do quadrado, visto que dificultaria a distinção de movimentos minuciosos desse objeto na tela. Portanto, precisamos recalibrar a escala utilizada, fazendo com que 1 unidade no plano cartesiano que estamos construindo correspondam a x pixels na tela do construct.

Nesse momento, caso esteja realizando a construção desse jogo com os estudantes, numa eletiva, por exemplo, temos a oportunidade de definir matematicamente o conceito de escala. De acordo com Bianchini [2], escala é a razão entre uma medida de comprimento de um desenho (ou de uma representação qualquer) e a medida de comprimento real correspondente em uma mesma unidade de medida.

$$\text{Escala} = \frac{\text{medida de comprimento no desenho}}{\text{medida de comprimento real}}$$

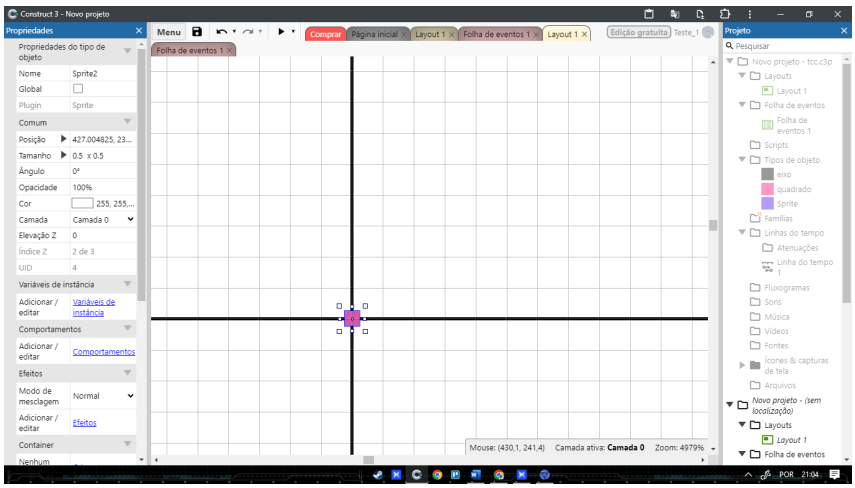
Por exemplo, em mapas, a escala é utilizada para representar proporcionalmente a

distância entre dois pontos no mundo real. Uma escala comum em cartografia é a de 1:50.000, o que significa que 1 cm no mapa corresponde a 50.000 cm (ou 500 metros) no terreno real.

Em nosso projeto, estamos interessados em fazer com que 1 unidade no plano cartesiano corresponda a um número x de `pixels`, com $x > 1$. Mas por que realizar essa alteração? A resposta está relacionada à visualização adequada dos elementos no ambiente digital.

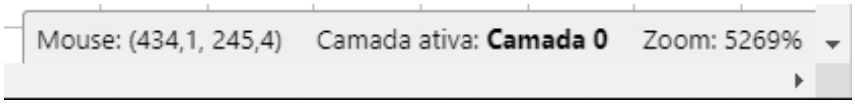
Se utilizarmos a escala 1:1, na qual uma unidade do plano cartesiano corresponde a um único pixel, as dimensões dos objetos no projeto se tornariam tão pequenas que seria necessário aplicar um zoom de aproximadamente 5000% para visualizar um simples quadrado de forma legível. Além disso, trabalharíamos com valores extremamente pequenos, o que tornaria a manipulação dos objetos pouco prática e desgastante.

Figura 24: Esboço de um plano cartesiano construído na escala 1:1.



Fonte: Elaboração própria.

Figura 25: Zoom de 5269% referente ao esboço feito na figura (Figura 24).



Fonte: Elaboração própria.

Por exemplo, ao trabalhar utilizando 1 píxel como unidade padrão, teríamos que fazer um quadrado com lado de 0,5 píxel para que o plano cartesiano fizesse sentido. (Figura 24).

Note que foi necessário aproximar o projeto em 5269% (Figura 25) para trabalhar com 1 píxel de unidade. Sem esse zoom, o quadrado utilizado como exemplo de ponto no plano cartesiano desapareceria devido ao seu tamanho reduzido. As linhas dos eixos também se

tornariam extremamente finas, inviabilizando movimentações precisas e sua visualização no plano. Portanto, ao executar o jogo, nada seria visível, pois não conseguimos distinguir 1 pixel na tela.

Para solucionar o problema de visualização e manipulação dos objetos no plano cartesiano, propomos alterar a escala de 1:1 para um valor que amplie significativamente o tamanho dos objetos, mantendo sua proporcionalidade. Nossa meta é um aumento de aproximadamente 5000%, o que permitirá que os objetos, antes praticamente invisíveis, sejam claramente visualizados sem a necessidade de zoom adicional.

Para realizar esse aumento, partimos da escala de 1 `pixel` para x `pixels`, de forma que a nova escala acomode um aumento de aproximadamente 5000%. Podemos expressar essa ideia da seguinte maneira:

$$1 \texttt{ pixel} + 5000\% \texttt{ de 1 pixel} = x \texttt{ pixels}$$

Ou, mais formalmente:

$$x = 1 + \frac{5000}{100} \times 1$$

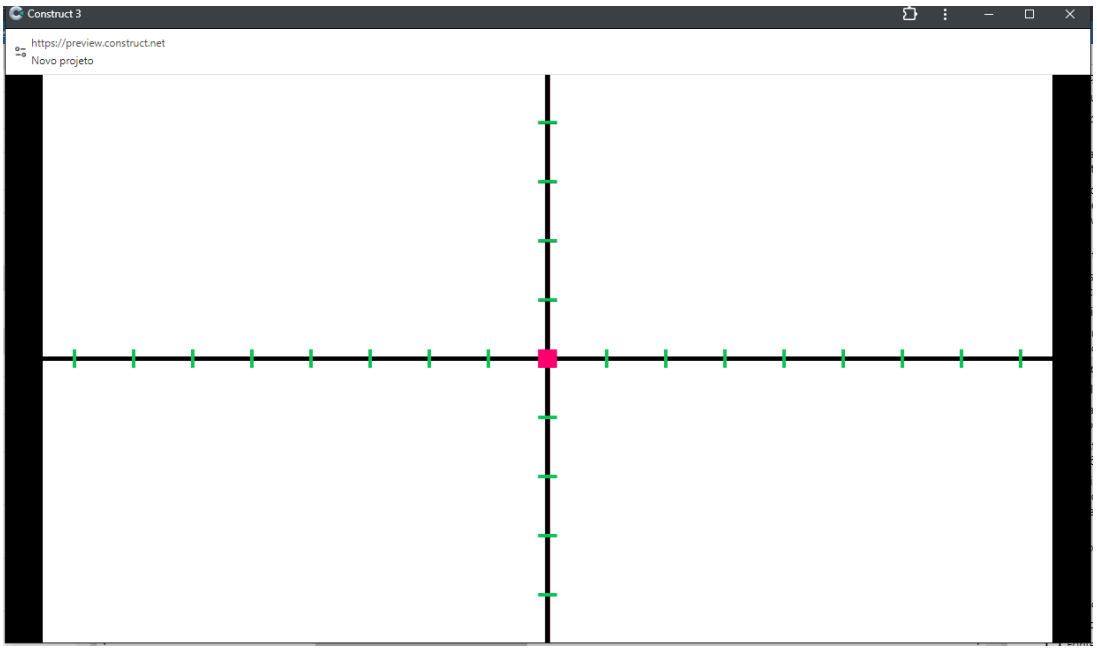
Simplificando, obtemos:

$$x = 1 + 50 = 51$$

Como estamos usando um valor aproximado para o zoom, podemos arredondar x para 50 `pixels`. Assim, adotaremos uma escala de 1:50, o que representa um aumento aproximado de 5000%. Com essa escala, conseguimos ampliar significativamente os objetos, tornando-os claramente visíveis e manipuláveis, mantendo a proporcionalidade e dispensando a necessidade de zoom adicional. Além disso, essa configuração de escala é flexível: pode ser ajustada conforme a necessidade ou preferência do usuário.

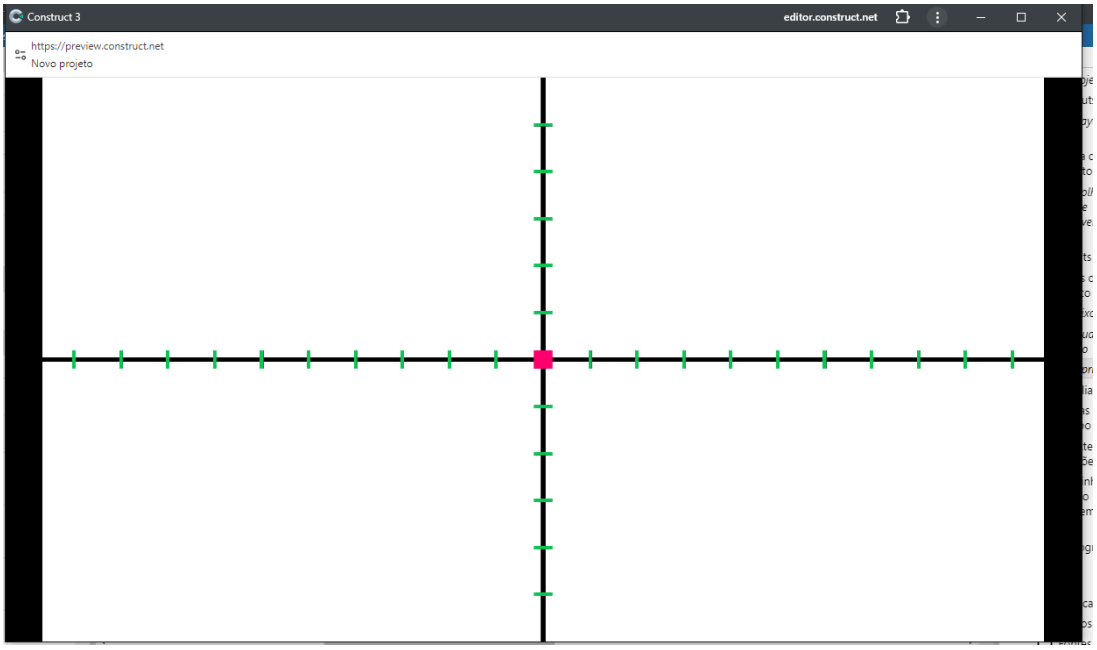
É importante destacar que, ao utilizar uma escala de 1:50, estamos priorizando a clareza visual dos elementos, sem perder de vista as relações de proporcionalidade entre eles. Nas figuras Figura 26 e Figura 27, podemos observar as diferenças entre o plano cartesiano com escalas de 1:50 e 1:40, respectivamente. À medida que diminuímos a escala, mais pontos podem ser inseridos nos eixos, mas os espaços entre eles ficam menores, o que pode comprometer a estética ou funcionalidade do jogo. Por isso, a escolha da escala deve considerar um equilíbrio entre a precisão visual e a praticidade no desenvolvimento.

Figura 26: Plano cartesiano em escala 1:50.



Fonte: Elaboração própria.

Figura 27: Plano cartesiano em escala 1:40.

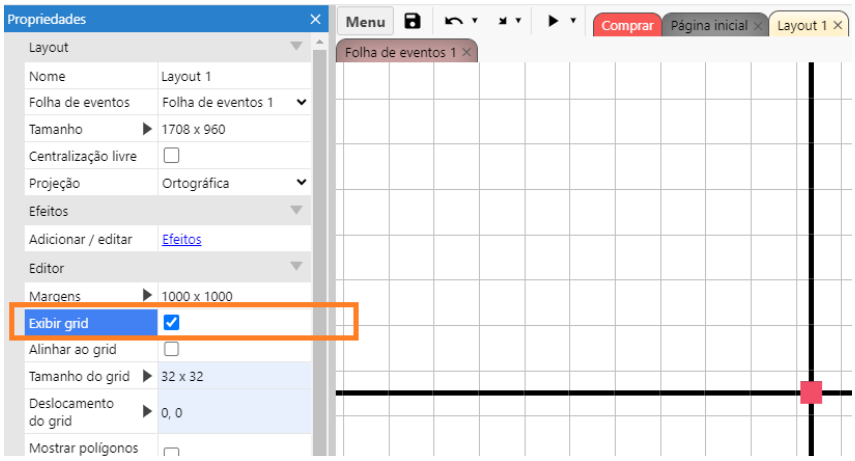


Fonte: Elaboração própria.

4.2.2 Ajustando o Grid

Para alcançar os resultados mostrados nas Figura 26 e Figura 27, é essencial utilizar uma funcionalidade muito útil no Construct 3: a grade, ou grid. Essa ferramenta permite alinhar os objetos de forma precisa, facilitando o posicionamento correto dos elementos no jogo. Por padrão, o grid não está ativado nos projetos. Para habilitá-lo, vá até as propriedades do layout e marque a poção `exibir grid` (Figura 28). As linhas de grade serão exibidas no layout. Estas linhas servem como guias visuais durante a fase de construção do jogo e não aparecerão quando o jogo estiver sendo executado, atuam como ferramenta de suporte nos bastidores do desenvolvimento.

Figura 28: Exibindo as linhas do grid.

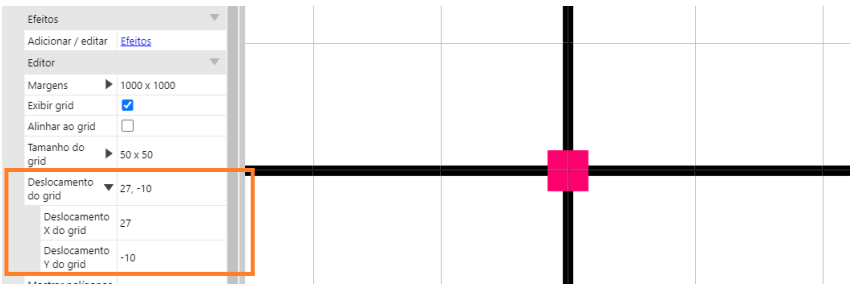


Fonte: Elaboração própria.

Ao configurar o tamanho do grid para `50 x 50`, definimos cada célula da grade com lados de 50 pixels. Essa escolha é diretamente alinhada com a escala adotada, em que cada unidade no plano cartesiano é representada por 50 pixels. Essa configuração cria uma estrutura uniforme que facilita o alinhamento preciso dos objetos no layout, mantendo a proporção da escala definida. Dessa forma, o grid atua como uma ferramenta visual que espelha nossa escala de 1:50, ajudando a posicionar cada elemento nas coordenadas corretas do plano.

Se perceber que o quadrado não está perfeitamente alinhado nas interseções das linhas do grid, será necessário ajustar o grid com deslocamentos horizontais e verticais. Nosso objetivo é posicionar o centro do quadrado exatamente em uma interseção do grid. Para isso, utilize a opção de deslocamento de grid disponível na janela de propriedades, que permite realizar esses ajustes com precisão (Figura 29).

Figura 29: Deslocando o grid para que esteja alinhado com o centro do quadrado.



Fonte: Elaboração própria.

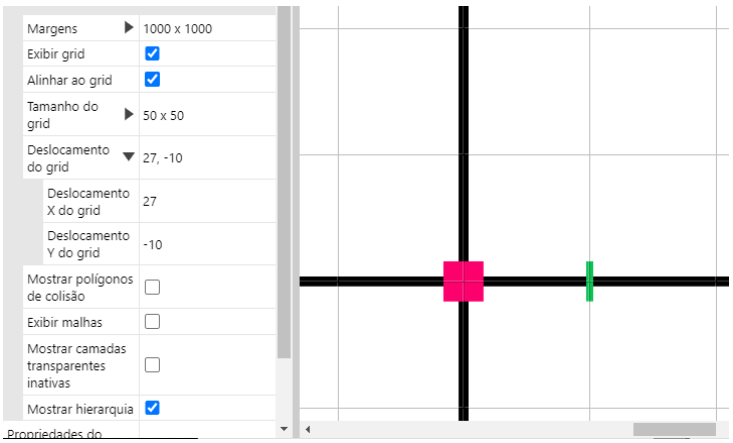
Os valores de `(27 e -10)` da Figura 29 foram encontrados por meio de experimentação visual e ajustes incrementais. Antes de experimentar valores para os deslocamentos X e Y do grid, é interessante aplicar zoom no projeto para que tenhamos mais precisão no alinhamento. Com esses ajustes, o grid foi reposicionado de maneira que suas linhas passaram a coincidir exatamente com o centro do quadrado e com os eixos X e Y, assegurando um alinhamento perfeito entre o grid e o objeto.

Após efetuar o alinhamento necessário, ative a opção `Alinhar ao Grid` encontrada na janela de propriedades. Essa funcionalidade garante que, ao mover qualquer objeto dentro do layout, ele se alinhe automaticamente com as linhas do grid, facilitando a organização e a precisão visual.

4.2.3 Marcando os valores das unidades no plano cartesiano

Vamos agora criar um novo sprite que funcionará como um marcador de unidades nos eixos cartesianos. Inicie criando um sprite de cor verde, seguindo o mesmo processo utilizado para o quadrado inicial. Modifique então as dimensões deste novo sprite para 3 pixels de largura por 16 pixels de altura. Para simplificar futuras referências, renomeie este objeto para `marcador`. Posicione o `marcador` na interseção do eixo X com as linhas do grid, conforme exibido na Figura 30.

Figura 30: Posicionando o primeiro marcador.

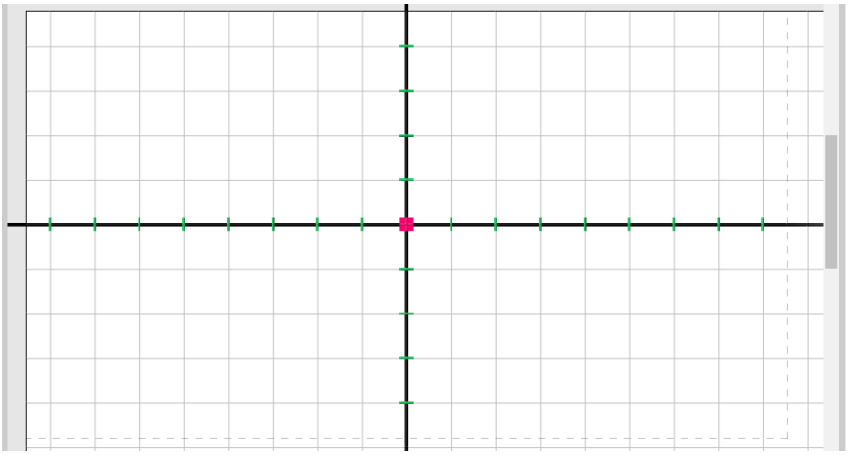


Fonte: Elaboração própria.

Para marcar as unidades subsequentes ao longo do eixo X, copie o marcador utilizando o atalho de teclado `CTRL + C` e `clique` no local desejado para colar o marcador, garantindo que cada novo marcador esteja posicionado nas interseções do eixo X com as linhas do grid.

Para o eixo Y, você precisará fazer uma cópia do marcador e ajustar seu ângulo para 90 graus, procedimento análogo ao que realizamos para criar o eixo Y. (Figura 22), permitindo que ele fique horizontal. Posicione este marcador na interseção do eixo Y com as linhas do grid. Copie este marcador horizontal e continue o processo similarmente ao feito no eixo X, marcando as unidades ao longo do eixo Y. Ao término do processo obteremos um resultado idêntico ao apresentado na Figura 31.

Figura 31: Plano cartesiano após adicionar os marcadores de unidade.

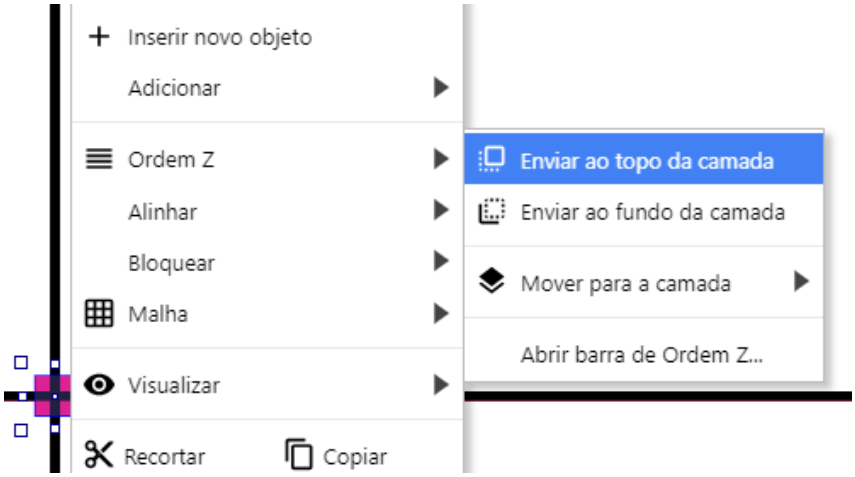


Fonte: Elaboração própria.

4.3 Enviando um objeto para o topo da camada

Caso um objeto esteja sobreposto a outro e você queira alterar essa ordem, é importante entender que o posicionamento é determinado por um sistema de camadas, semelhante ao eixo Z em uma representação tridimensional. Quando um objeto é visualizado acima de outro, ele possui um valor maior no eixo Z. Na versão gratuita do Construct 3, o controle deste eixo é restrito. No entanto, há uma maneira de ajustar essa configuração para alcançar a sobreposição desejada. Clique com o `botão direito do mouse` no objeto posicionado no layout, navegue até a opção `Ordem Z` no menu contextual e selecione `Enviar para o topo da camada` (Figura 32).

Figura 32: Enviando o objeto quadrado para o topo da camada.



Fonte: Elaboração própria.

Agora, o objeto quadrado, que estava abaixo dos eixos, será exibido acima deles. Esse procedimento deve ser repetido sempre que for necessário posicionar um objeto na camada superior, garantindo que ele tenha a visibilidade e a prioridade adequadas no layout do jogo.

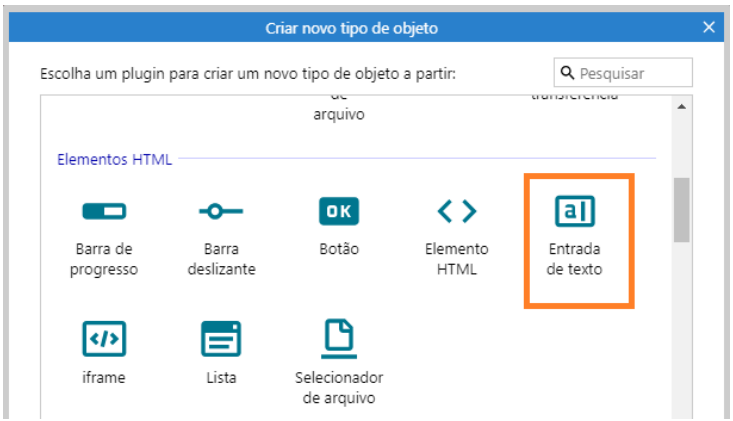
Capítulo 5

Criando a parte lógica

5.1 Capturando coordenadas no plano cartesiano através do jogo

Nesta etapa do desenvolvimento, nosso objetivo é possibilitar que o carteiro do nosso jogo (objeto quadrado) se movimente no plano cartesiano conforme os valores fornecidos pelo jogador. Para tal, é imprescindível que o jogador insira as coordenadas desejadas, fazendo com que o quadrado que está na origem chegue em um ponto aleatório que será gerado ao executar o jogo (faremos isso em breve). Serão necessários dois valores: o primeiro destinado ao deslocamento no eixo X e o segundo para o deslocamento na direção do eixo Y. A captura desses valores será realizada por meio de dois objetos de `entrada de texto`. Para cria-los, basta dar um clique duplo em um espaço vazio do layout e selecionar entrada de texto, que está localizando na seção elementos HTML janela de **Criar novo tipo de objeto** (Figura 33).

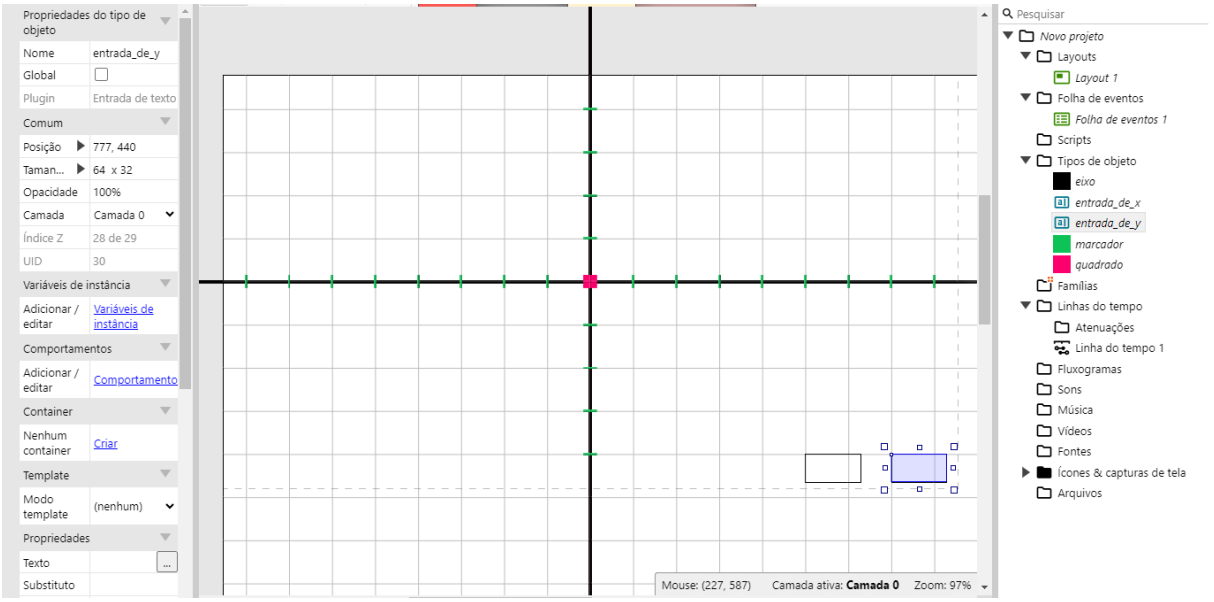
Figura 33: Objeto entrada de texto.



Fonte: Elaboração própria.

Para facilitar a organização e compreensão dentro do jogo, esses objetos serão nomeados como `entrada_de_x` e `entrada_de_y`, correspondendo, respectivamente, à abscissa e ordenada fornecidas pelo jogador. Para aprimorar a interface, ajustaremos o tamanho destes objetos de texto para `64 x 32 pixels` e os posicionaremos no canto inferior direito da tela, próximo à borda pontilhada da tela visível (Figura 34).

Figura 34: Caixas de texto posicionadas no layout.



Fonte: Elaboração própria.

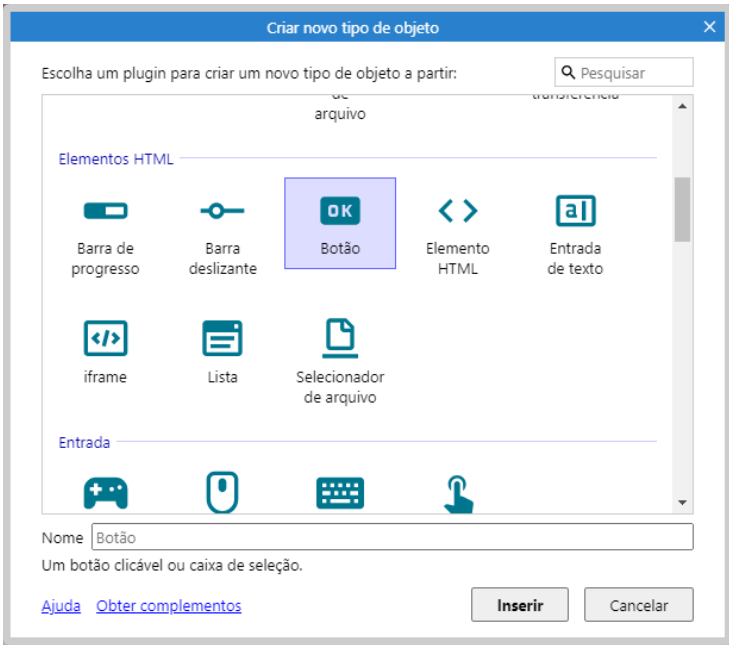
Ao executarmos o jogo, notamos que as caixas de texto já estão operacionais. No entanto, ainda precisamos definir as funções que processarão os valores inseridos. O próximo passo será incluir um `botão OK`. Este botão servirá para indicar ao programa o momento de capturar e utilizar os valores digitados, integrando-os às ações do objeto no plano.

5.2 Implementando o botão de confirmação

Para direcionar o objeto quadrado até as coordenadas inseridas pelo jogador, criamos um botão de confirmação, denominado `OK`. Esse botão servirá como o gatilho para iniciar toda a programação que desenvolveremos adiante. Após o jogador inserir as coordenadas nas caixas de texto e acionar o `botão OK`, o quadrado, que inicialmente está centralizado no plano cartesiano, se moverá automaticamente para as coordenadas fornecidas.

Para adicionar o objeto botão no layout, procure por `botão` na seção `Elementos HTML` da janela de objetos (Figura 35).

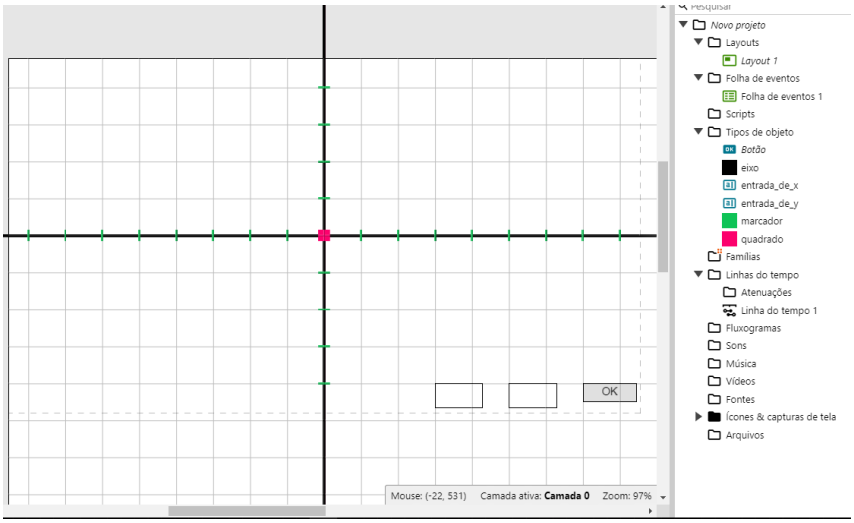
Figura 35: Objeto botão.



Fonte: Elaboração própria.

Ajuste o tamanho do botão em `32 x 32 pixels` para manter a harmonia visual com os demais elementos interativos. Posicione o botão ao lado do objeto `entrada_de_y`, facilitando a interação do usuário (Figura 36).

Figura 36: Botão adicionado ao layout.



Fonte: Elaboração própria.

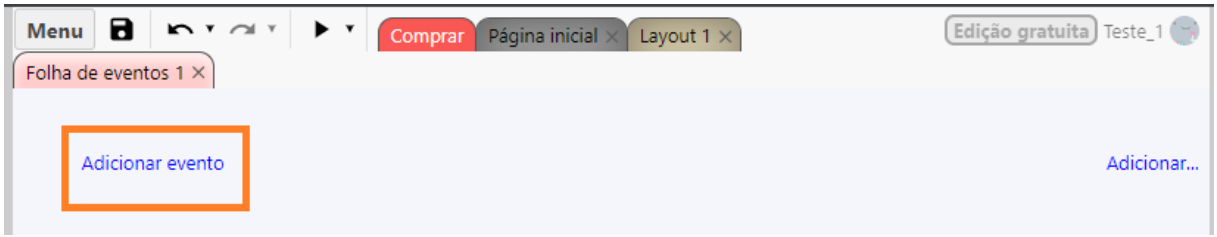
5.3 Eventos

Até o momento, nosso projeto já apresenta visualmente um plano cartesiano, um quadrado que representa o carteiro que irá percorrer o plano, duas caixas de entrada, sendo uma para coletar a abscissa e outra a ordenada do ponto de destino, além de um botão de confirmação `OK`. No entanto, ao executar o jogo, inserir valores nas caixas de texto e pressionar o `botão OK`, nada acontece. Isso é esperado, pois ainda não programamos as ações que o sistema deve executar com esses objetos. Para que possamos definir essas ações, precisamos utilizar a folha de eventos do Construct 3.

A folha de eventos é onde programamos o comportamento dos objetos em resposta a determinadas ações ou condições. Nela, podemos especificar o que acontece quando o usuário interage com a interface, como clicar em um botão ou inserir dados. Por exemplo, ao clicarmos no `botão OK`, queremos que o quadrado se desloque um número m de unidades no eixo X e, em seguida, n unidades na direção do eixo Y, de acordo com os valores inseridos pelo usuário. Para isso, devemos criar um evento que defina esse comportamento.

O processo de criação do evento é simples. Selecione a aba `folha de eventos` na 3 e em seguida clique em `Adicionar Evento` (Figura 37).

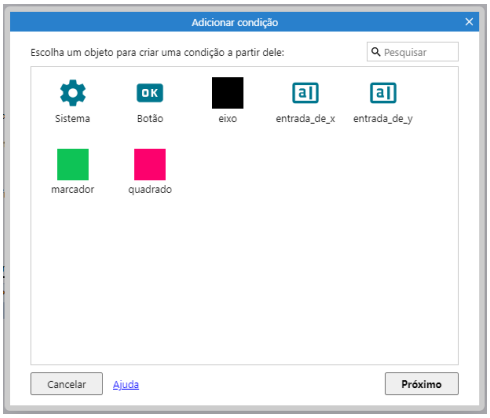
Figura 37: Folha de eventos vazia, com a opção destacada de adicionar evento.



Fonte: Elaboração própria.

Em seguida, uma nova janela será exibida, solicitando a escolha de um objeto para iniciar a criação de uma condição (Figura 38). Nesse caso, o objeto que desejamos usar é o `botão OK`. A partir dele, podemos programar as ações que moverão o quadrado conforme os valores fornecidos.

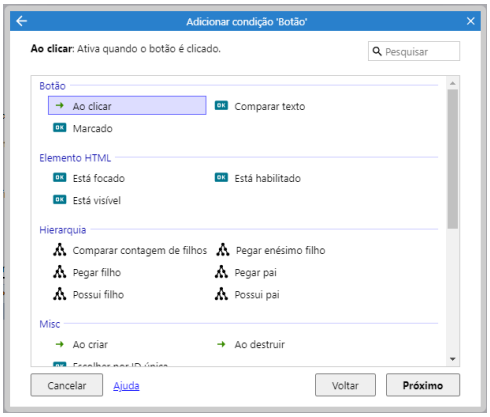
Figura 38: Objetos disponíveis para adicionar uma condição.



Fonte: Elaboração própria.

Queremos que o quadrado seja deslocado ao clicar no `botão OK`. Então selecionamos o objeto botão e em seguida clicamos em `próximo`. Dentre as várias opções fornecidas na tela seguinte, queremos que a ação aconteça quando clicarmos no `botão`. Portanto, selecione `ao clicar` e depois clique em `próximo` (Figura 39).

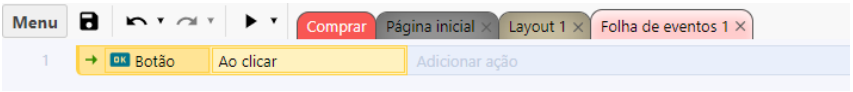
Figura 39: Lista de opções disponíveis para ser utilizadas com o objeto botão.



Fonte: Elaboração própria.

Note que agora a `folha de eventos` não está mais vazia (Figura 40). Acabamos de inserir um evento que será acionado ao clicar no `botão`. Nesse momento se executarmos o jogo e clicarmos em `OK`, nada acontecerá, pois ainda não especificamos o que queremos que aconteça quando clicarmos em `OK`.

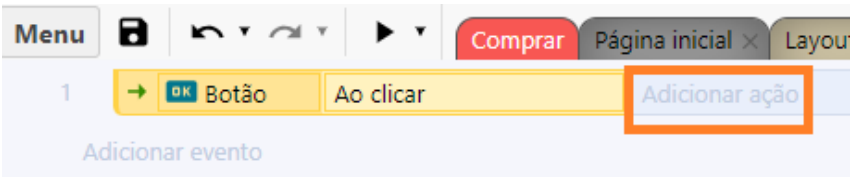
Figura 40: Primeiro evento adicionado na folha de eventos.



Fonte: Elaboração própria.

O construct está preparado para executar uma ação quando clicarmos em `ok`, porém é necessário "ensinar" o que deve ser feito quando clicarmos no `botão ok`. Faremos isso nesse momento. Ainda na folha de eventos, clique em `adicionar ação` (Figura 41).

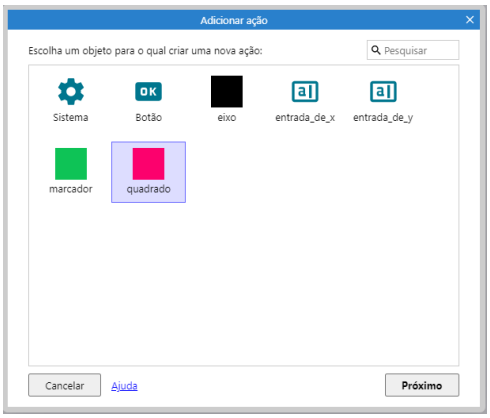
Figura 41: Opção de adicionar ação.



Fonte: Elaboração própria.

Recorde que queremos que o quadrado se desloque quando clicarmos no `botão OK`. Então agora devemos selecionar o quadrado, pois é esse objeto que queremos que uma ação aconteça após clicar no `botão OK`. Portanto, `selecione o quadrado` e clique em `próximo` (Figura 42).

Figura 42: Quadrado selecionado para adicionar uma ação.

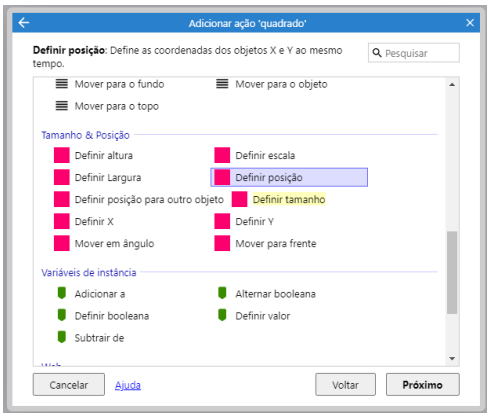


Fonte: Elaboração própria.

Em seguida será exibida uma tela com várias possibilidades de ação para o objeto quadrado. Existem dezenas de ações possíveis, como alterar cor, alterar ângulo, alterar valores de variáveis relacionadas a esse objeto... Porém o que queremos é alterar a posição

do objeto quadrado. Na seção `tamanho & posição`, procure por `definir posição` e clique em `próximo` (Figura 43).

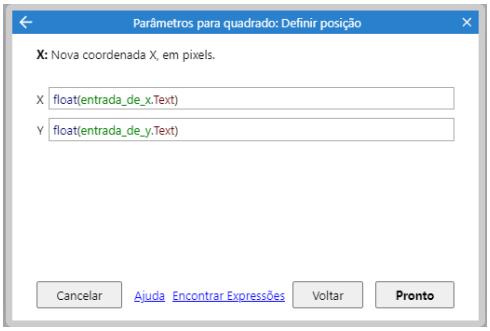
Figura 43: Ação de definir posição.



Fonte: Elaboração própria.

Queremos que o quadrado seja movimentado para os locais fornecidos pelo usuário nos campos de entrada para xe Y. Para fazer isso, em X, digite `float(entrada_de_x.Text)` e em Y digite `float(entrada_de_y.Text)` e em seguida clique em `pronto` (Figura 44).

Figura 44: Parâmetros da nova posição do objeto quadrado.



Fonte: Elaboração própria.

5.4 Sintaxe float e .Text

Antes de avançarmos, vamos entender de forma mais detalhada o que está acontecendo com essa instrução. Nosso objetivo é capturar os valores que o usuário digita nos campos de entrada. Vamos analisar o valor de X, mas a explicação vale da mesma forma para o valor de Y.

Primeiro, precisamos dizer ao Construct onde buscar o valor digitado. No nosso projeto, o valor de X está armazenado em um campo de entrada que chamamos de `entrada_de_x`. Em seguida, precisamos informar o que exatamente queremos desse campo de entrada. No caso, queremos acessar o texto digitado pelo usuário na caixa de texto, por isso adicionamos `.Text` ao final de `entrada_de_x`. Isso diz ao Construct que ele deve olhar para o conteúdo (o texto) presente na caixa de entrada nomeada `entrada_de_x`.

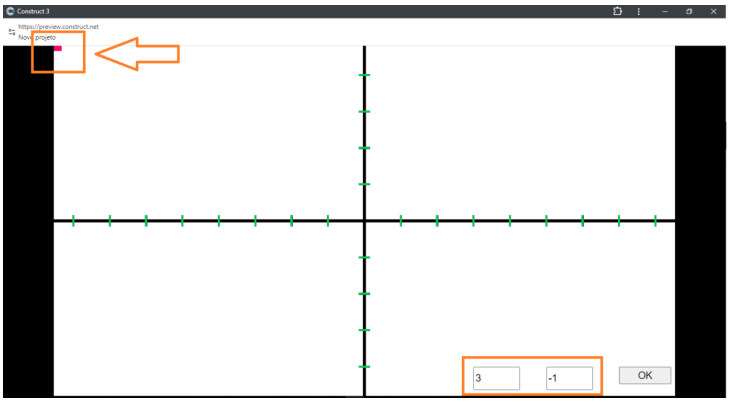
Quando o valor é capturado, ele é tratado como uma `string`, que é uma sequência de caracteres que pode conter letras, números e símbolos, mas sem interpretação numérica direta. Para realizar operações matemáticas com esse valor, como calcular a posição do quadrado, é necessário convertê-lo em um número. Nesse caso, o tipo de dado adequado para representar números com casas decimais é o `float` (número de ponto flutuante), que permite a manipulação precisa de números reais.

A conversão é feita de maneira simples: usamos `float()`, e dentro dos parênteses colocamos o valor digitado pelo jogador, que pode ser acessado pelo Construct com a instrução `entrada_de_x.Text`. Isso indica ao Construct que ele deve transformar o texto capturado na caixa de entrada em um número.

5.5 Inserção de Coordenadas

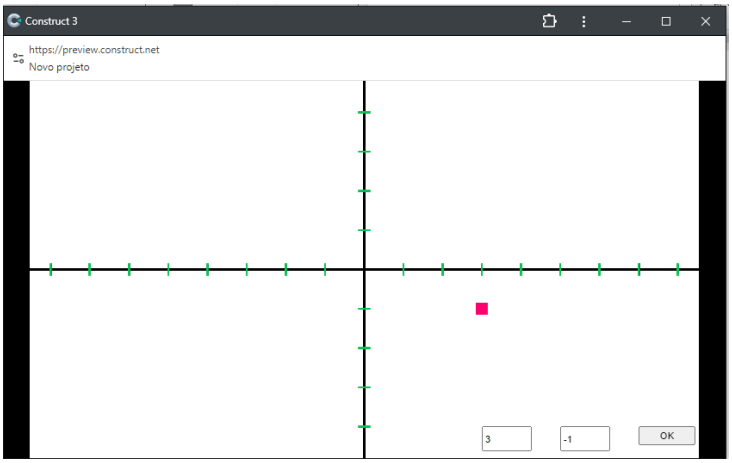
Ao testarmos o jogo, podemos inserir valores para X e Y. Tomemos como exemplo os valores 3 para X e -1 para Y. Ao clicarmos em `ok`, observamos que o quadrado foi de fato deslocado, isso significa que a ação de modificar a posição do quadrado ao clicar no `botão ok` está funcionado. Porém o resultado exibido é diferente do resultado esperado. Note que o quadrado aparece de forma "tímida" no canto superior esquerdo da tela do jogo (Figura 45). Em relação ao plano cartesiano, o nosso quadrado saiu da origem e foi para o 2^{o} quadrante do plano, porém sabemos que $(3, -1)$ é um ponto que pertence ao 4^{o} quadrante do plano cartesiano (Figura 46). Isso está acontecendo por conta que o construct está usando o seus sistema de coordenadas nativo, conforme abordado no tópico sobre sistema de coordenadas no construct. Logo é necessário fazer alguns ajustes na folha de eventos para que o nosso quadrado seja enviado para o local correto do nosso plano cartesiano.

Figura 45: Resultado obtido ao inserir as coordenadas $(3, -1)$ e clicar em `ok`.



Fonte: Elaboração própria.

Figura 46: Resultado esperado ao adicionar as coordenadas $(3, -1)$ e clicar em `ok`. Quadrado no 4^{o} quadrante nas coordenadas corretas.



Fonte: Elaboração própria.

Capítulo 6

Adaptação de Coordenadas ao Sistema Construct

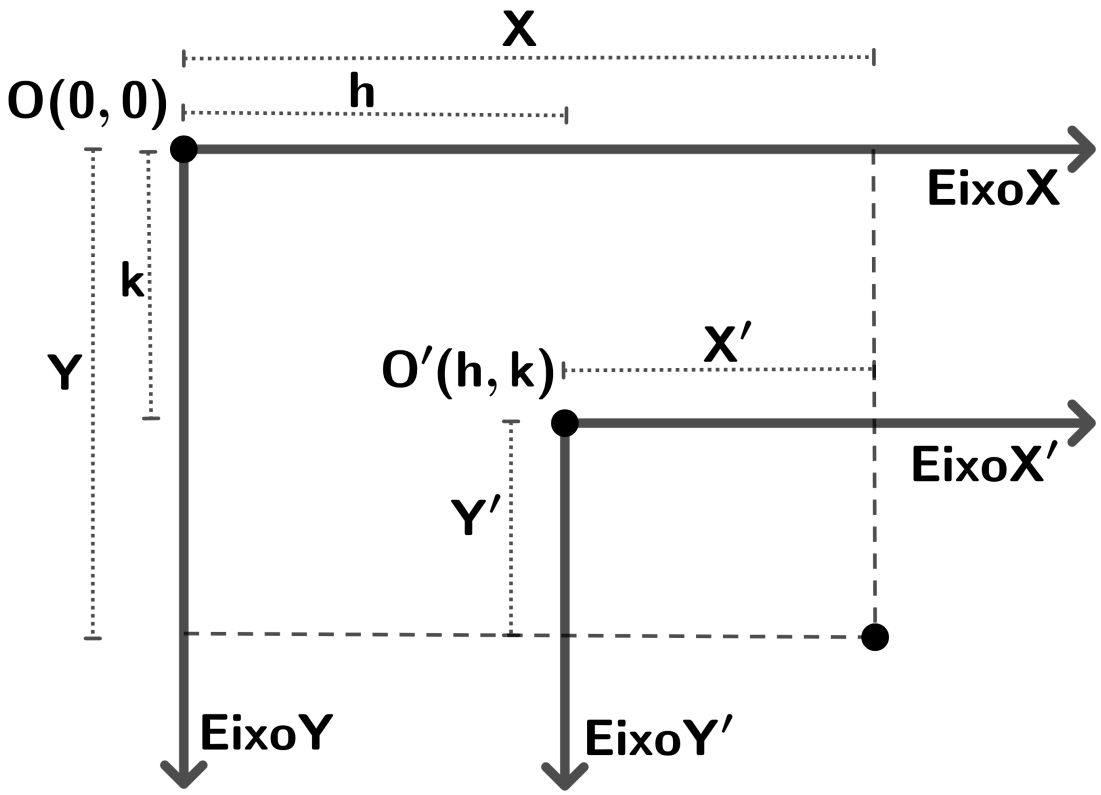
Para assegurar que as coordenadas inseridas no campo de entrada correspondam às coordenadas do plano cartesiano que estamos construindo, é necessário adaptar o nosso plano cartesiano ao plano do Construct. Para isso, são necessários os seguintes ajustes:

1. Deslocar a origem;
2. Ajustar a escala para que corresponda às unidades utilizadas no plano cartesiano;
3. Inverter o sentido de crescimento do eixo Y.

6.1 Deslocando a origem

Começaremos deslocando a origem. Precisamos que os valores inseridos pelo jogador, referentes ao plano cartesiano que construímos, sejam transportados para o plano cartesiano adotado pelo Construct. Matematicamente, esse deslocamento da origem é um processo de translação (mais detalhes na seção 6.5). Para tanto, tomemos como base a origem do nosso plano cartesiano em relação ao plano do Construct, que chamaremos de $O'(h, k)$, sendo h e k as coordenadas do ponto O' em relação ao plano do Construct (Figura 47).

Figura 47: Sistemas de coordenadas para deslocamento da origem.



Fonte: Elaboração própria.

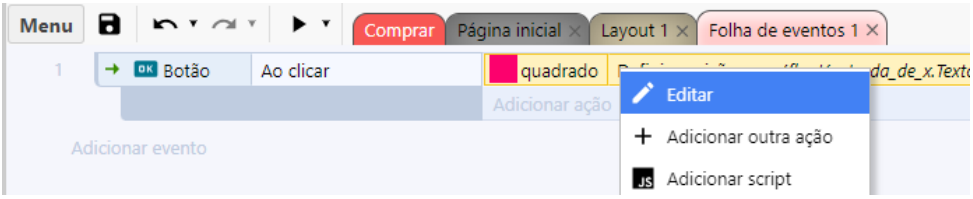
Vamos introduzir um novo sistema $X'O'Y'$ de tal forma que os eixos $O'X'$ e $O'Y'$ tenham a mesma unidade de medida, direção e sentido dos eixos OX e OY adotados pelo Construct. Portanto, cada ponto P no plano terá duas representações: $P(x, y)$ no sistema XY (sistema de coordenadas do Construct) e $P(x', y')$ no sistema $X'O'Y'$ (sistema de coordenadas do nosso plano cartesiano).

Note que, em relação ao plano do Construct, o ponto P tem coordenadas (x, y). Mas $x = x' + h$ e $y = y' + k$. Portanto, $P(x, y) = P(x' + h, y' + k)$. Com isso, conseguimos resolver o problema referente ao deslocamento da origem, pois x' e y' serão os valores fornecidos pelo jogador e h e k são conhecidos, mais precisamente são as coordenadas da origem do nosso plano cartesiano desenhado dentro do Construct.

Os valores de h e k podem ser observados consultando as coordenadas do objeto quadrado, que está exatamente no centro do nosso plano cartesiano. Pela janela de propriedades, as coordenadas do quadrado são 427 e 240. Ou seja, nossa origem tem coordenadas $O'(427, 240)$, onde $h = 427$ e $k = 240$. Em outras palavras, precisamos adicionar 427 pixels ao valor de x fornecido pelo jogador e 240 pixels ao valor de y fornecido pelo jogador.

Agora, precisamos retornar à folha de eventos para resolver o problema da origem que acabamos de discutir. Na folha de eventos, clique com o `botão direito` sobre o `quadrado`. Uma janela com várias opções será exibida. Clique em `editar` (Figura 48).

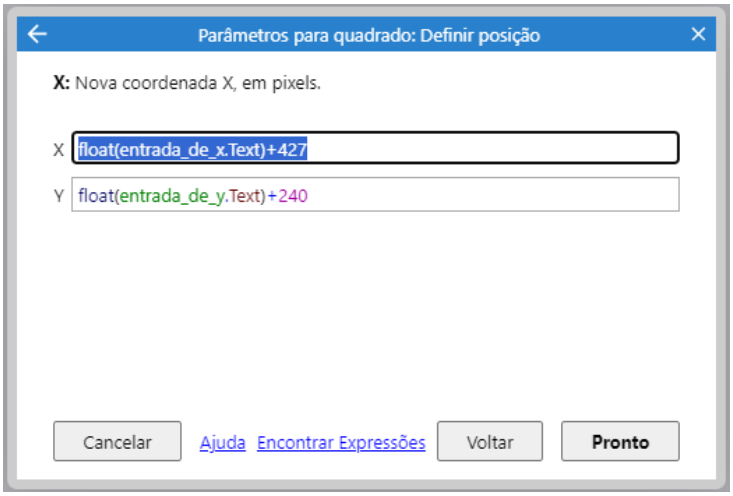
Figura 48: Editando as ações de posição do quadrado.



Fonte: Elaboração própria.

Isso abrirá novamente a janela para definição de parâmetros. Em X, adicione `+ 427`, ficando `float(entrada_de_x.Text) + 427`. Em Y, adicione `+ 240`, deixando-o da seguinte forma `float(entrada_de_y.Text) + 240`. Clique em `pronto` (Figura 49).

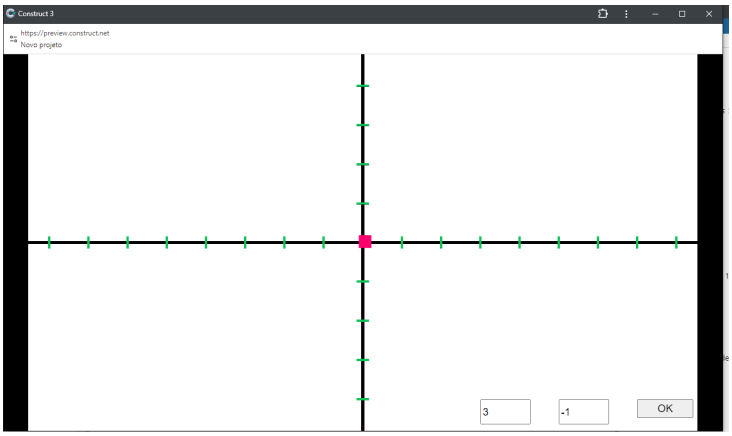
Figura 49: Ajuste de origem na folha de eventos.



Fonte: Elaboração própria.

Ao executar o jogo novamente, já é notável que o quadrado encontra-se sobre a origem do nosso plano cartesiano. Porém ao testar as coordenadas $(3, -1)$, observa-se que o quadrado não estará mais posicionado no canto superior da tela, mas sim muito próximo à origem do plano cartesiano (Figura 50). O deslocamento feito pelo quadrado é quase imperceptível, uma vez que o Construct ainda está utilizando o `pixel` como unidade de medida padrão. Como o `pixel` é uma unidade extremamente pequena, esse movimento sutil dificilmente pode ser detectado a olho nu.

Figura 50: Execução após o deslocamento da origem com teste de coordenadas $(3, -1)$.



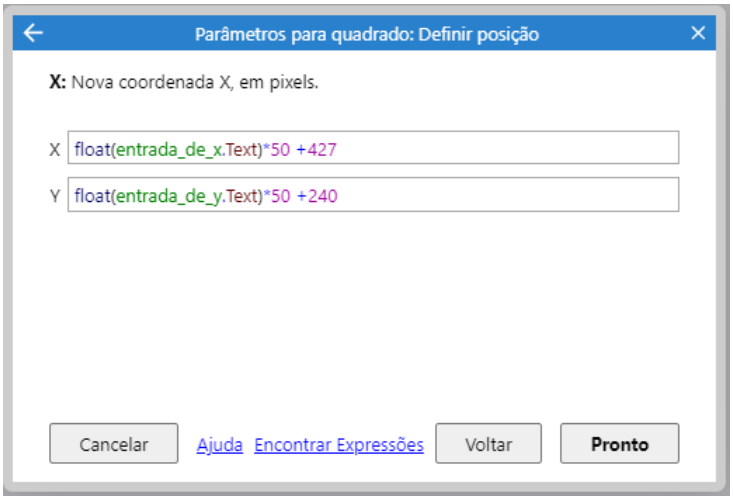
Fonte: Elaboração própria.

6.2 Ajuste de escala

Para solucionar o problema relacionado à unidade utilizada no plano cartesiano, é necessário relembrar o critério estabelecido no início do projeto, no qual uma unidade do plano cartesiano corresponde a `50 pixels`. Assim, ao receber os valores inseridos pelo usuário nos campos de texto que representam as coordenadas X e Y, devemos multiplicá-los por 50. Isso ocorre porque, se o usuário inserir o número 4, por exemplo, o quadrado se deslocaria apenas `4 pixels`. No entanto, para que o deslocamento seja coerente com a escala definida, é necessário multiplicar esse valor por 50, o que resultaria em um deslocamento de `200 pixels` ($4 \times 50 = 200$), garantindo a movimentação correta no ambiente gráfico. Em termos matemáticos, esse processo de ajuste de escala é uma semelhança (mais detalhes na seção 6.5). Para corrigir essa questão, altere os valores de X e Y para que X seja igual a `float(entrada_de_x.Text)*50 +427` e Y seja igual a `float(entrada_de_y.Text)*50 +240` (Figura 51).

Utilizamos o sinal de asterísco $(*)$ no Construct para realizar operações de multiplicação, seguindo a mesma ordem de precedência das operações matemáticas tradicionais. Para ilustrar, considere o exemplo `float(entrada_de_x.Text)*50 +427`. Aqui, há duas operações envolvidas: o valor inserido pelo usuário em `entrada_de_x.Text` é convertido para um número decimal `float` e, em seguida, multiplicado por `50`. O resultado dessa multiplicação é então somado a `427`. Esse valor final, expresso em `pixels`, será usado pelo Construct para deslocar o quadrado ao longo do eixo X. O mesmo procedimento é aplicado para o eixo Y.

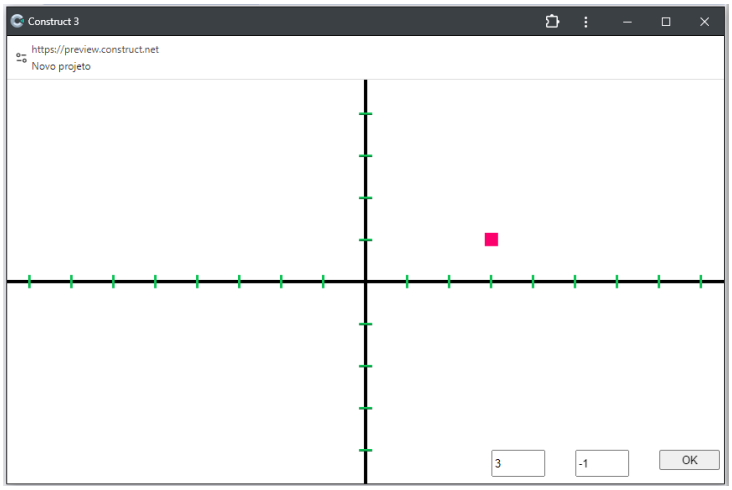
Figura 51: Ajuste da escala na folha de eventos.



Fonte: Elaboração própria.

Podemos executar novamente o jogo e observar o comportamento do quadrado no plano cartesiano. Testaremos mais uma vez as entradas `3` e `-1`. Agora, o quadrado percorre 3 unidades para a direita no eixo X, e no eixo Y, 1 unidade para cima (Figura 52). O deslocamento no eixo X está correto e o deslocamento no eixo Y foi realizado corretamente em termos de escala, mas no sentido oposto.

Figura 52: Execução do jogo após o ajuste de escala com teste de coordenadas $(3, -1)$.



Fonte: Elaboração própria.

6.3 Invertendo o sentido do eixo Y

A inversão do eixo Y é realizada multiplicando o valor de entrada de Y por -1, o que inverte o movimento vertical, permitindo que valores positivos movam o objeto para baixo e valores negativos para cima. No entanto, é importante manter o deslocamento da origem inalterado. Matematicamente esse processo é uma reflexão (mais detalhes na seção 6.5). Assim, a fórmula correta inverte apenas o valor de entrada de Y, sem alterar o valor fixo do deslocamento da origem. Para isso, implementamos a fórmula:

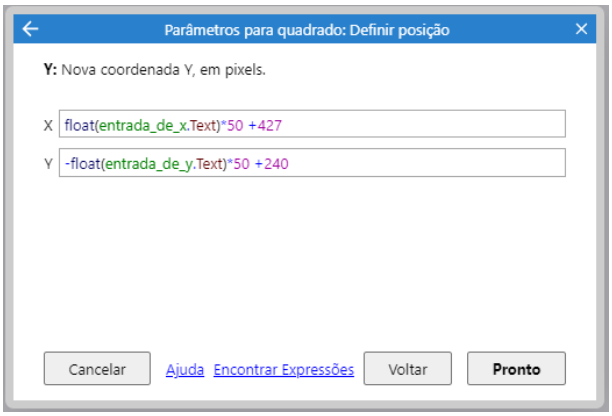
$$-\text{float}(\text{entrada\_de\_y.Text}) * 50 + 240$$

onde:

- Multiplicamos o valor de entrada de Y por -1 para inverter o eixo;
- Aplicamos uma escala com o fator multiplicativo 50;
- Mantemos o valor de $+240$ para ajustar corretamente a origem no layout.

Dessa forma, conseguimos inverter o eixo Y sem afetar o posicionamento da origem. Vá até a folha de eventos e altere o campo de `Y` na janela de **Parâmetros para o quadrado: Definir posição**, adicionando o sinal de menos no início da formula, conforme exibido na Figura 53.

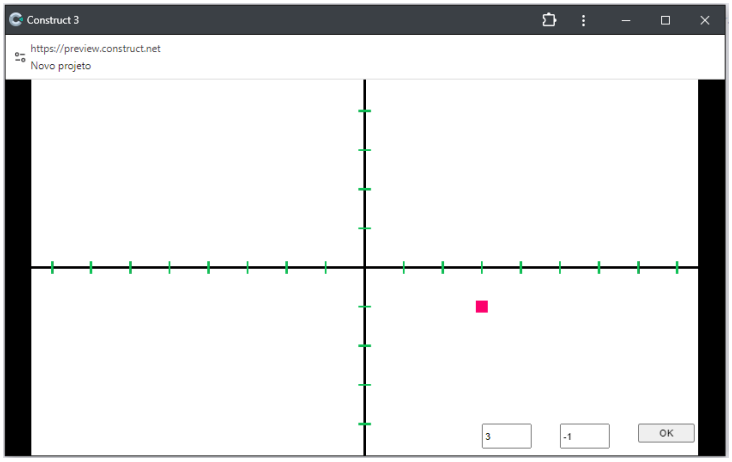
Figura 53: Ajuste do parâmetro de Y na folha de eventos.



Fonte: Elaboração própria.

Após realizar esse ajuste, podemos testar novamente o sistema inserindo as mesmas coordenadas $(3, -1)$. Como resultado, observaremos que o quadrado será corretamente posicionado no quarto quadrante, conforme indicado pelas coordenadas $(3, -1)$ no plano cartesiano da Figura 54.

Figura 54: Execução do jogo após todos os ajustes serem realizados.



Fonte: Elaboração própria.

6.4 Testes de Validação

Para garantir que todas as alterações realizadas nos parâmetros de posição do objeto quadrado estão corretas, é essencial realizar testes utilizando coordenadas distribuídas pelos quatro quadrantes do plano cartesiano: 1º, 2º, 3º e 4º quadrantes. Além disso, é recomendável testar situações em que $x = 0$ e/ou $y = 0$, verificando o comportamento do sistema ao longo dos eixos.

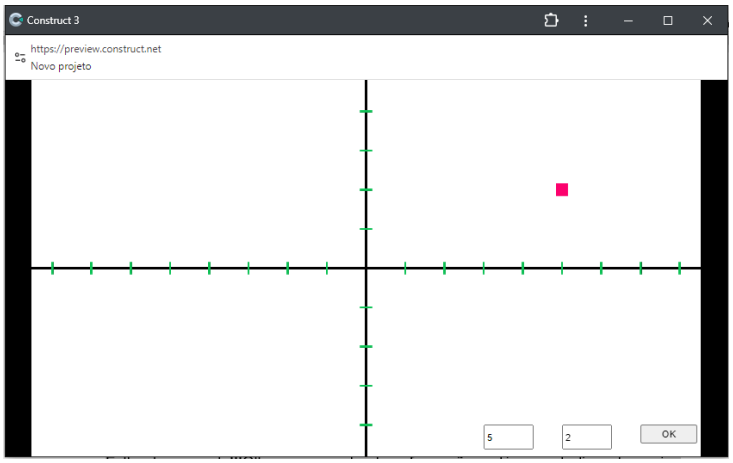
O quarto quadrante foi analisado na (Figura 54). Agora, consideremos o ponto de coordenadas $P(5, 2)$. Este ponto pertence ao primeiro quadrante, e o resultado está de acordo com o esperado, conforme demonstrado na (Figura 55). Mantendo $Y = 2$ e tomando $X = -5$, espera-se que o ponto pertença ao segundo quadrante, o que é confirmado pela (Figura 56).

Em seguida, mantendo $X = -5$ e ajustando $Y = -2$, devemos obter um ponto no terceiro quadrante, o que é validado na (Figura 57). Fixando a primeira coordenada em 0 e $Y = -2$, espera-se que o ponto esteja sobre o eixo Y, como ilustrado na (Figura 58). Quando a segunda coordenada é fixada em 0 e $X = 5$, o ponto deve situar-se sobre o eixo X, conforme mostrado na (Figura 59). Por fim, se $X = 0$ e $Y = 0$, o ponto deve coincidir com a origem do plano cartesiano, como indicado na (Figura 60). Esses testes abrangem algumas das possíveis combinações de coordenadas, o que assegura que o jogo esteja preparado para lidar com diferentes cenários e condições de execução. Dessa forma, garante-se uma implementação confiável, capaz de responder adequadamente a qualquer situação que possa ocorrer durante o funcionamento do jogo.

Além dos valores inteiros, também é possível utilizar valores não inteiros, como $1,5$ e $-0,77$. No entanto, é importante ressaltar que, para garantir o funcionamento adequado no ambiente do Construct 3, é necessário utilizar o ponto (.) ao invés da vírgula (,) para

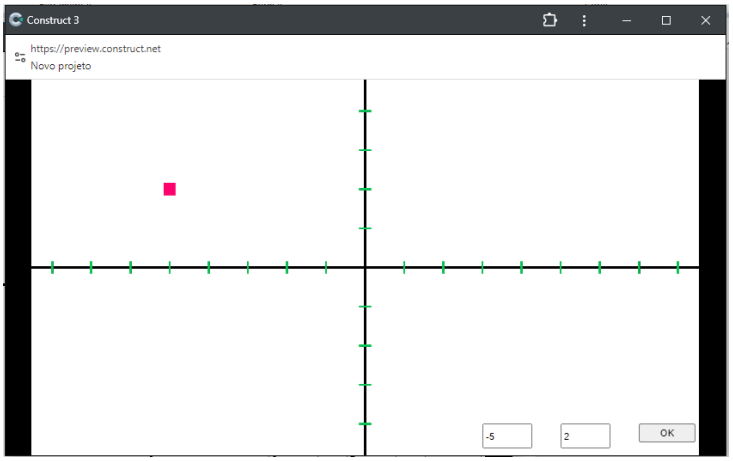
separar a parte inteira da parte decimal. Assim, valores como 1.5 e -0.77 devem ser inseridos dessa forma, permitindo que o jogo também manipule coordenadas e cálculos com precisão em números decimais.

Figura 55: Teste no primeiro quadrante com coordenadas $(5, 2)$.



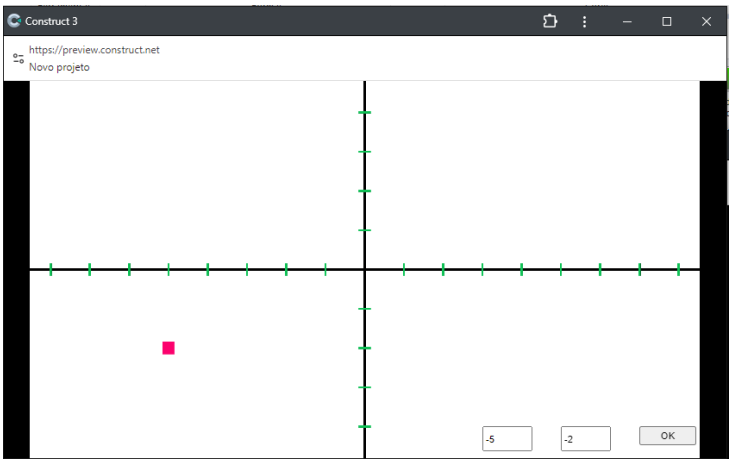
Fonte: Elaboração própria.

Figura 56: Teste no segundo quadrante com coordenadas $(-5, 2)$.



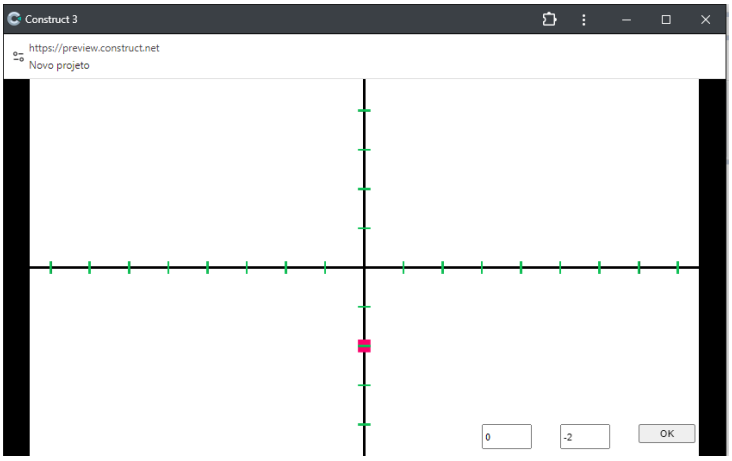
Fonte: Elaboração própria.

Figura 57: Teste no terceiro quadrante com coordenadas $(-5, -2)$.



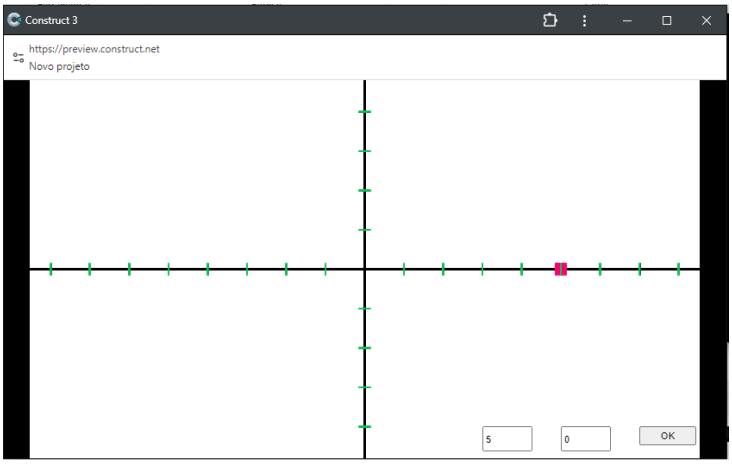
Fonte: Elaboração própria.

Figura 58: Teste com abscissa igual a zero.



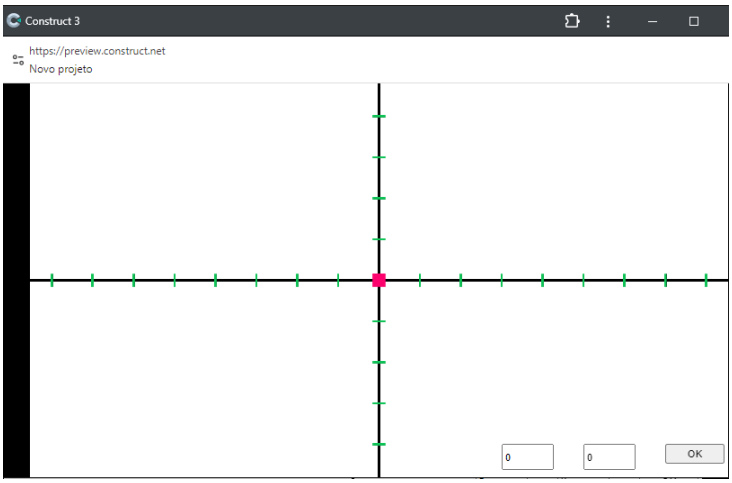
Fonte: Elaboração própria.

Figura 59: Teste com ordenada igual a zero.



Fonte: Elaboração própria.

Figura 60: Teste com ordenada e abscissa iguais a zero (origem).



Fonte: Elaboração própria.

6.5 Transformações no plano

Uma transformação no plano é uma maneira de mudar a posição de pontos no plano. Podemos pensar nela como uma "regra" que pega cada ponto do plano e o leva para um novo ponto. Por exemplo, imagine que você tem um ponto P em uma folha de papel. A transformação T vai pegar esse ponto P e transformá-lo em um novo ponto P_1, que chamamos de "imagem" de P pela transformação T.

Em termos mais precisos, Elon [13] define uma transformação T no plano Π como uma função $T : \Pi \to \Pi$, isto é, uma correspondência que associa a cada ponto P do plano outro ponto $P_1 = T(P)$ do plano, chamando sua imagem por T. Além disso, dadas duas transformações $S, T : \Pi \to \Pi$, a composta $S \circ T : \Pi \to \Pi$ é uma transformação que associa a cada ponto P do plano Π o ponto $S(T(P))$. Portanto, por definição, $(S \circ T)(P) = S(T(P))$. Ou seja, $S \circ T$ consiste em aplicar primeiro T e depois S.

Elon [13] também diz que uma vez que um sistema de coordenadas Π tenha sido estabelecido, uma transformação T pode ser descrita por suas equações, isto é, pelas expressões das coordenadas (x_1, y_1) do ponto $P_1 = T(P)$, obtido pela aplicação de T ao ponto $P = (x, y)$.

Na construção do nosso jogo, para adaptarmos o nosso plano cartesiano ao plano do Construct, foram necessárias três transformações, sendo definidas da seguinte forma:

- **Transformação E** para modificar a escala, sendo suas equações dadas por (Semelhança):
$$x_1 = 50x$$
$$y_1 = 50y$$

- **Transformação I** para inverter o eixo Y, sendo suas equações dadas por (Reflexão):
$$x_1 = x$$
$$y_1 = -y$$

- **Transformação D** para deslocar a origem, sendo suas equações dadas por (Translação):
$$x_1 = x + 427$$
$$y_1 = y + 240$$

Em seguida, fizemos uma composição de transformações, tomando $D(I(E(P)))$, onde $P = (x, y)$ é um ponto do nosso plano cartesiano. Isso nos diz que um ponto $P = (x, y)$ é transformado por E em $P_1 = (50x, 50y)$, que por sua vez é transformado por I em $P_2 = (50x, -50y)$ e que finalmente é transformado por D em $P_3 = (50x+427, -50y+240)$.

Recorde que esses foram exatamente os valores que inserimos na folha de eventos para definir a posição do objeto quadrado na Figura 53.

Capítulo 7

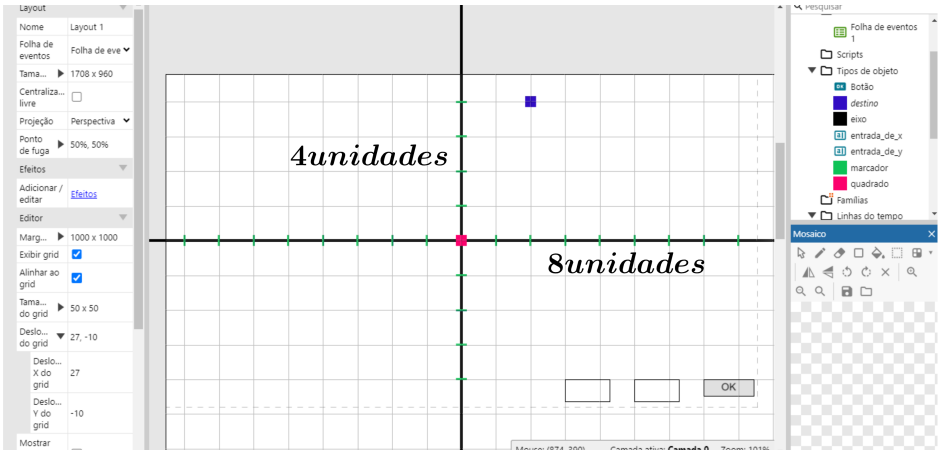
Ponto aleatório no plano cartesiano

Até o presente momento já construímos o plano cartesiano, inserimos o quadrado que representa o carteiro do jogo e que percorrerá o plano, criamos caixas de texto para a captura das coordenadas e implementamos um botão de confirmação. Resta, contudo, a adição do ponto aleatório que será o destino para o qual o carteiro deverá se deslocar no plano cartesiano e realizar a sua entrega.

Para inserir o ponto aleatório no plano cartesiano, começaremos criando um `novo objeto` do tipo `sprite`. Este objeto, denominado `destino`, terá a cor azul, 24 pixels de lado e opacidade de 60%. As escolhas para o tamanho e a opacidade visam proporcionar uma visualização mais clara, permitindo que o estudante perceba facilmente quando o carteiro se sobrepõe ao ponto de destino.

Sabemos que o plano cartesiano é ilimitado nos eixos X e Y. Porém estamos limitados ao que a tela do jogo pode exibir. Em nossa construção o eixo X abarca o intervalo real $(-8, 8)$ e o eixo Y o intervalo real $(-4, 4)$ (Figura 61).

Figura 61: Objeto destino adicionado ao layout.



Fonte: Elaboração própria.

Desejamos que o objeto destino apareça em coordenadas inteiras aleatórias dentro dos limites do plano cartesiano sempre que o jogo for iniciado. Nesse estágio de desenvolvimento do jogo, trabalharemos apenas com coordenadas do tipo (a, b), com $a, b \in \mathbb{Z}$. Caso o desenvolvedor queira adicionar coordenadas com abscissa ou ordenadas decimais, por exemplo, poderá fazê-lo. Porém, nesse momento, o nosso destino só poderá assumir coordenadas que pertençam ao produto cartesiano dos conjuntos $X = \{-8, -7, \ldots, 7, 8\}$ e $Y = \{-4, -3, \ldots, 3, 4\}$.

7.1 Produto cartesiano

Caso esteja trabalhando a construção do jogo em uma disciplina eletiva, esse momento torna-se oportuno para definir o produto cartesiano ao estudante. Essa definição é abordada no ensino médio em diversos livros. No livro [14], o produto cartesiano $X \times Y$ de dois conjuntos X e Y é o conjunto $X \times Y$ formado por todos os pares ordenados (x, y) cuja primeira coordenada x pertence a X e cuja segunda coordenada y pertence a Y. Simbolicamente:

$$X \times Y = \{(x, y); x \in X \text{ e } y \in Y\}$$

Além disso, se $X = \{x_1, x_2, \ldots, x_m\}$ e $Y = \{y_1, y_2, \ldots, y_p\}$ são conjuntos finitos com m e p elementos, respectivamente, então o produto cartesiano $X \times Y$ é finito e possui mp elementos. Podemos pensar nisso como uma matriz retangular com p colunas, onde cada uma possui m elementos.

$$\begin{pmatrix} (x_1, y_1) & (x_1, y_2) & \ldots & (x_1, y_p) \\ (x_2, y_1) & (x_2, y_2) & \ldots & (x_2, y_p) \\ \vdots & \vdots & \ddots & \vdots \\ (x_m, y_1) & (x_m, y_2) & \ldots & (x_m, y_p) \end{pmatrix}$$

Em nosso jogo, o conjunto $X = \{-8, -7, \ldots, 7, 8\}$ possui 17 elementos e o conjunto $Y = \{-4, -3, \ldots, 3, 4\}$ possui 9 elementos, totalizando $17 \times 9 = 153$ pares ordenados. Inserir todas essas possibilidades de pares ordenados para o nosso objeto destino de forma "manual" seria demasiadamente trabalhoso. Na seção 7.2 mostraremos como fazer isso de forma mais prática.

7.2 Comando choose (escolher)

A função `choose` é utilizada para selecionar aleatoriamente um dos parâmetros fornecidos como entrada. Esta função é flexível e permite a escolha entre números, strings ou uma combinação de ambos. A sintaxe básica da função é:

```
choose(a, b, c, ...)
```

onde `a`, `b`, `c`, etc. representam os parâmetros fornecidos. Ao chamar esta função, um dos valores passados será escolhido aleatoriamente e retornado como resultado. Por exemplo:

```
choose(1, 3, 9, 20)
```

poderia retornar qualquer um dos números 1, 3, 9 ou 20. De maneira semelhante, a função pode trabalhar com strings. Considere o seguinte exemplo:

```
choose("Pergunta 1", "Pergunta 2")
```

Neste caso, a função pode retornar `"Pergunta 1"` ou `"Pergunta 2"`. Esta funcionalidade é bastante útil em jogos para implementar decisões aleatórias, como a escolha de uma resposta do jogador ou a variação de elementos visuais ou sonoros em um ambiente de jogo. Para saber mais sobre isso, veja [5].

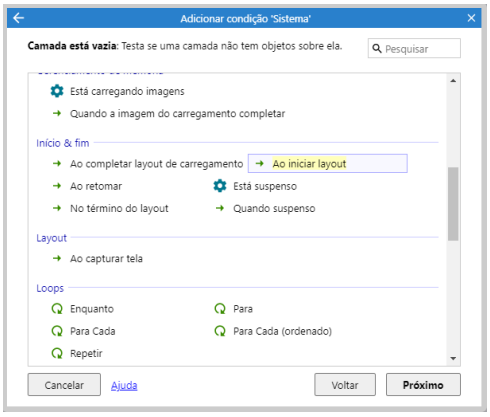
7.2.1 Implementando o comando Choose

Para evitarmos listar todos os pares ordenados, vamos fazer com que o Construct escolha um valor aleatório do conjunto X para ser nossa abscissa e um valor do conjunto Y para ser a ordenada.

Além disso, queremos que, quando o jogo for iniciado, nosso objeto destino já esteja posicionado em um dos 153 pares ordenados disponíveis de forma aleatória. A seguir, detalhamos o procedimento para alcançar esse objetivo.

Na `folha de eventos`, clique em `adicionar evento`, selecione `sistema` e escolha `ao iniciar layout` (Figura 62). Dentro deste evento, adicione a ação correspondente ao objeto `destino`. Vá até a seção `Tamanho & Posição` e selecione `definir posição`. Será exibida uma caixa para preenchimento das coordenadas X e Y em `pixel`.

Figura 62: Opção “Ao iniciar Layout”.



Fonte: Elaboração própria.

Conforme ilustrado na (Figura 63), em X escrevemos:

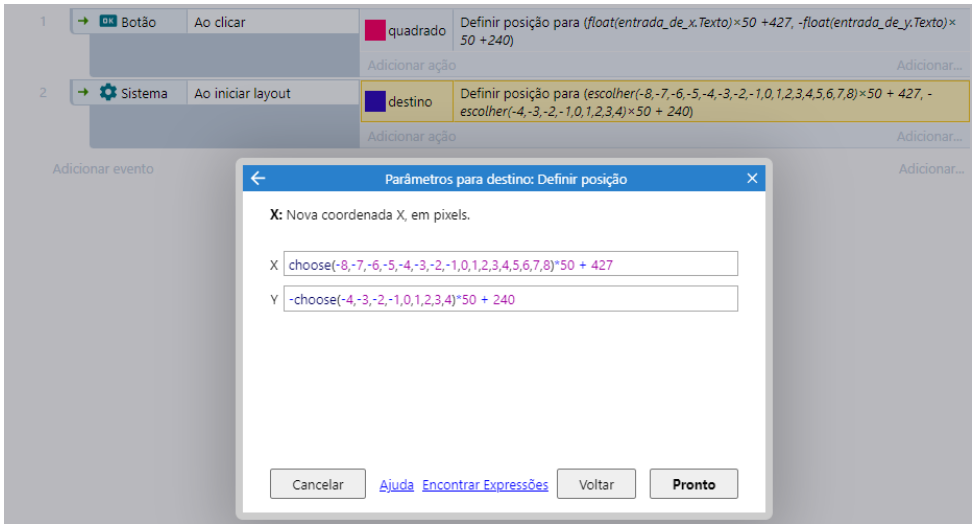
```
choose(-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8)*50 + 427
```

e em Y escrevemos:

```
-choose(-4,-3,-2,-1,0,1,2,3,4)*50 + 240
```

Note que essa sintaxe é muito parecida com o que já fizemos. Para determinar a abscissa, o Construct irá selecionar de forma aleatória um dos valores disponíveis do conjunto `choose` e, em seguida, aplicará as transformações que já foram detalhadas neste trabalho, na seção 6.5. O mesmo procedimento ocorre para definir o valor da ordenada. Veja que o comando `choose` nos dá a possibilidade de escolher os valores que desejarmos, permitindo que o desenvolvedor utilize os valores que quiser, inclusive números do conjunto $\mathbb{Q}$, $\mathbb{I}$ e $\mathbb{R}$. Por exemplo, para inserir o número real $\sqrt{2} \times \pi$, basta utilizar a expressão `sqrt(2) * pi`.

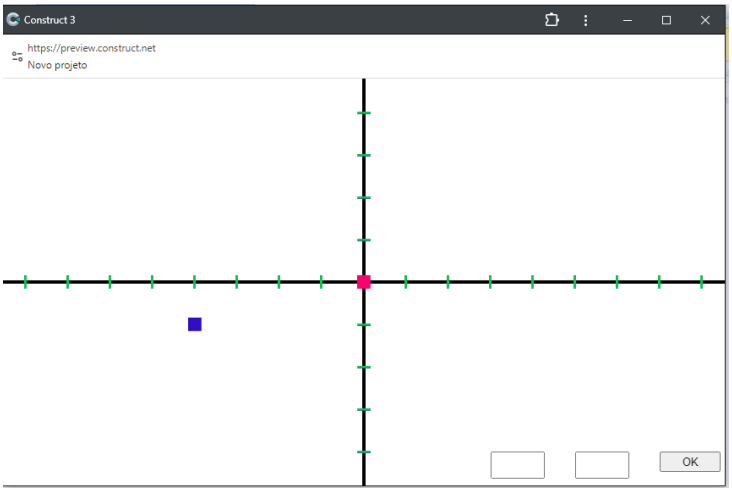
Figura 63: Parâmetros de posição do objeto destino.



Fonte: Elaboração própria.

Ao clicar no botão de `play` para executar o projeto, o objeto `destino` será exibido no plano cartesiano (Figura 64). Caso o jogo seja reiniciado, o objeto surgirá em um novo ponto aleatório, respeitando as coordenadas inteiras e os limites previamente definidos para os conjuntos X e Y. Dessa forma, a funcionalidade de geração de pontos aleatórios no plano cartesiano é completada, assegurando que cada execução do jogo apresente um novo desafio ao jogador.

Figura 64: Execução do jogo após programar o objeto destino.

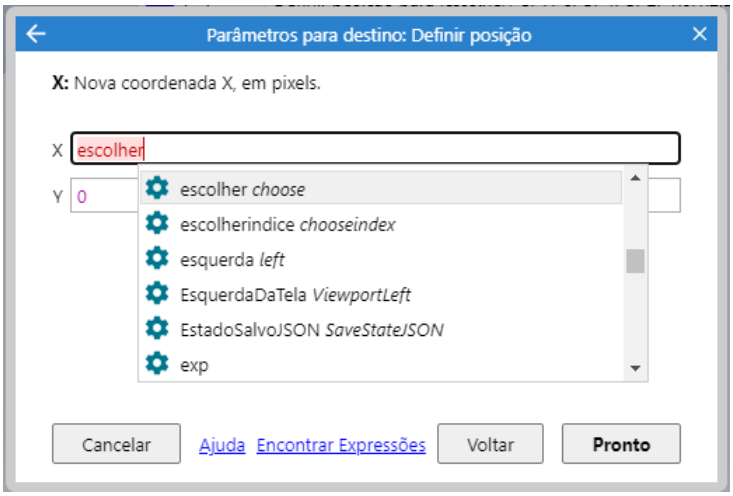


Fonte: Elaboração própria.

Caso o idioma do Construct esteja configurado para Português (BR), o comando

`choose` não será encontrado ao digitá-lo diretamente na janela de propriedades. Em vez disso, é necessário digitar `escolher` e pressionar a tecla `Enter`. O comando `choose` será inserido corretamente sem qualquer impedimento (Figura 65). Neste trabalho, utilizamos a versão em português do Construct. No entanto, é recomendável que, após a familiarização com os comandos básicos, o desenvolvedor opte pela versão em inglês, uma vez que a maioria dos materiais disponíveis em fóruns e tutoriais na internet está nesse idioma.

Figura 65: Comando escolher na versão em português.



Fonte: Elaboração própria.

Capítulo 8

Mecânica de colisões

No desenvolvimento de jogos, o conceito de colisões é fundamental para definir as interações entre objetos dentro de um ambiente virtual. Colisões ocorrem quando dois ou mais objetos se encontram no mesmo espaço no jogo, desencadeando uma série de eventos predefinidos. Esses eventos podem variar desde efeitos visuais, sons, mudanças no comportamento dos personagens ou até mesmo o fim do jogo, dependendo do contexto da colisão.

A detecção de colisões envolve algoritmos matemáticos que verificam se as áreas ocupadas por diferentes objetos estão se sobrepondo. No Construct 3, essa detecção é simplificada através de comportamentos e eventos que podem ser atribuídos a diferentes objetos, permitindo que desenvolvedores criem interações complexas sem precisar lidar diretamente com código ou fórmulas matemáticas avançadas.

8.1 Implementação de Mecânica de Colisão e Reposicionamento Aleatório

Com o ponto aleatório posicionado no plano cartesiano, podemos proceder para a implementação da mecânica de colisão. Esta mecânica será crucial para a interação do jogador com o jogo, uma vez que verificará se as coordenadas inseridas estão corretas, gerando novos desafios ao longo da jogabilidade. A implementação da mecânica de colisão segue as etapas descritas abaixo:

I. O jogador insere as coordenadas onde acredita que o `objeto destino` se encontra.

II. Ao clicar no `botão OK`, o quadrado desloca-se para as coordenadas inseridas pelo jogador.

III. Se as coordenadas inseridas estiverem corretas, o `quadrado` colidirá com o `destino`.

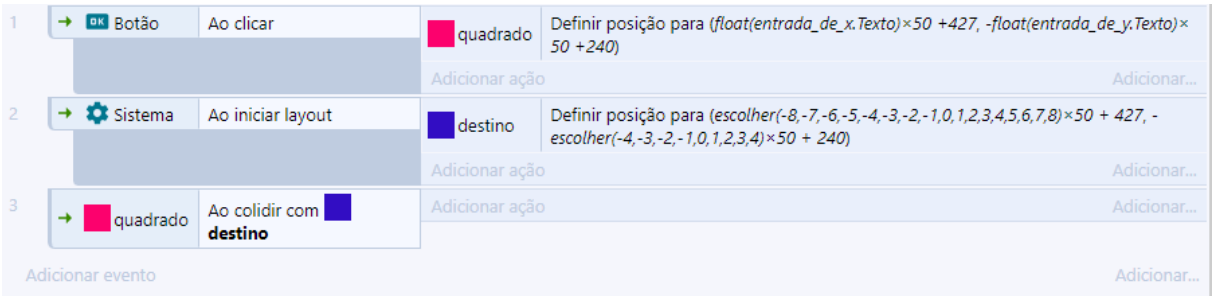
IV. Em caso de colisão:

- O `quadrado` retornará à sua posição de origem no plano cartesiano;
- O `destino` será reposicionado aleatoriamente em novas coordenadas no plano cartesiano.

V. Se as coordenadas inseridas estiverem incorretas, o `quadrado` mover-se-á para a posição informada, mas o `destino` permanecerá no local original, aguardando uma nova entrada do jogador.

Para implementar essa lógica de colisão no ambiente do Construct 3, deve-se proceder da seguinte forma: na `folha de eventos`, clique em `adicionar evento`, selecione o `objeto quadrado`. Em seguida, na seção de `colisões`, escolha a opção `ao colidir com outro objeto` e selecione o `objeto destino`. Finalize clicando em `pronto`. Até o momento, nossa folha de eventos contém três eventos (Figura 66), dos quais dois (evento 1 e evento 2) já possuem as ações definidas. O terceiro evento (evento 3) ainda está pendente de adição das ações necessárias.

Figura 66: Folha de eventos até o momento.



Fonte: Elaboração própria.

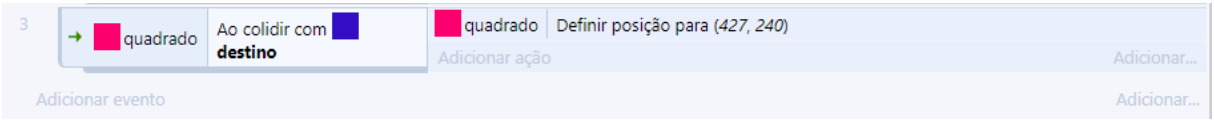
Queremos que o `objeto quadrado` retorne à posição de origem após colidir com o `objeto destino`. Para isso, dentro do evento de colisão que acabamos de criar, devemos seguir os seguintes passos:

1. Clique em `adicionar ação`.
2. Selecione o `objeto quadrado` na lista de objetos disponíveis.
3. Vá até a seção `Tamanho & Posição` e escolha a opção `Definir Posição`.
4. Nos campos correspondentes às coordenadas X e Y, insira as coordenadas adotadas para a origem do nosso plano cartesiano:

$$X = 427,$$
$$Y = 240.$$

Essas coordenadas garantem que o `objeto quadrado` será reposicionado na origem do nosso plano cartesiano após a colisão ser detectada. Ao término dessa ação, o evento número 3 passa a ter sua primeira ação definida, conforme exibido na Figura 67.

Figura 67: Ação de retornar o quadrado para a origem ao colidir com o destino.



Fonte: Elaboração própria.

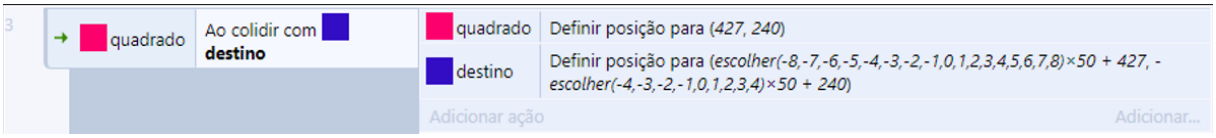
Isso completará a ação de retorno do `objeto quadrado` à origem no plano cartesiano. Além disso, após a colisão com o destino, queremos que este seja reposicionado em um novo par ordenado no plano cartesiano. Para isso, dentro do mesmo evento de colisão, devemos adicionar uma nova ação.

1. Clique novamente em `Adicionar ação` e selecione o objeto `destino`.
2. Navegue até a seção `Tamanho & Posição` e escolha a opção `Definir posição`.
3. Nos campos correspondentes às coordenadas X e Y, utilizaremos valores gerados aleatoriamente, respeitando as mesmas restrições impostas anteriormente (intervalos limitados de números inteiros). As fórmulas para determinar as novas coordenadas serão as seguintes:

$$X = \texttt{choose(-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8)} * 50 + 427,$$
$$Y = -\texttt{choose(-4,-3,-2,-1,0,1,2,3,4)} * 50 + 240.$$

Essas fórmulas garantem que o destino será reposicionado de maneira aleatória dentro do intervalo previamente definido. Ao término dessa ação, nosso evento número 3 deverá ter duas ações, conforme exibido na Figura 68.

Figura 68: Ações programadas no evento 3.



Fonte: Elaboração própria.

8.2 Testando e Melhorando a Mecânica de Colisão

Para finalizar esta etapa de desenvolvimento, é necessário testar a mecânica de colisão implementada no jogo. O processo de teste permite verificar se as interações entre quadrado e o destino estão funcionando conforme o esperado, além de identificar possíveis melhorias para otimizar a experiência do usuário.

Inicialmente, inserimos as coordenadas do destino nos campos preparados para receber os valores da abscissa X e da ordenada Y. Em seguida, clicamos no botão `OK` para que o quadrado se desloque até a posição correspondente às coordenadas inseridas.

Ao testar a jogabilidade, notamos que algumas pequenas melhorias podem ser realizadas para aprimorar a experiência visual e a fluidez das interações no jogo.

8.2.1 Sobreposição do Objeto Quadrado

Uma das melhorias envolve a visualização do quadrado. Queremos que o quadrado esteja sempre visível por cima de todos os outros objetos na tela, especialmente ao se posicionar corretamente sobre o ponto de destino. O objetivo é garantir que o quadrado sobreponha o ponto de destino quando as coordenadas estiverem corretas, facilitando a identificação da colisão visualmente.

Para alcançar esse efeito, devemos enviar o objeto quadrado para o topo da camada na qual ele está posicionado. Esse ajuste garante que ele sempre será desenhado por cima de todos os outros elementos da interface. (Veja a seção 4.3).

8.2.2 Melhoria na Visualização da Colisão

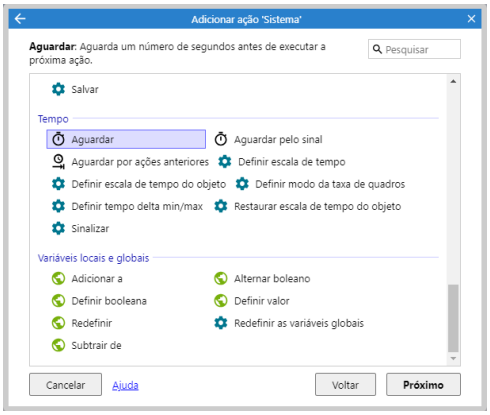
Outro ponto importante que identificamos durante os testes está relacionado à percepção visual da colisão entre o quadrado e o destino. Atualmente, a colisão não é claramente visível, pois o Construct executa de forma imediata as ações programadas e assim que detecta a colisão o quadrado é enviado de volta ao centro, e o destino é reposicionado, ambos de forma instantânea.

Para resolver esse problema e permitir que o jogador veja o quadrado sobre o destino antes que a posição de ambos seja alterada, precisamos adicionar um pequeno intervalo de tempo entre a detecção da colisão e a execução das ações subsequentes.

Para implementar essa melhoria, devemos modificar o evento de colisão entre o quadrado e o destino, seguindo os seguintes passos:

1. Dentro do evento de colisão, adicione uma `nova ação`.
2. Selecione o objeto `Sistema`.
3. Na seção `Tempo`, selecione a ação `Aguardar` (Figura 69).

Figura 69: Ação de aguardar.

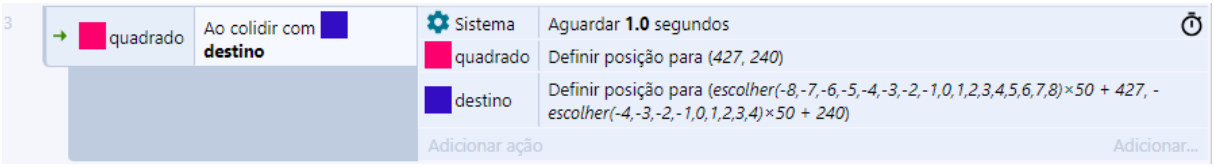


Fonte: Elaboração própria.

Essa ação, conforme descrito na interface do Construct, permite adicionar um atraso, em segundos, antes que a próxima ação seja executada. O objetivo aqui é dar tempo ao jogador para visualizar o quadrado corretamente posicionado sobre o ponto de destino, o que indicará que ele inseriu as coordenadas corretas.

A ação de aguardar deve preceder as mudanças de coordenadas do quadrado e do destino. Para isso, basta arrastar a ação de `Sistema` que acabamos de adicionar para o topo da lista de ações dentro do evento de colisão (Figura 70).

Figura 70: Ação de aguardar 1 segundo no topo do evento.



Fonte: Elaboração própria.

Note que o valor padrão para o tempo de espera foi definido como 1 segundo. Caso deseje alterar esse valor, basta dar um duplo clique na ação `Aguardar` e modificar o campo correspondente ao número de segundos. Execute novamente o jogo e veja a diferença.

Capítulo 9

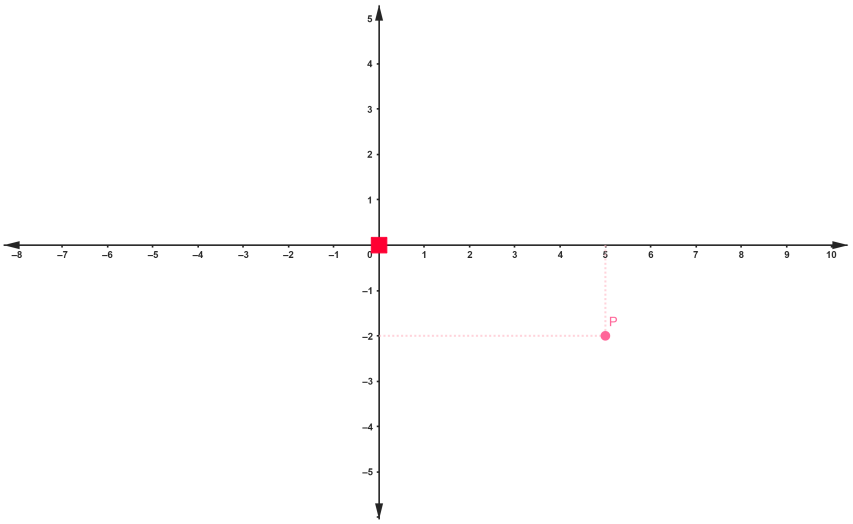
Aplicando o jogo do carteiro aventureiro

É importante ressaltar como o professor pode utilizar este jogo como uma ferramenta pedagógica em sala de aula para ensinar a localização de pontos no plano cartesiano. Uma abordagem didática consiste em fazer com que o aluno compreenda as coordenadas como instruções para guiar um objeto em um mapa, neste caso, um quadrado posicionado inicialmente na origem do plano cartesiano.

Considerando um ponto $P(a, b)$ no plano, isso implica que o objeto, inicialmente localizado na origem, deverá se deslocar a unidades no eixo X e, posteriormente, b unidades no eixo Y. Valores positivos de a ou b indicam que o deslocamento ocorrerá no sentido de crescimento dos eixos X ou Y, respectivamente, enquanto valores negativos indicam um movimento no sentido oposto. Em outras palavras, o quadrado, que parte do centro, precisará realizar dois movimentos para chegar ao ponto $P(a, b)$ no plano cartesiano. O primeiro movimento será para a esquerda (se $a < 0$) ou para a direita (se $a > 0$) em um total de $|a|$ unidades, e o segundo movimento será para cima (se $b > 0$) ou para baixo (se $b < 0$), em um total de $|b|$ unidades, onde $|\cdot|$ denota o valor absoluto de um número.

Tomemos, por exemplo, um ponto P localizado no plano, cujas coordenadas não são fornecidas de forma algébrica, e queremos que o estudante identifique essas coordenadas (Figura 71). Nesse caso, o estudante deverá usar as pistas visuais do jogo para deduzir o valor de a e b, verificando a posição relativa do ponto P em relação à origem, e aplicando o raciocínio de deslocamento em ambos os eixos.

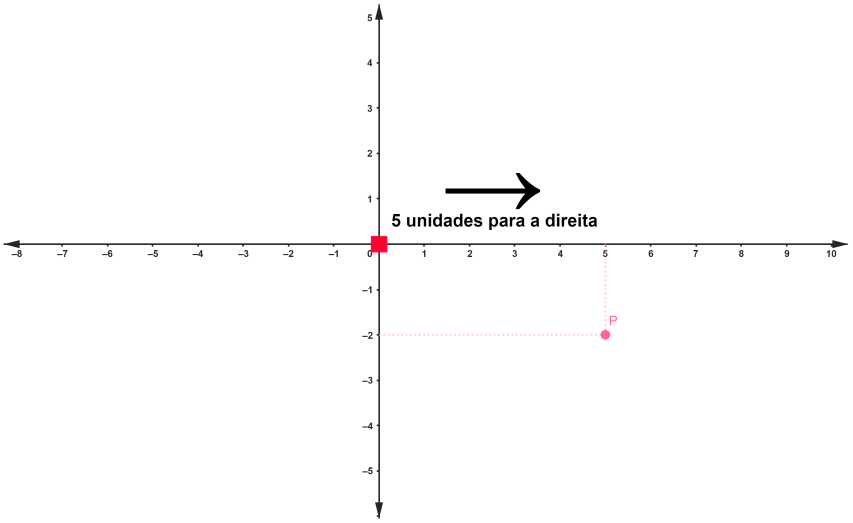
Figura 71: Ponto P no plano cartesiano.



Fonte: Elaboração própria.

Para chegarmos à localização do ponto P, devemos inicialmente mover o quadrado para a esquerda ou para a direita. No exemplo em questão, o quadrado se aproxima do ponto P ao mover-se para a direita (no mesmo sentido de crescimento do eixo X) em 5 unidades (Figura 72).

Figura 72: Encontrando o valor da abscissa do ponto P.

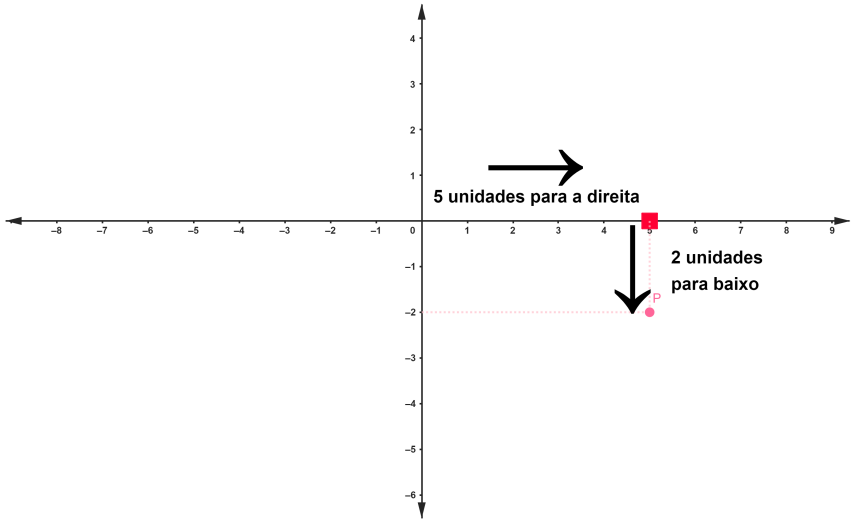


Fonte: Elaboração própria.

Após o primeiro movimento, o quadrado estará alinhado com o ponto P no eixo X. O próximo movimento será na direção do eixo Y, podendo ser para cima ou para baixo. No

exemplo, o quadrado deve descer 2 unidades, movendo-se no sentido oposto ao crescimento do eixo Y (Figura 73).

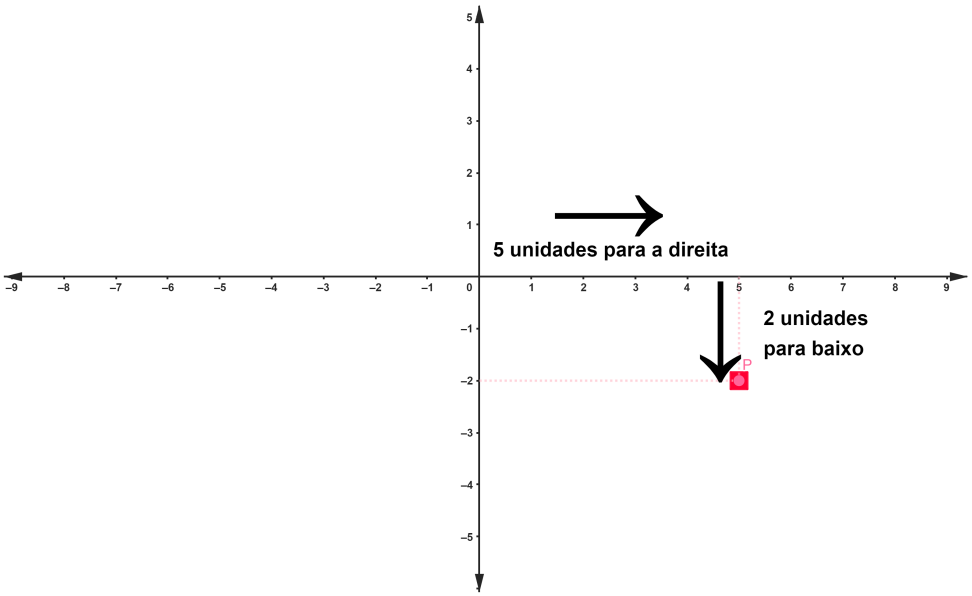
Figura 73: Definindo a ordenada do ponto P.



Fonte: Elaboração própria.

Assim, o quadrado chegará ao ponto P, com um deslocamento de 5 unidades no sentido positivo do eixo X e depois 2 unidades no sentido negativo do eixo Y (Figura 74). Algebricamente, podemos então determinar as coordenadas do ponto P como $P(5, -2)$.

Figura 74: Quadrado no ponto P(5,-2).

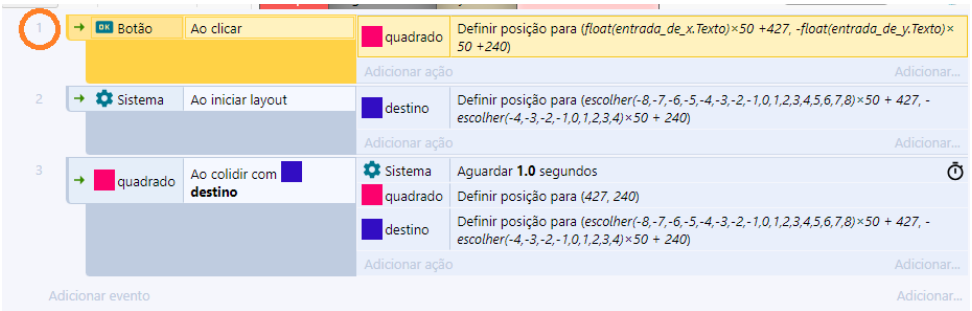


Fonte: Elaboração própria.

Podemos implementar o movimento em dois passos do carteiro em nosso jogo. Em vez de fazer com que o carteiro se desloque diretamente até o destino, podemos programar o movimento de forma que, primeiro, ele se mova ao longo do eixo X e, em seguida, se desloque na direção do eixo Y. Dessa forma, o estudante poderá visualizar claramente como são obtidos os valores das coordenadas do destino.

Para criar esse efeito visual no Construct, precisamos realizar uma pequena modificação na estrutura que já foi desenvolvida até o momento.

Figura 75: Folha de eventos do nosso projeto atual, com destaque para o evento 1.

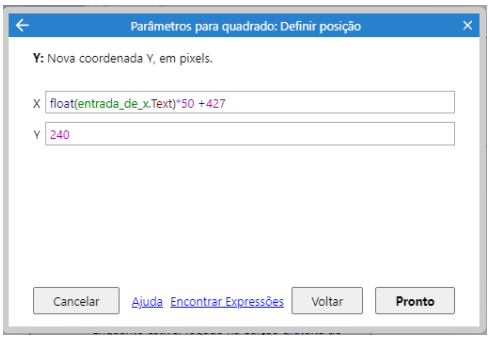


Fonte: Elaboração própria.

Vamos modificar as ações do `Evento 1` (`botão, ao clicar`) (Figura 75). Clique duas vezes sobre a ação do quadrado para editá-la (ou clique com o botão direito e selecione

`editar`). Mantenha os parâmetros do eixo X como estão e modifique o valor do eixo Y para 240. Isso garantirá que o quadrado se desloque inicialmente apenas ao longo do eixo X. Vale lembrar que a origem do nosso plano cartesiano é o ponto $(427, 240)$, e por isso, o valor de Y deve ser mantido em 240 (Figura 76).

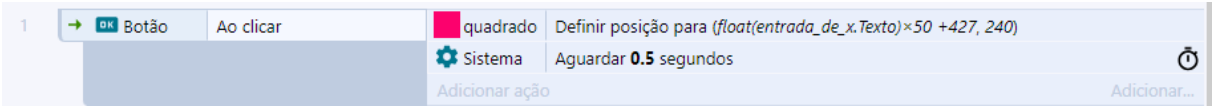
Figura 76: Modificando os parâmetros do quadrado.



Fonte: Elaboração própria.

Em seguida, precisamos adicionar um intervalo de tempo para que o quadrado fique parado sobre o eixo X antes de realizar o movimento no eixo Y. Adicione um tempo de 0,5 segundos para que a próxima ação seja executada. Lembre-se de que o Construct utiliza ponto (`.`) para separar a parte inteira da decimal. Assim, para adicionar números racionais, utilize o ponto decimal. Nesse caso, o valor adicionado será `0.5` (Figura 77).

Figura 77: Adição do tempo de 0,5 segundos para iniciar a próxima ação.



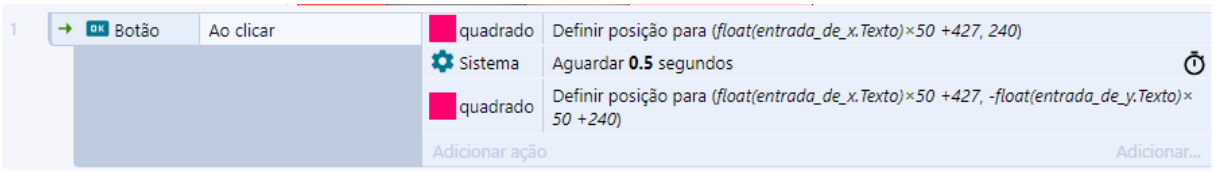
Fonte: Elaboração própria.

Após o quadrado se deslocar ao longo do eixo X e aguardar 0,5 segundos, o próximo passo é fazer com que ele se mova Y unidades na direção do eixo Y (Figura 78). Para isso, adicionaremos uma nova ação, de modo que o quadrado seja movido para as coordenadas finais dadas por:

$$X = \texttt{float(entrada\_de\_x.Text)} * 50 + 427$$

$$Y = -\texttt{float(entrada\_de\_y.Text)} * 50 + 240$$

Figura 78: Evento 1 após realizar as modificações.



Fonte: Elaboração própria.

Com essas alterações, finalizamos a estrutura básica do nosso primeiro jogo. Dessa forma, garantimos que o jogador possa praticar e aprimorar suas habilidades na localização de pontos no plano cartesiano por meio de uma mecânica de jogo interativa.

9.1 Sugestão de aplicação 1

Aplicar o jogo do Carteiro Aventureiro em sala de aula

O jogo do carteiro aventureiro pode ser uma maneira prática e divertida de ensinar a localização de pontos no plano cartesiano. Ele permite que os alunos aprendam de forma interativa, explorando conceitos matemáticos enquanto participam de uma atividade lúdica.

Uma boa forma de usar o jogo é projetá-lo em uma TV ou datashow para que toda a turma acompanhe. O professor pode começar explicando a ideia principal: o carteiro deve alcançar um destino marcado no plano, seguindo dois passos. Primeiro, ele se desloca no eixo X, indo para a esquerda ou para a direita. Depois, ajusta sua posição no eixo Y, subindo ou descendo. Essa mecânica simples ajuda os alunos a entenderem as coordenadas como direções claras.

Na prática, o professor pode pedir que os próprios alunos insiram coordenadas no jogo. À medida que o carteiro se move, todos acompanham os resultados na tela. Essa interação faz com que os estudantes percebam como os valores inseridos se traduzem em movimentos no plano, conectando teoria e prática de forma visual.

O jogo também pode ser usado como uma introdução ao plano cartesiano antes de explicações formais. Ele ajuda os estudantes a visualizar os conceitos de maneira prática e engajante, tornando mais fácil conectar a teoria à prática.

Para deixar a experiência mais dinâmica, o professor pode propor desafios como localizar pontos com coordenadas que incluam números racionais ou realizar tarefas dentro de um tempo limitado. Essas variações são simples de implementar e podem ser feitas pelo próprio professor no Construct 3, ajustando o jogo para atender às necessidades e características específicas de sua escola.

9.2 Sugestão de aplicação 2

Eletiva de Criação de Jogos Digitais com Construct 3

A oferta de uma disciplina eletiva baseada no Construct 3 pode abranger tanto a criação de jogos matemáticos quanto a produção de jogos em outros gêneros. Essa proposta reforça, primeiramente, a importância de replicar e adaptar os projetos descritos neste trabalho, valorizando os conceitos de plano cartesiano e função afim, que será abordada no próximo capítulo. Em segundo plano, mostra que as mesmas técnicas e princípios podem ser aproveitados em jogos de caráter mais amplo (aventura, ação, narrativa), mantendo vivo o estímulo ao pensamento lógico e às habilidades de programação.

A adoção dessa estratégia alinha-se tanto aos pressupostos de metodologias ativas [15, 18, 19] quanto ao perfil de nativos digitais [22], para quem abordagens mais práticas e dinâmicas tendem a ser mais atrativas. Além disso, o desenvolvimento de projetos que envolvem a autoria do aluno vai ao encontro da ideia de *learning by doing* [18, 19], proporcionando um ambiente em que o estudante constrói o conhecimento ao produzir algo significativo.

Assim, o professor tem duas frentes de aplicação principais:

1. Dar continuidade ao que foi exposto nesta dissertação, replicando ou reinventando os jogos matemáticos propostos para fixação de conteúdos do currículo de Matemática.

2. Explorar novos horizontes de criação lúdica, seja em jogos de aventura, narrativa ou temática livre, apoiando-se na mesma lógica de programação e na metodologia de construção de eventos descrita ao longo do texto.

1. Foco na Matemática: Jogos Replicados ou Inspirados neste Trabalho

Como ponto de partida, o professor pode propor aos alunos a recriação do Carteiro Aventureiro e do Jogo da Joaninha (este último será apresentado no Capítulo 10), conforme descritos nesta dissertação. Esses projetos fornecem exemplos consolidados de como o Construct 3 pode ser integrado ao ensino de conceitos como plano cartesiano, funções afim e transformações geométricas, em consonância com a Base Nacional Comum Curricular (BNCC) [3].

Uma vez compreendidos os princípios básicos, é possível expandir ou personalizar esses jogos, adicionando fases e desafios que reforcem outros tópicos matemáticos (geometria, estatística, álgebra etc.). O material deste trabalho, portanto, serve de referência direta para o professor, tanto no que diz respeito à lógica de eventos quanto aos elementos visuais e à aplicação dos conteúdos.

Ao explorar cada mecânica e cada trecho de código, estudantes e docentes mergulham em problemas matemáticos de forma prática, aliando lógica de programação à

consolidação de conceitos curriculares. Nesse sentido, a proposta fundamenta-se nas ideias construcionistas de Papert [18, 19], que valorizam a construção ativa do conhecimento ao criar artefatos (no caso, jogos digitais) que podem ser compartilhados e analisados em sala de aula.

2. Exploração de Jogos Não Necessariamente Matemáticos

Mesmo ao se afastar de um tema estritamente matemático, o professor pode recorrer aos exemplos e técnicas apresentados ao longo deste trabalho para orientar o desenvolvimento de jogos de aventura, ação ou *puzzle*. Itens como movimentação de *sprites*, criação de fases, colisões e pontuações são processos que exigem raciocínio lógico e algoritmos, aspectos essenciais para o pensamento computacional [19, 22], ainda que a aplicação direta de conceitos matemáticos não seja explícita.

A possibilidade de introduzir narrativas, elementos de Artes (ilustração de personagens, cenários), Língua Portuguesa (roteirização, criação de diálogos) e outras áreas reforça a dimensão multidisciplinar do Construct 3. Para isso, o professor pode indicar trechos específicos desta dissertação que exemplifiquem a lógica de programação e a estrutura de eventos, incentivando a busca por soluções criativas, em sintonia com propostas de ensino que integram diferentes componentes curriculares [3].

Ainda que o jogo não aborde conteúdo matemático de modo ostensivo, conceitos como variáveis, condições e funções (no sentido de programação) sustentam a construção de qualquer jogo. Dessa forma, a metodologia descrita neste trabalho mantém sua relevância, pois ilustra boas práticas de desenvolvimento que podem ser aplicadas em jogos de qualquer gênero.

Em ambas as frentes de atuação — tanto ao criar jogos matemáticos quanto ao explorar novos gêneros — o objetivo é fortalecer o raciocínio lógico-matemático, desenvolver a criatividade dos estudantes e incorporar novas tecnologias de forma significativa na prática docente.

9.3 Relato de Experiência

A aplicação deste projeto ocorreu na Escola Estadual da Paraíba – Escola Cidadã Integral e Técnica Cônego Francisco Gomes de Lima, localizada no Bairro Ernesto Geisel, em João Pessoa. A iniciativa foi desenvolvida no âmbito de uma disciplina eletiva, modalidade em que os estudantes escolhem, entre as propostas pedagógicas ofertadas pelos docentes, aquelas que mais se alinham com seus interesses. Participaram da atividade 35 estudantes, que se envolveram em um processo interdisciplinar focado no aprendizado de matemática por meio da criação de jogos digitais utilizando o Construct 3.

A motivação dos alunos foi um dos elementos mais marcantes do processo. A possibilidade de criar seus próprios jogos despertou curiosidade e protagonismo, resultando em engajamento significativo. Mesmo sem experiência prévia em programação, os

estudantes mergulharam na elaboração de narrativas, design de personagens e na programação de comandos básicos. A produção de arte própria para os jogos, por exemplo, incentivou a criatividade e proporcionou a aplicação prática de conceitos geométricos, como posicionamento de elementos na tela, escalas e simetria.

Um caso que ilustra a profundidade da aprendizagem ocorreu com um aluno que, embora tivesse habilidades em programação, deparou-se com uma diferença fundamental entre o plano cartesiano tradicional e o sistema do Construct 3. Apesar de dominar algoritmos, ele não compreendia por que a origem, que sempre estivera no canto superior esquerdo da tela, agora se encontrava no centro. Essa curiosidade tornou-se uma oportunidade valiosa para discutir transformações geométricas necessárias para transitar entre sistemas de coordenadas, conectando teoria abstrata à prática em software.

Embora os estudantes tivessem liberdade para criar jogos sem vinculá-los diretamente à matemática, diversos conceitos matemáticos foram abordados durante o desenvolvimento da disciplina eletiva. Entre eles, destacam-se:

- A periodicidade de funções trigonométricas, utilizada para posicionar plataformas com movimento de vai e vem;
- O conceito básico de funções, em que entradas determinam saídas específicas;
- Operações aritméticas aplicadas a situações diversas;
- O plano cartesiano e o produto cartesiano;
- O pensamento lógico como base para a resolução de problemas.

Além disso, contei com o apoio de professores de outras disciplinas. O docente de Língua Inglesa destacou a presença do idioma no universo da tecnologia, enquanto a professora de Língua Portuguesa conduziu uma oficina de produção textual, auxiliando os estudantes na criação de diálogos para os personagens.

Ao final da eletiva, os estudantes compartilharam seus jogos com alunos de outras escolas durante o evento anual "Se Liga Prota", iniciativa promovida por escolas estaduais da Paraíba para incentivar jovens concluintes do ensino fundamental a ingressarem no ensino médio da rede. A experiência foi um sucesso: a interação com os jogos cativou os visitantes e gerou interesse em ingressar na instituição.

Os jogos Carteiro Aventureiro e Jogo da Joaninha foram implementados em uma turma da mesma escola para coletar feedback e avaliar o engajamento. Durante a atividade, observei que, em questão de minutos, os alunos mergulharam na dinâmica lúdica, demonstrando não apenas entusiasmo, mas também compreensão ágil de conceitos como plano cartesiano e coeficientes de funções lineares. Enquanto a turma se agrupava em torno do computador, o clima colaborativo e a curiosidade coletiva evidenciaram o potencial da abordagem: cada movimento no jogo transformou-se em discussão ativa sobre matemática, com estudantes sugerindo valores e explicando aos colegas o raciocínio por trás de suas escolhas.

Capítulo 10

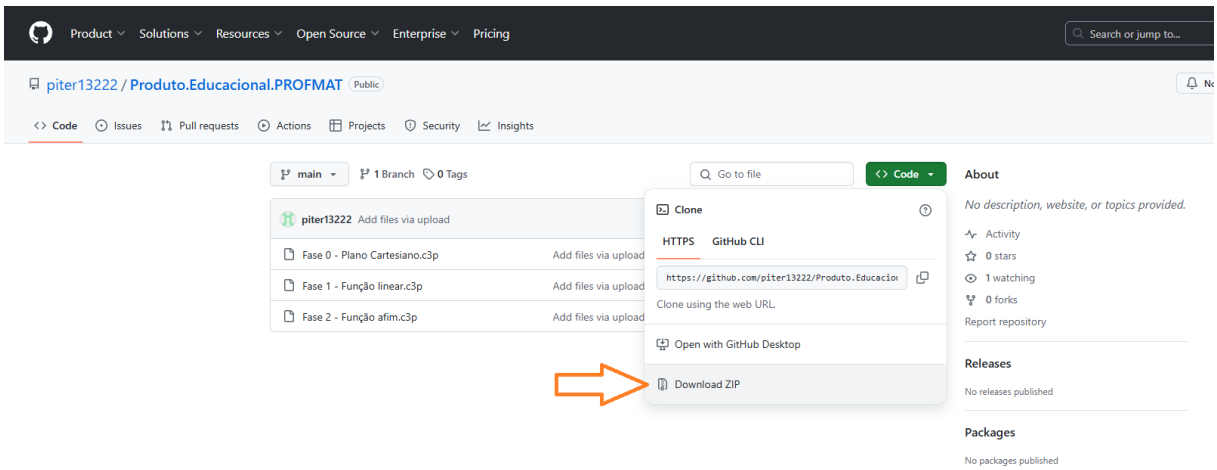
Jogo da Joaninha

Após concluir o jogo do carteiro, o leitor já adquiriu uma base sólida sobre a estrutura fundamental do Construct 3. Agora, daremos um passo adiante, explorando de forma mais aprofundada o potencial do Construct 3 na criação de jogos voltados ao ensino de matemática. Para que o texto seja mais objetivo, os próximos jogos serão apresentados com explicações mais concisas, mas sem perder o foco nos aspectos essenciais para sua compreensão e replicação. Para acessar o jogo da joaninha e também uma versão modificada do jogo do carteiro aventureiro, o leitor pode acessar o meu repositório do GitHub https://github.com/piter13222/Produto.Educacional.PROFMAT.

Para abrir e reproduzir os jogos, siga as instruções detalhadas abaixo:

1. **Baixe os arquivos do repositório no GitHub:**
    - Acesse o link do repositório fornecido acima.
    - Clique no botão verde `Code` e selecione `Download ZIP` para baixar todos os arquivos do projeto. (Figura 79)
    - Extraia o arquivo ZIP para uma pasta no seu computador.

Figura 79: Opção de Download ZIP no GitHub.

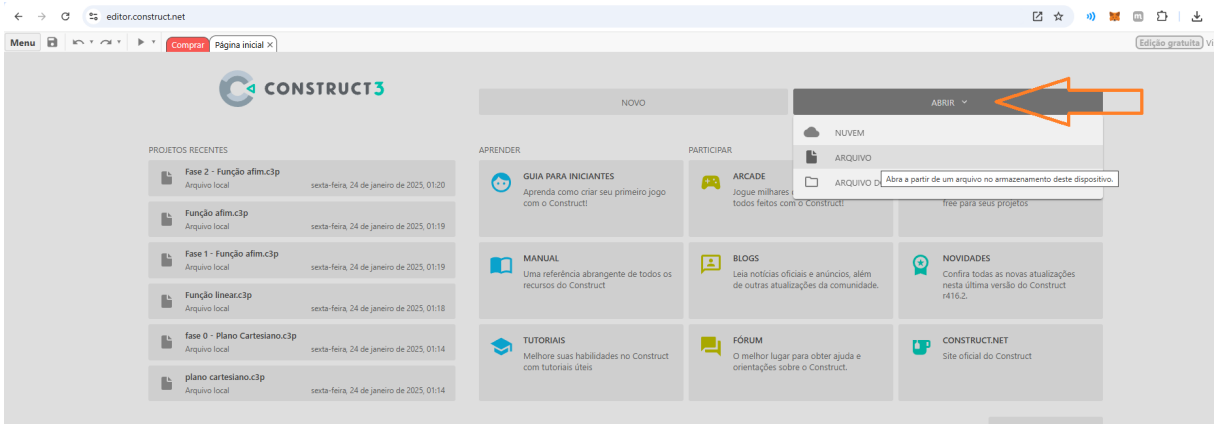


Fonte: Elaboração própria.

2. **Acesse o ambiente do Construct 3:**

   - Abra o navegador de sua preferência e acesse o site oficial do editor do Construct 3: https://editor.construct.net/.
   - No menu principal do Construct 3, clique em `Menu` no canto superior esquerdo.
   - Selecione a opção `Open Project` (Abrir Projeto).
   - Navegue até a pasta onde você extraiu os arquivos do repositório e selecione o arquivo com extensão `.c3p` (Construct 3 Project) referente a fase que você deseja carregar.

Figura 80: Opção de abrir arquivo no Construct 3.



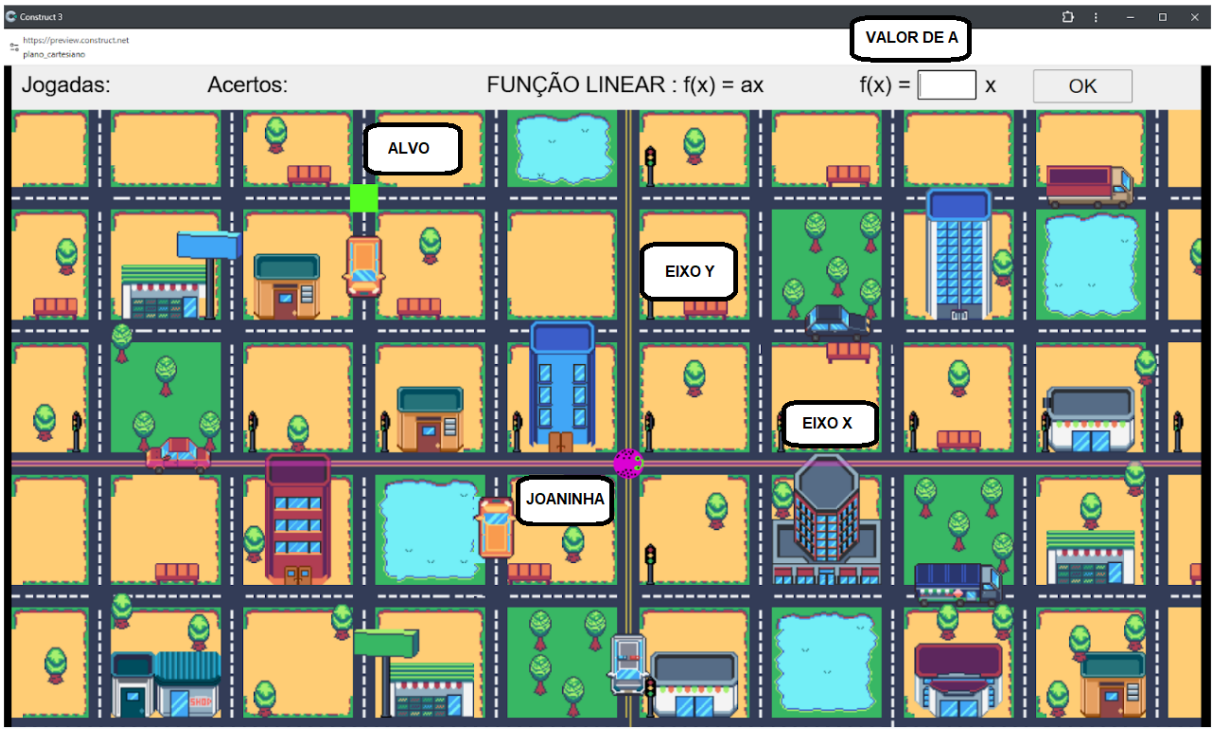
Fonte: Elaboração própria.

10.1 Fase 1: Função linear

O Jogo da Joaninha foi desenvolvido como uma ferramenta educativa para introduzir o estudo de funções lineares de forma intuitiva e interativa, antes de qualquer formalização teórica. O objetivo é que o jogador, ao explorar o plano cartesiano e conduzir a joaninha até o ponto-alvo, identifique padrões e relações entre os deslocamentos nos eixos X e Y. Essa experiência visa construir uma base perceptiva inicial, preparando o aluno para compreender os conceitos formais das funções, que serão aprofundados posteriormente em sala de aula.

O cenário do jogo é uma representação bidimensional de uma cidade, onde as avenidas e as ruas estão organizadas de maneira a formar um sistema de coordenadas cartesianas. As avenidas correspondem aos eixos principais (horizontal e vertical), enquanto as ruas, paralelas a esses eixos, marcam os números inteiros nos eixos X e Y. Cada quarteirão é delimitado por ruas espaçadas por uma unidade, de modo que as intersecções entre as ruas criam pontos no plano cartesiano (Figura 81). Por exemplo, o encontro da rua 1 paralela ao eixo X com a rua 1 paralela ao eixo Y corresponde ao ponto $(1, 1)$.

Figura 81: Interface do jogo da joaninha fase 1.



Fonte: Elaboração própria.

Nesta fase, a joaninha inicia sua jornada na origem $(0, 0)$ do plano cartesiano, representado pelo encontro das avenidas centrais, e deve alcançar um ponto-alvo marcado

no mapa, representado por um quadrado que está piscando. O voo realizado pela joaninha será por meio de uma linha reta entre a origem e o alvo. Podemos modelar esse percurso com uma função linear do tipo $f(x) = ax$, onde a é a taxa de variação da função.

Uma função $f : \mathbb{R} \to \mathbb{R}$ é chamada de função afim quando existem constantes $a, b \in \mathbb{R}$ tais que $f(x) = ax + b$ para todo $x \in \mathbb{R}$. No caso particular em que $b = 0$, a função é chamada de função linear. Segundo Elon [14], a função linear é frequentemente usada como modelo matemático para problemas de proporcionalidade. Além disso, como apontado por Elon [14], quando o coeficiente $a = 0$, ou seja, quando a função afim assume a forma $f(x) = b$ para todo $x \in \mathbb{R}$, estamos diante de um caso particular dessa função. Nesse caso, a função é chamada de função constante, pois o valor de $f(x)$ não depende de x e permanece sempre igual a b, independentemente do valor de x.

O valor do coeficiente a pode ser determinado a partir de dois valores conhecidos da função, $f(x_1)$ e $f(x_2)$, para pontos arbitrários e distintos x_1 e x_2. Se temos as equações:

$$f(x_1) = ax_1 + b = y_1$$

e

$$f(x_2) = ax_2 + b = y_2,$$

podemos subtrair a segunda equação da primeira:

$$f(x_2) - f(x_1) = (ax_2 + b) - (ax_1 + b).$$

Observe que os termos b se cancelam:

$$y_2 - y_1 = ax_2 - ax_1.$$

Fatorando a do lado direito, temos:

$$y_2 - y_1 = a(x_2 - x_1).$$

Dividindo ambos os lados por $x_2 - x_1$ (note que $x_1 \neq x_2$):

$$\frac{y_2 - y_1}{x_2 - x_1} = a.$$

Assim, o coeficiente a é dado por:

$$a = \frac{y_2 - y_1}{x_2 - x_1}.$$

No caso de uma função linear $f(x) = ax$, podemos simplificar a determinação do coeficiente a escolhendo o ponto $(0, 0)$, que é sempre um ponto da função, pois $f(0) = 0$. Com isso, temos $x_1 = 0$ e $y_1 = 0$.

Assim, a expressão para a é dada por:

$$a = \frac{y_2 - y_1}{x_2 - x_1} = \frac{y_2 - 0}{x_2 - 0},$$

resultando em:

$$a = \frac{y_2}{x_2}.$$

No jogo, o valor de a pode ser encontrado observando como a joaninha se desloca para alcançar o ponto-alvo. A ideia é simples: o jogador precisa perceber a relação entre os movimentos da joaninha no eixo vertical e no eixo horizontal. Veja como isso funciona:

1. **Descubra quanto a joaninha anda para os lados:** Conte quantas unidades a joaninha precisa se mover para a direita ou esquerda até estar alinhada verticalmente com o alvo, isto é, até a abscissa da joaninha se igualar à do alvo. Esse número corresponde ao deslocamento horizontal.

2. **Descubra quanto a joaninha sobe ou desce:** Conte quantas unidades a joaninha precisa se mover para cima ou para baixo até alcançar o ponto-alvo. Esse número corresponde ao deslocamento vertical.

3. **Relacione esses movimentos:** Para cada unidade que a joaninha se move no eixo X, quantas unidades ela se move no eixo Y? Essa relação de proporcionalidade é o valor de a, e pode ser obtida dividindo o número de "passos" verticais pelo número de "passos" horizontais.

Posteriormente, o professor explicará ao estudante os conceitos formais relacionados ao comportamento de uma função:

- Uma função é considerada crescente quando $x_1 < x_2 \implies f(x_1) < f(x_2)$.
- Decrescente quando $x_1 < x_2 \implies f(x_1) > f(x_2)$.
- Monótona não-decrescente quando $x_1 < x_2 \implies f(x_1) \leq f(x_2)$.
- Monótona não-crescente quando $x_1 < x_2 \implies f(x_1) \geq f(x_2)$.

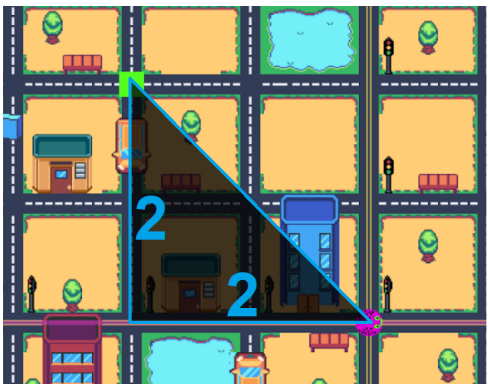
Contudo, o objetivo do jogo é introduzir esses conceitos de forma mais natural e facilitar a compreensão futura do formalismo, podemos adotar uma abordagem mais intuitiva neste momento.

Para determinar o sinal de a, o jogador deve comparar as posições da joaninha e do ponto-alvo, identificando qual dos dois está mais à esquerda no plano cartesiano. Partindo desse ponto mais à esquerda e seguindo em direção ao ponto mais à direita, podemos verificar quem está mais alto, ou seja, quem possui a maior ordenada.

- Se o ponto à esquerda estiver mais baixo do que o ponto à direita, a reta será crescente e a será positivo.
- Se o ponto à esquerda estiver mais elevado do que o ponto à direita, a reta será decrescente e a será negativo.
- Por fim, se ambos os pontos possuírem a mesma ordenada (isto é, estiverem na mesma altura), a reta será paralela ao eixo X, resultando em $a = 0$. Na Fase 1, o jogo foi projetado de modo que o alvo só pudesse assumir os valores de ordenada -1, 1 e 2, pois esses são os únicos valores que cabem na tela do jogo na direção vertical. Mais detalhes sobre a lógica do jogo podem ser encontrados na seção 10.1.1.

Por exemplo, se o ponto-alvo está em $(-2, 2)$, a joaninha anda 2 unidades para a esquerda e sobe 2 unidades. Assim, $|a| = \frac{2}{2} = 1$ (Figura 82).

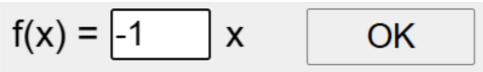
Figura 82: Exemplo para ilustrar como determinar o valor de a.



Fonte: Elaboração própria.

Observamos também que o ponto mais à esquerda (alvo) está mais elevado que a joaninha, portanto a reta é decrescente e a será negativo. O valor $a = -1$ deve ser inserido no campo de entrada localizado na barra superior do jogo (Figura 83).

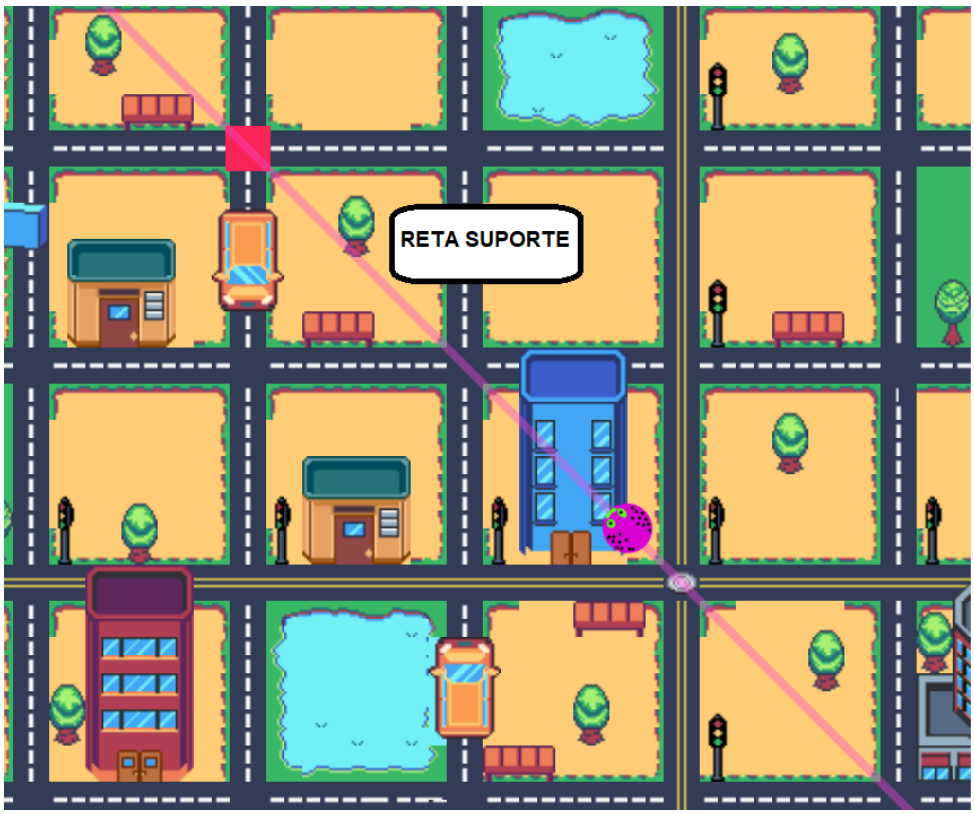
Figura 83: Inserindo o valor da taxa de variação da função no campo de entrada.



Fonte: Elaboração própria.

Após clicar em `OK` ou pressionar a tecla `ENTER`, o jogo ajustará a inclinação da reta de suporte (representada pela cor roxa). Se a reta passar pelo ponto-alvo, a joaninha se moverá até ele, coletando-o e retornando à origem.

Figura 84: Joaninha fazendo o percurso até o alvo.



Fonte: Elaboração própria.

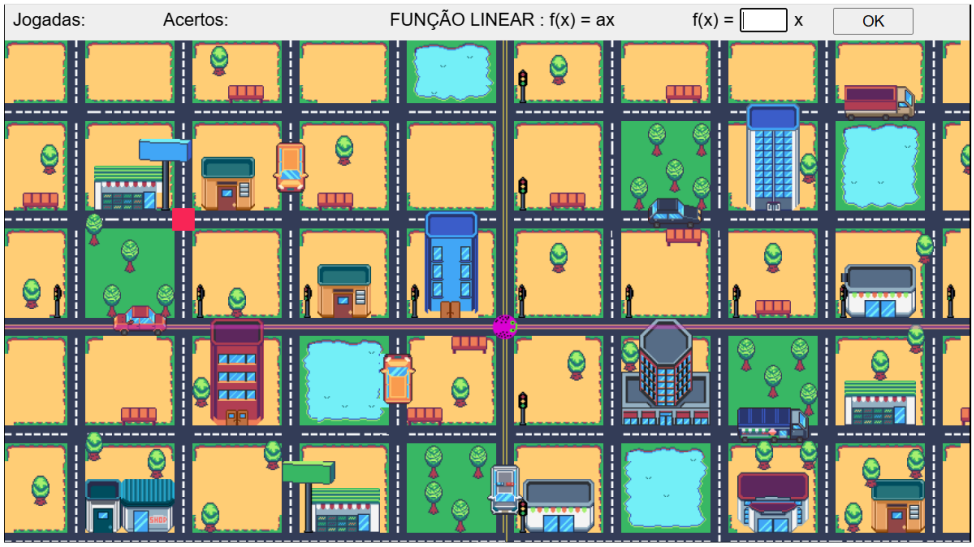
Em seguida, um novo ponto aleatório aparecerá em um dos cruzamentos. Caso a reta não passe pelo ponto de origem, a joaninha permanecerá no centro da cidade, aguardando que o jogador insira o valor correto do coeficiente a, enquanto a reta de suporte permanecerá visível, indicando o percurso que a joaninha faria (Figura 85).

Figura 85: Simulando um valor errado para a. Nesse caso, $a = 1$

Fonte: Elaboração própria.

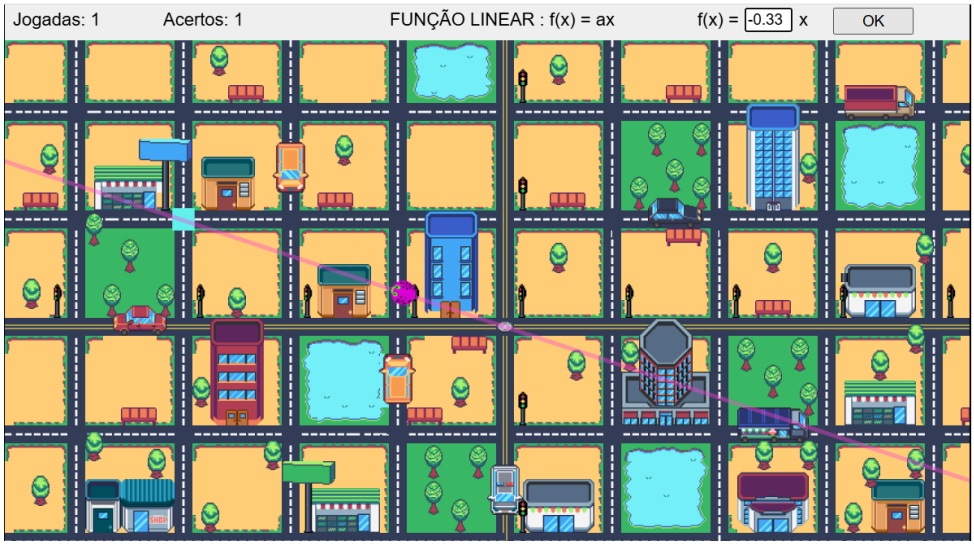
Quando o coeficiente a for um número racional, deve-se utilizar o ponto (.) em vez da vírgula (,) e garantir pelo menos duas casas decimais. Por exemplo, se o alvo estiver localizado no ponto $(-3, 1)$ (Figura 86), o valor do coeficiente a a ser inserido no campo de entrada será -0.33 (Figura 87).

Figura 86: Exemplo em que o coeficiente a será um número racional.



Fonte: Elaboração própria.

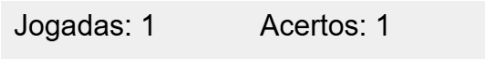
Figura 87: Coeficiente $a = -0.33$ inserido no campo de entrada, com a reta suporte correspondente ao valor esperado.



Fonte: Elaboração própria.

Para contabilizar os valores, a barra superior do jogo está munida de dois contadores: Um contador para guardar o número de acertos e outro para contabilizar o número de tentativas feitas pelo jogador até aquele momento (Figura 88).

Figura 88: Barra de jogadas e acertos.



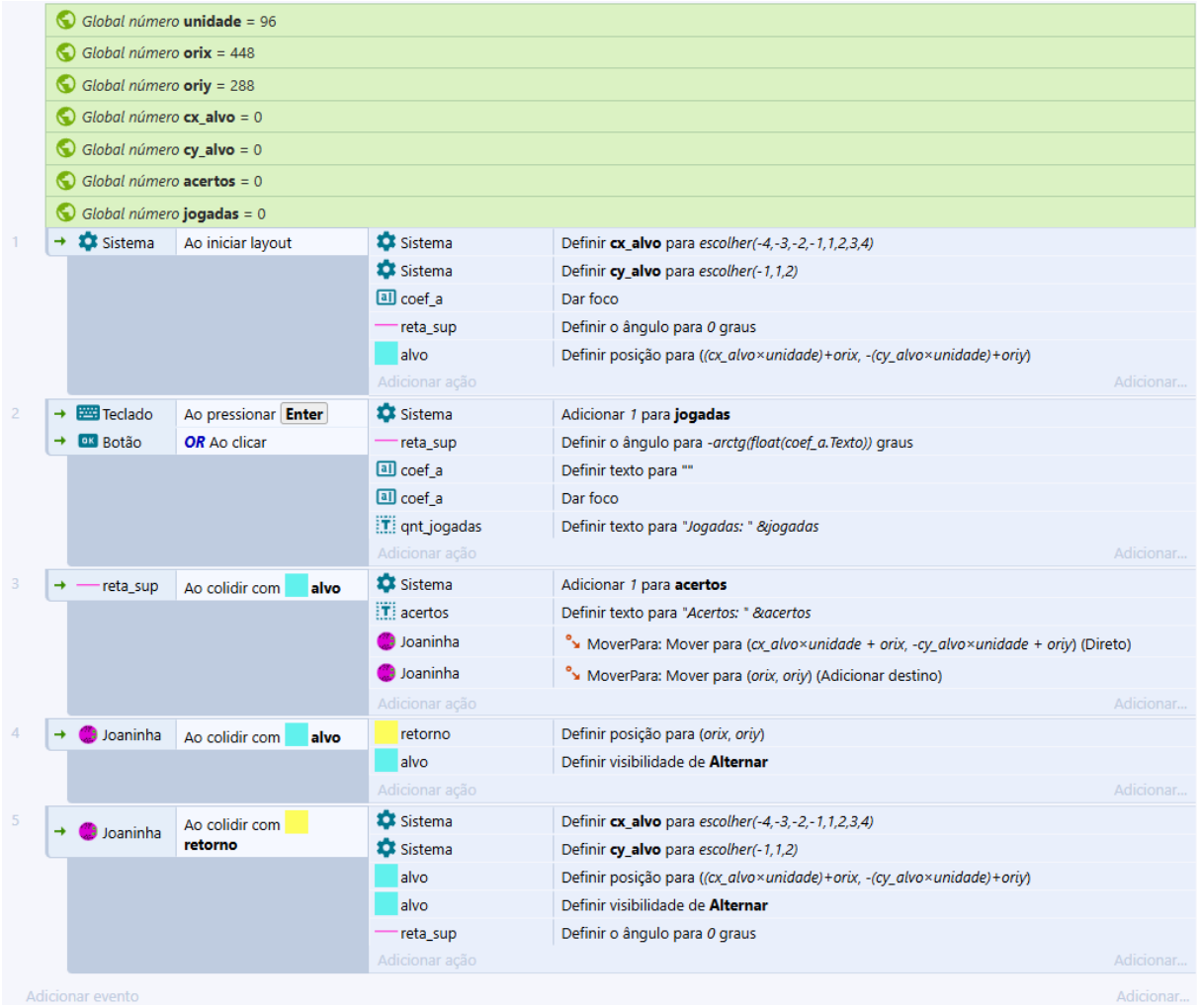
Fonte: Elaboração própria.

10.1.1 Explicando a lógica do jogo da joaninha fase 1

Inicialmente, ao observar a folha de eventos completa (Figura 89), notamos uma lista de variáveis definidas de forma global (blocos verdes que antecedem os azuis), que têm como objetivo facilitar o trabalho do desenvolvedor. Por exemplo, a unidade padrão foi configurada como 96 pixels. Assim, sempre que for necessário utilizar essa unidade, basta referenciar o valor armazenado na variável correspondente. Caso seja preciso ajustar a unidade, o desenvolvedor precisará alterar esse valor apenas uma vez, sem necessidade de modificações em outros pontos do código.

As variáveis `Orix` e `Oriy` armazenam as coordenadas da origem no plano cartesiano do Construct 3, enquanto `cx_alvo` e `cy_alvo` representam as coordenadas do ponto aleatório que será gerado no mapa. Além disso, foram criadas duas variáveis adicionais para registrar o número de jogadas realizadas e o número de acertos obtidos.

Figura 89: Folha de eventos da primeira fase.



Fonte: Elaboração própria.

O primeiro bloco de instruções é executado assim que o jogo inicia. Nesse momento, os valores das coordenadas do alvo são definidos, limitados pelos parâmetros estabelecidos pelo comando `choose`. Em seguida:

- O foco é direcionado para a caixa de texto onde o jogador deve inserir o valor.
- A reta de suporte é alinhada ao eixo X.
- O alvo é posicionado no mapa, respeitando as coordenadas previamente definidas.

O próximo passo ocorre quando o jogador pressiona a tecla `ENTER` ou clica no `botão OK`. Nesse momento:

- A variável que registra o número de jogadas é incrementada, mesmo que o jogador não tenha inserido nenhum valor.
- A inclinação da reta de suporte é ajustada para corresponder ao arco tangente do valor inserido pelo jogador.
- A janela de entrada de texto é limpa, o foco retorna para o campo de entrada, e o texto que exibe o número de jogadas é atualizado.

Neste ponto, é necessário discutir a sintaxe utilizada para determinar o novo ângulo da reta suporte, representada pela expressão:

$$-\text{atan}(\text{float}(\text{coef\_a.Text}))$$

Para acessar o valor inserido pelo jogador, utiliza-se a propriedade `.Text` do objeto que armazena o texto correspondente, sendo, neste caso, a variável `coef_a`. Em seguida, o texto extraído é convertido para um valor numérico por meio da função `float`. Após essa conversão, aplica-se a função arco tangente ao valor numérico, obtendo o ângulo correspondente. Por fim, como o Construct 3 adota o sentido horário para rotacionar objetos, é necessário inverter o sentido de rotação. Essa inversão é realizada multiplicando o valor do ângulo por -1.

O terceiro evento é ativado quando a reta suporte colide com o alvo, ou seja, quando o jogador insere o valor correto no campo de entrada. Ao ativar este bloco:

- O contador de acertos é incrementado em 1.
- O texto que exibe o número de acertos é atualizado.
- A joaninha se desloca até o ponto-alvo e retorna à origem, utilizando o comportamento `Mover para`.

Até este ponto, ainda não discutimos os comportamentos, um recurso essencial para a criação de jogos no Construct 3. Comportamentos são funcionalidades pré-definidas que podem ser adicionadas a objetos para que eles realizem ações específicas ou possuam características que tornem a jogabilidade mais rica e interativa. Esses recursos eliminam a necessidade de criar scripts complexos, permitindo que o desenvolvedor configure mecânicas de forma rápida e eficiente.

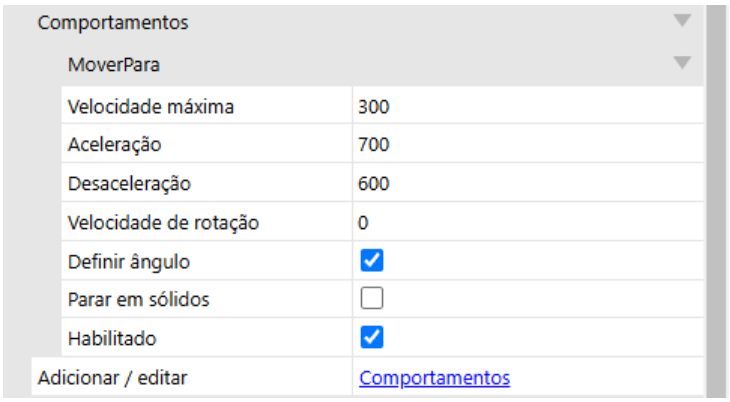
Para adicionar um comportamento a um objeto, como no caso da Joaninha, deve-se selecionar o objeto, clicar no botão `Comportamentos` no painel de propriedades e, em seguida, escolher o comportamento desejado na lista. No nosso exemplo, utilizamos o comportamento `Mover para`, que permite que a Joaninha se desloque automaticamente entre dois pontos, criando uma movimentação suave e eficiente.

Entre os comportamentos mais comuns no Construct 3, podemos citar o `Plataforma`, usado para jogos em que o personagem pode correr e saltar; o `Sólido`, que impede

que objetos atravessem uns aos outros; e o `Arrastar e soltar`, ideal para jogos com interações simples e intuitivas.

No comportamento `Mover para`, é possível acessar e ajustar configurações como velocidade, aceleração, desaceleração e outros parâmetros diretamente na barra de propriedades da Joaninha, garantindo o controle preciso sobre a mecânica de movimento (Figura 90).

Figura 90: Configurações Utilizadas no Comportamento `Mover para` da Joaninha.



Fonte: Elaboração própria.

Ainda no terceiro evento, destacamos como é realizada a atualização do número de acertos exibido na interface. O texto mostrado ao jogador é composto por duas partes: uma mensagem fixa e o valor dinâmico correspondente à quantidade de acertos armazenada na variável. Essa estrutura é representada da seguinte maneira:

```
"Acertos:  "&acertos
```

Para tornar essa composição mais clara, segue uma explicação detalhada de cada elemento:

- `"Acertos:  "`: Texto fixo exibido no início, que indica ao jogador que o número a seguir corresponde aos acertos.
- `&`: Operador usado para unir o texto fixo com o valor armazenado na variável, formando a mensagem final.
- `acertos`: Variável que contém o número atual de pontos alcançados pelo jogador, sendo atualizada a cada acerto.

O evento seguinte (bloco 4) funciona como um gatilho para ativar o quinto bloco.

- Um objeto chamado `retorno` é reposicionado na origem após a joaninha alcançar o ponto-alvo.

- O ponto-alvo torna-se invisível para evitar confusões caso seja gerado novamente no mesmo local.

Por fim, o último evento é ativado quando a joaninha retorna à origem e colide com o objeto `retorno` que foi posicionado na origem momentos antes pelo evento 4. Nesse momento:

- As variáveis `cx_alvo` e `cy_alvo` recebem novos valores.
- O alvo é reposicionado no mapa e sua visibilidade é reativada.
- O ângulo da reta de suporte é redefinido para 0, alinhando-a novamente ao eixo X.

Com essas ações concluídas, o jogo está pronto para que o jogador inicie um novo ciclo.

10.1.2 Cenário do jogo

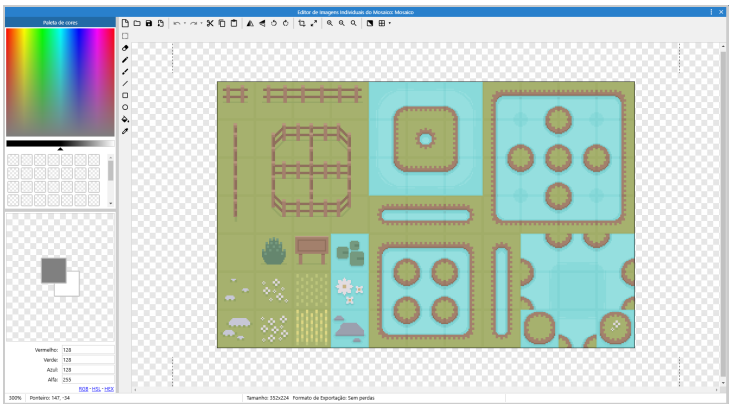
No Construct 3, os mosaicos (ou *tilesets*, em inglês) que são pequenas imagens, geralmente quadradas ou retangulares, que podem ser repetidas e organizadas para criar cenários ou objetos em um jogo. Eles funcionam como peças de um quebra-cabeça ou azulejos de um banheiro, permitindo que você use essas pequenas partes para montar ambientes maiores de forma eficiente e criativa.

Esses mosaicos são muito úteis para criar cenários, como chão, paredes, caminhos, grama, água, entre outros elementos que compõem o mundo do jogo. Além disso, oferecem vantagens práticas, como:

- **Economia de espaço:** É possível reutilizar as mesmas peças várias vezes, em vez de desenhar imagens grandes e complexas, tornando o jogo mais leve e rápido.
- **Facilidade de edição:** Se for necessário mudar algo no cenário, basta alterar um único mosaico, e as mudanças serão refletidas automaticamente em todos os lugares onde ele foi utilizado.

Adicionando Mosaicos ao Jogo

Figura 91: Mosaico padrão fornecido pelo Construct 3



Fonte: Elaboração própria.

Para adicionar um mosaico no Construct 3:

1. Insira um novo objeto no layout e selecione a opção **Mosaico** (ou **Tilemap**).
2. Uma janela será aberta exibindo um mosaico padrão, que serve apenas como exemplo fornecido pelo Construct 3 (Figura 91).

Você pode explorar opções mais criativas e personalizadas de mosaicos para o seu projeto, pesquisando por *tilesets* em sites especializados, como:

- **Itch.io - Game Assets** ([10]): Uma plataforma com uma vasta coleção de recursos para jogos, incluindo *tilesets*. Possui tanto opções gratuitas quanto pagas, com preços acessíveis e recursos diversificados.
- **CraftPix** ([6]): Um site especializado em *assets*[1] premium para jogos, como mosaicos, personagens e interfaces. Os recursos são pagos, mas frequentemente há pacotes gratuitos disponíveis.
- **OpenGameArt** ([17]): Uma plataforma colaborativa que oferece *assets* gratuitos para jogos, incluindo *tilesets*, músicas e efeitos sonoros. Todo o conteúdo é gratuito, geralmente sob licenças permissivas.

[1] No contexto de desenvolvimento de jogos, *assets* referem-se a recursos utilizados na criação de um jogo, como imagens, *tilesets* (mosaicos), músicas, efeitos sonoros, animações e outros elementos gráficos ou auditivos.

No desenvolvimento deste jogo, utilizamos o *tileset* **City Pack - Top Down - Pixel Art**, criado por **NYKNCK** e disponibilizado gratuitamente em **NYKNCK - City Pack Pixel Art** ([16]).

Você também pode optar por criar seus próprios (*tilesets*), permitindo uma personalização total do estilo visual do jogo. Essa abordagem é uma excelente oportunidade para estudantes desenvolverem suas habilidades artísticas e explorarem sua criatividade. Existem ferramentas (gratuitas e pagas) que facilitam a criação de mosaicos, como:

- **Piskel** ([21]): Ideal para iniciantes, é uma ferramenta online que permite criar sprites e mosaicos de forma simples e intuitiva.
- **Krita** ([12]): Um software de código aberto, muito usado para arte digital, que oferece recursos avançados para criação de mosaicos.
- **Aseprite** ([1]): Voltado para pixel art, permite criar mosaicos com animações e é amplamente utilizado na indústria de jogos.

Carregando um Tileset Personalizado

Após encontrar ou criar um mosaico da sua preferência, precisamos carrega-lo no editor de imagens do mosaico.

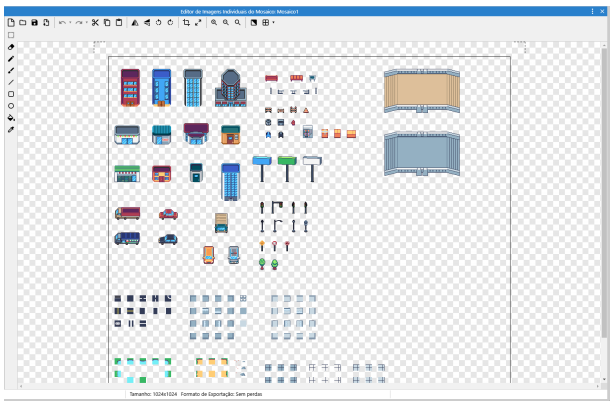
1. Clique no ícone de pasta (Figura 92) ou use o atalho `CTRL + O` para abrir a janela de seleção de arquivos.

Figura 92: Botão para carregar imagem de arquivo.

Fonte: Elaboração própria.

2. Encontre o *tileset* e clique em `Abrir` para carregá-lo no editor.
3. O mosaico padrão será substituído pela arte escolhida (Figura 93).
4. Feche o editor para retornar ao layout principal.

Figura 93: Arte do mosaico carregada e pronta para uso.



Fonte: Elaboração própria.

Pintando o Layout com Mosaicos

Agora que o mosaico está pronto:

1. Acesse a janela de Mosaico (ative-a caso não esteja visível, você pode consultar como fazer isso clicando aqui).
2. Selecione a ferramenta `Desenhar Bloco` (Figura 94) (ícone de lápis ou atalho `tecla 2 no teclado`).
3. Escolha a peça desejada na janela do mosaico, selecionando-a conforme feito na (Figura 94).
4. Clique e arraste no layout para posicionar as peças conforme sua criatividade.

Figura 94: Selecionando um prédio do Mosaico.

Fonte: Elaboração própria.

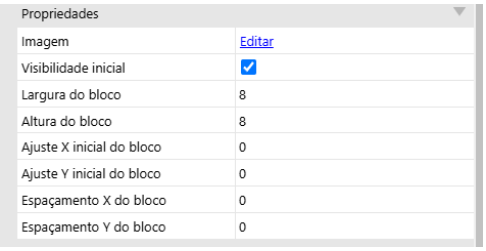
Configurando o tamanho dos blocos

Com o mosaico selecionado, você pode ajustar o tamanho do bloco utilizado para desenhar no layout. O bloco é representado por um quadrado visível na janela do mosaico, indicando o tamanho atual da área que será "pintada" de uma só vez. Para modificar esse valor:

- Vá até a janela de Propriedades do mosaico.
- Altere as dimensões do pincel, definindo a largura e a altura em pixels.

No caso deste jogo, utilizamos um tamanho de 8×8 pixels (Figura 95), ideal para trabalhos detalhados em *pixel art*. Tamanhos maiores podem ser mais adequados para cenários com menos detalhes ou para acelerar o processo de pintura. Recomenda-se testar diferentes configurações para encontrar o tamanho que melhor se adapta ao seu projeto.

Figura 95: Configurações utilizadas no tamanho dos blocos do Mosaico.



Fonte: Elaboração própria.

Organização das Camadas

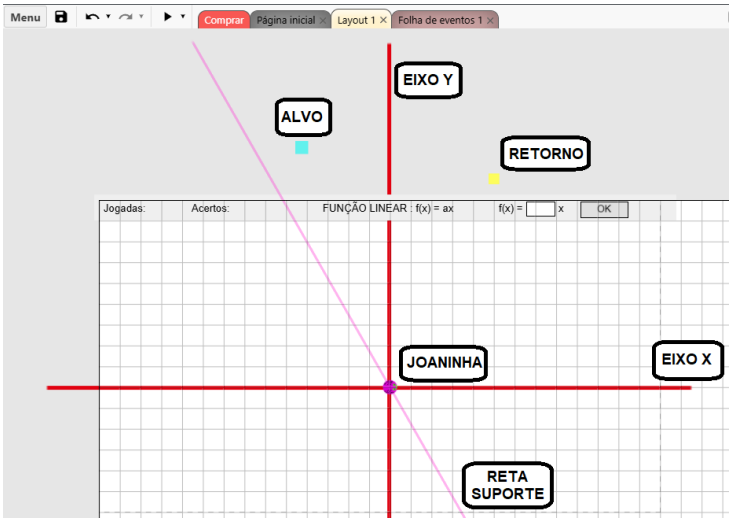
Sempre que finalizar o trabalho em uma camada proteja-a contra alterações acidentais clicando com o botão direito na camada correspondente no layout, selecionando a opção `Bloquear` e, em seguida, `Bloquear Seleção`. Caso precise editar uma camada bloqueada posteriormente, clique com o botão direito na tela, selecione `Bloquear` e escolha `Desbloquear Tudo`.

Após garantir que a organização das camadas esteja clara, apresentarei como organizei os mosaicos neste projeto. O objetivo foi distribuir os elementos em camadas distintas, facilitando o gerenciamento e a edição de cada parte do mosaico separadamente. Para isso, utilizei três camadas montadas estrategicamente, conforme descrito a seguir:

Jogo sem mosaico

No estágio atual, o layout já conta com a representação de um alvo e de uma joaninha, mas ainda carece de elementos adicionais que possam delimitar os quarteirões ou enriquecer a ambientação (Figura 96).

Figura 96: Layout sem adição de mosaico.

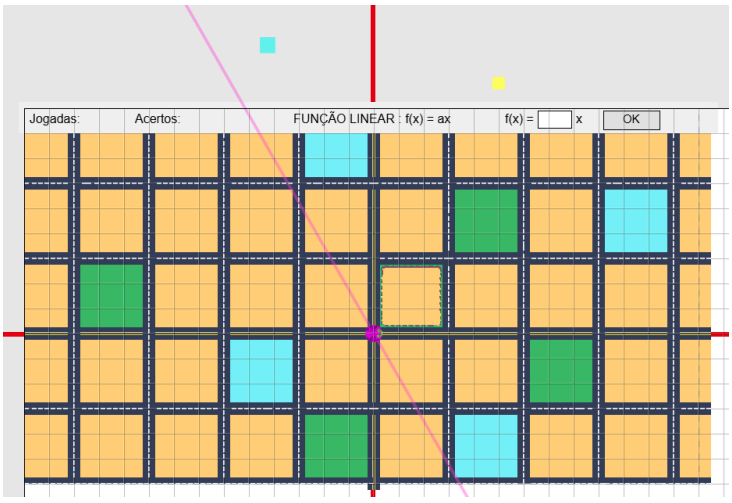


Fonte: Elaboração própria.

Camada 1: Definição dos Quarteirões.

A primeira camada contém as ruas horizontais e verticais que definem os quarteirões. Nesta etapa, também foram atribuídas as cores de cada quarteirão, garantindo a organização visual básica do mosaico (Figura 97).

Figura 97: Finalização do mosaico 1 no Layout

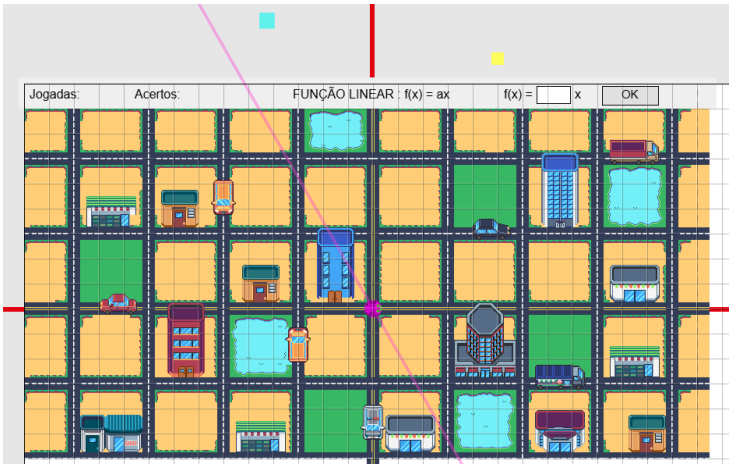


Fonte: Elaboração própria.

Camada 2: Construções e Bordas.

Na segunda camada, adicionei as casas e prédios, além dos detalhes das bordas dos quarteirões, complementando a estrutura básica com elementos que trazem vida e personalidade ao mosaico (Figura 98).

Figura 98: Adição da segunda camada ao Layout.

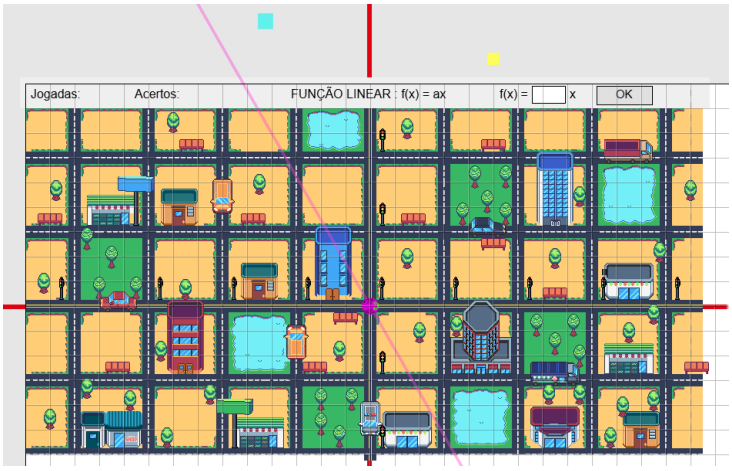


Fonte: Elaboração própria.

Camada 3: Elementos Ambientais e Detalhes Finais.

A última camada foi reservada para adicionar elementos ambientais e os detalhes finais, como árvores, placas, pontos de ônibus e os cruzamentos das ruas. Esses elementos complementam o mosaico e conferem maior realismo e profundidade ao design (Figura 99).

Figura 99: Adição da terceira camada ao Layout.



Fonte: Elaboração própria.

Cada mosaico adicionado ao layout foi tratado como um objeto distinto, permitindo maior controle sobre sua disposição e configuração. A Figura 100 exibe a lista completa de objetos utilizados no projeto, evidenciando a organização e a separação de camadas para facilitar ajustes e aprimoramentos no design.

Figura 100: Visualização completa da Barra de Projeto da Fase 1.

Fonte: Elaboração própria.

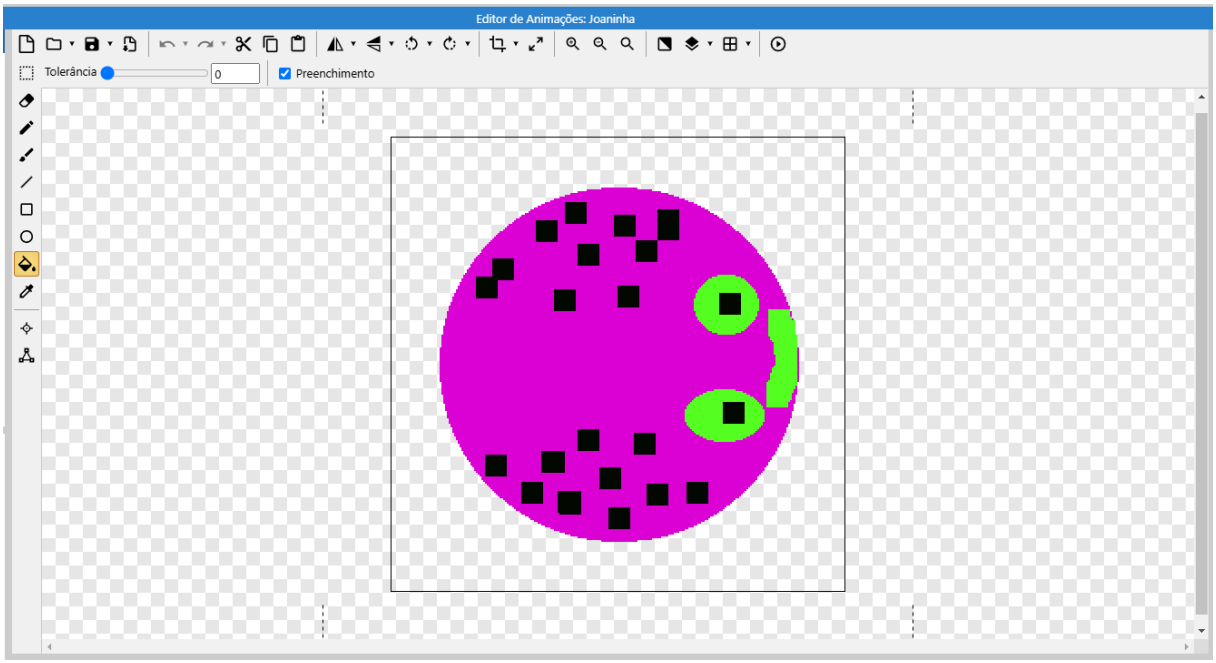
Note que na lista de objetos foi adicionado um teclado, que desempenha um papel fundamental na captura das entradas do jogador. No Construct 3, o objeto teclado permite detectar e responder a teclas pressionadas durante o jogo, facilitando a criação de interações baseadas em comandos específicos, como movimentação, ações ou seleção de opções no jogo. Embora as teclas numéricas e muitas outras teclas básicas já possam ser detectadas diretamente, sem a necessidade de adicionar explicitamente o objeto teclado ao projeto, sua inclusão torna-se essencial para capturar eventos específicos, como pressionar a tecla `ENTER` ou outras teclas especiais (por exemplo, `SHIFT`, `CTRL`, `ESPACO`).

Com isso, na fase da Joaninha, o jogador pode simplesmente digitar o valor desejado e pressionar a tecla `ENTER`, sem precisar clicar no botão `OK` para iniciar o movimento da Joaninha.

10.1.3 Personagens customizados

No novo projeto, o sprite da joaninha apresenta uma representação mais fiel à sua aparência natural, diferentemente do jogo do carteiro aventureiro, no qual o personagem principal era simbolizado por um simples quadrado rosa. Para a criação da joaninha, foi utilizado o editor de animações, fazendo uso das ferramentas de lápis, pincel, elipse e preenchimento, obtendo o resultado da Figura 101

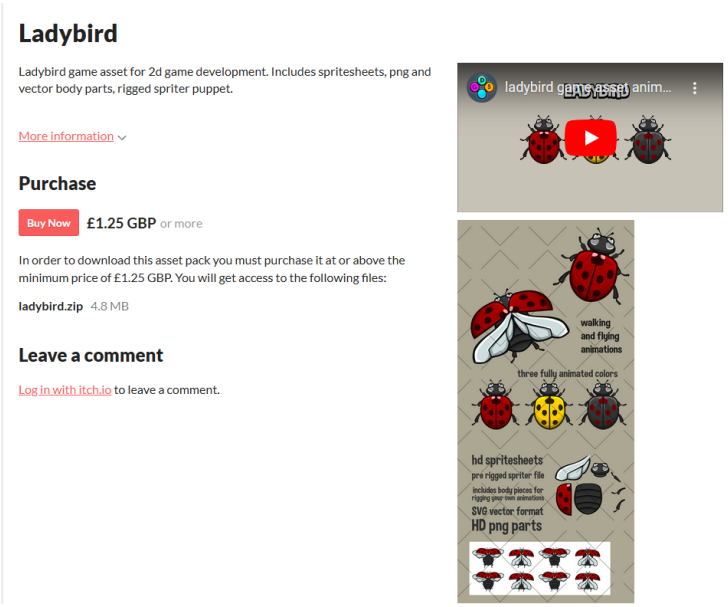
Figura 101: Joaninha criada apenas com as ferramentas do Construct 3



Fonte: Elaboração própria.

Esse processo oferece uma excelente oportunidade para integrar o lado artístico e criativo dos estudantes, possibilitando o desenvolvimento de habilidades visuais e técnicas. Caso os alunos não se sintam à vontade para criar seus próprios personagens, podem recorrer a sites como [10], [6] e [17], que oferecem boas referências para o projeto. Um exemplo disso é uma referência obtida em [10], onde é possível encontrar uma arte de joaninha mais detalhada (Figura 102). Embora essa arte não esteja disponível gratuitamente, ela pode servir como uma fonte de inspiração para a criação de personagens originais, utilizando softwares como Aseprite [1] ou Piskel [21].

Figura 102: Referência de uma joaninha encontrada no *itch.io* [10].



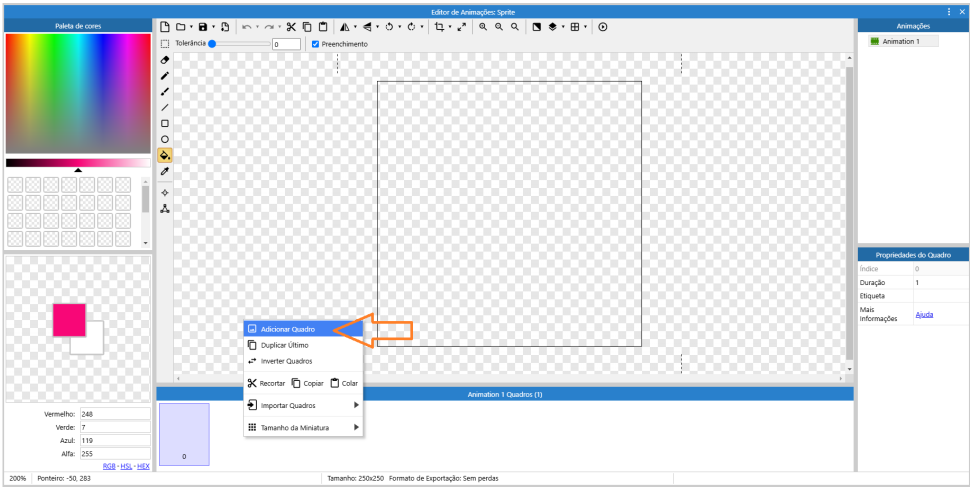
Fonte: https://gamedeveloperstudio.itch.io/ladybird

10.1.4 Animações

A animação é um processo fundamental no desenvolvimento de jogos, responsável por criar a ilusão de movimento e dar vida aos personagens e elementos gráficos. No contexto do nosso projeto, estamos utilizando animações para alterar a cor do destino no mapa, com o objetivo de chamar a atenção do jogador. Esse efeito visual é realizado por meio de animações simples, mas eficientes, que alteram as cores de forma dinâmica.

Para criar essa animação, utilizamos a janela **Editor de Animações**. Na parte inferior dessa janela, existe uma aba chamada `Animations`, onde são armazenadas e editadas todas as animações. Para adicionar uma nova animação, basta clicar com o `botão direito` em um espaço em branco dentro dessa área e selecionar a opção `Adicionar Quadro` (Figura 103). Isso fará com que um novo quadro seja gerado na aba de animações. Ao selecionar esse quadro, é possível criar uma nova arte para ele.

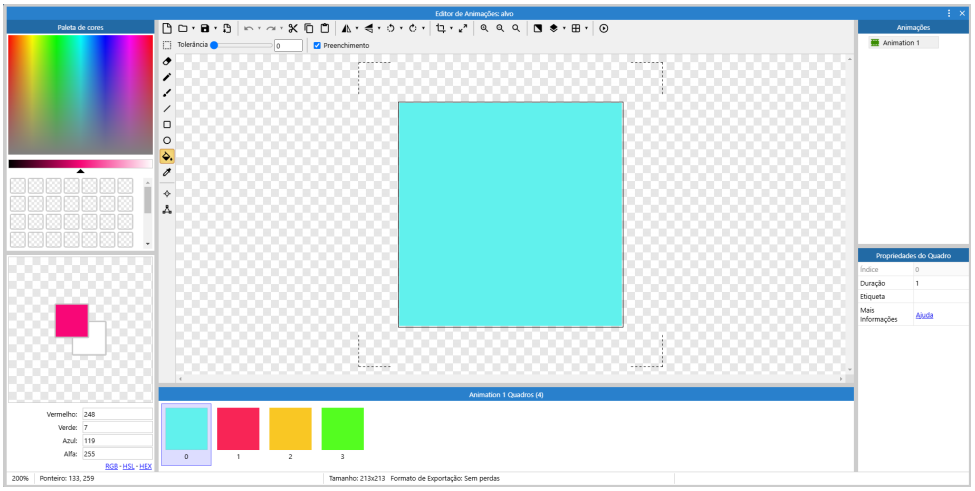
Figura 103: Opção de adicionar um novo quadro na aba de animações.



Fonte: Elaboração própria.

No caso do efeito de piscamento do alvo, criamos uma sequência de quadros com cores variadas, conforme ilustrado na Figura 104. Esse padrão de quadros alternados cria a impressão de que o destino está piscando no mapa, chamando a atenção do jogador.

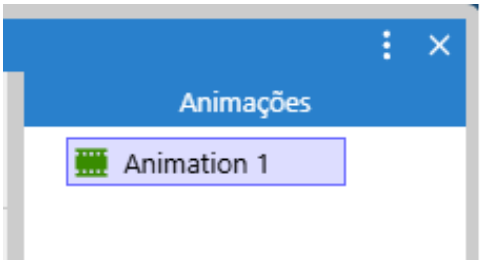
Figura 104: Quadros utilizados para fazer a animação do alvo.



Fonte: Elaboração própria.

Além de desenhar os quadros, é necessário configurar alguns parâmetros para garantir que a animação funcione corretamente. Na aba direita da janela do **editor de animações**, você encontrará um arquivo de animação nomeado `Animation 1`, conforme exibido na Figura 105.

Figura 105: Arquivo `Animation 1` da janela do editor de animações.



Fonte: Elaboração própria.

Ao selecionar esse arquivo, a aba `Propriedades da Animação` será exibida na parte inferior da janela. Nesse painel, é possível ajustar várias variáveis que controlam o comportamento da animação. Para o efeito do destino piscante, ajustamos a velocidade da animação para `16`, ativamos a opção `Repetir` (o que faz com que a animação continue indefinidamente após completar a leitura dos quadros) e selecionamos o efeito `Ping Pong` (Figura 106). O efeito Ping Pong é um tipo de animação em que os quadros são lidos de forma alternada: ao chegar ao último quadro, a animação reverte e começa a ser executada na direção oposta, criando um movimento de vai-e-vem. Esse efeito é ideal para criar a ilusão de um ciclo contínuo, como o do piscar do destino no mapa.

Figura 106: Aba de propriedades da animação.

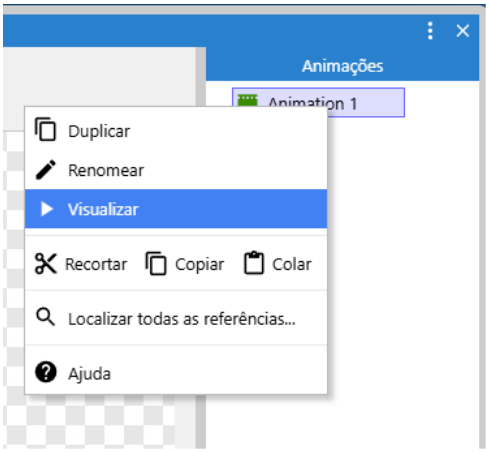
Propriedades da Animação	
Nome	Animation 1
Velocidade	16
Repetir	☑
Contagem de Repetições	1
Repetir Para	0
Ping Pong	☑
Mais Informações	Ajuda

Fonte: Elaboração própria.

Uma maneira de verificar como a animação ficará no jogo é utilizando a função de visualização. Para isso, clique com o botão `direito sobre` o arquivo `Animação`

`1` e selecione a opção `Visualizar` (Figura 107). Isso abrirá uma janela onde será possível observar como a animação será apresentada ao jogador final. Se o resultado não corresponder às expectativas do desenvolvedor, é possível ajustar os parâmetros da animação até que o efeito desejado seja alcançado.

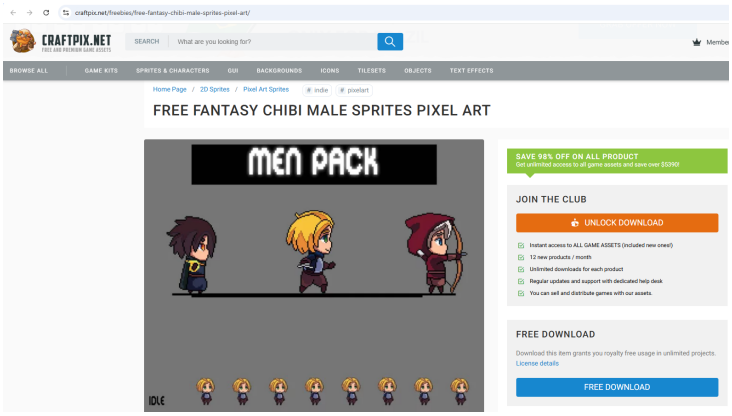
Figura 107: Opção de visualizar a animação.



Fonte: Elaboração própria.

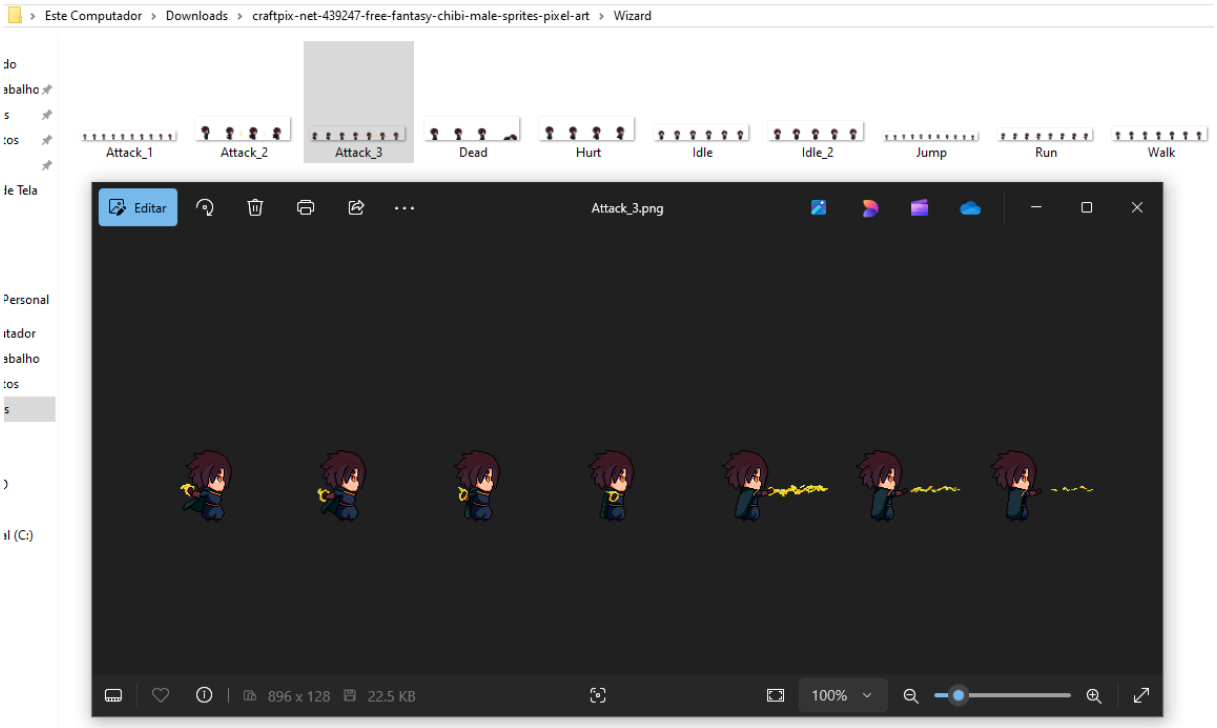
Além de criar animações do zero, os desenvolvedores podem optar por baixar animações prontas da internet. Usando a referência [6] é possível baixar de forma gratuita o arquivo *Free Fantasy Chibi Male Sprites Pixel Art* (Figura 108). Nesse arquivo, estão disponíveis três personagens, com animações que incluem situações como ficar parado, andar, pular, atacar, entre outras. Após o download, você encontrará animações como a mostrada na Figura 109.

Figura 108: Parte da página da animação *Free Fantasy Chibi Male Sprites Pixel Art*, disponível em [6].



Fonte: https://craftpix.net/freebies/free-fantasy-chibi-male-sprites-pixel-art/

Figura 109: Animação *attack 3* de *Free Fantasy Chibi Male Sprites Pixel Art*, disponível em [6].

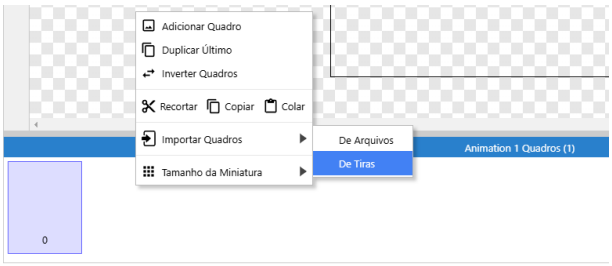


Fonte: Elaboração própria.

Para adicionar essa animação ao seu jogo, o processo é simples. Primeiro, crie um novo

`sprite` e, na janela de edição do sprite, vá até a aba `Animações`. Clique com o `botão direito` e selecione a opção `Importar Quadros`, depois escolha `De tiras` (Figura 110).

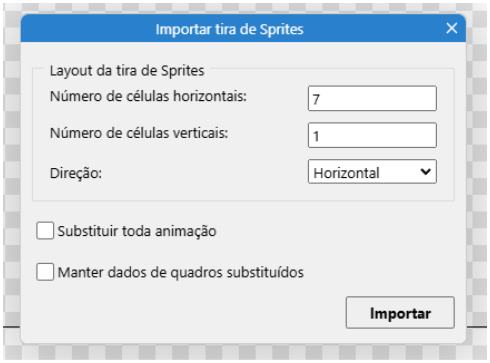
Figura 110: Importando uma animação em tiras.



Fonte: Elaboração própria.

Na sequência, localize e selecione o arquivo da animação que deseja importar. No nosso exemplo, escolhemos a animação mostrada na Figura 109, na qual o personagem aparece 7 vezes, indicando que a animação contém 7 quadros. Após selecionar o arquivo, uma janela surgirá, sugerindo o número de células horizontais (quadros). Como neste caso a sugestão é de 7, que corresponde corretamente ao número de quadros da animação, basta clicar em `Importar` (Figura 111).

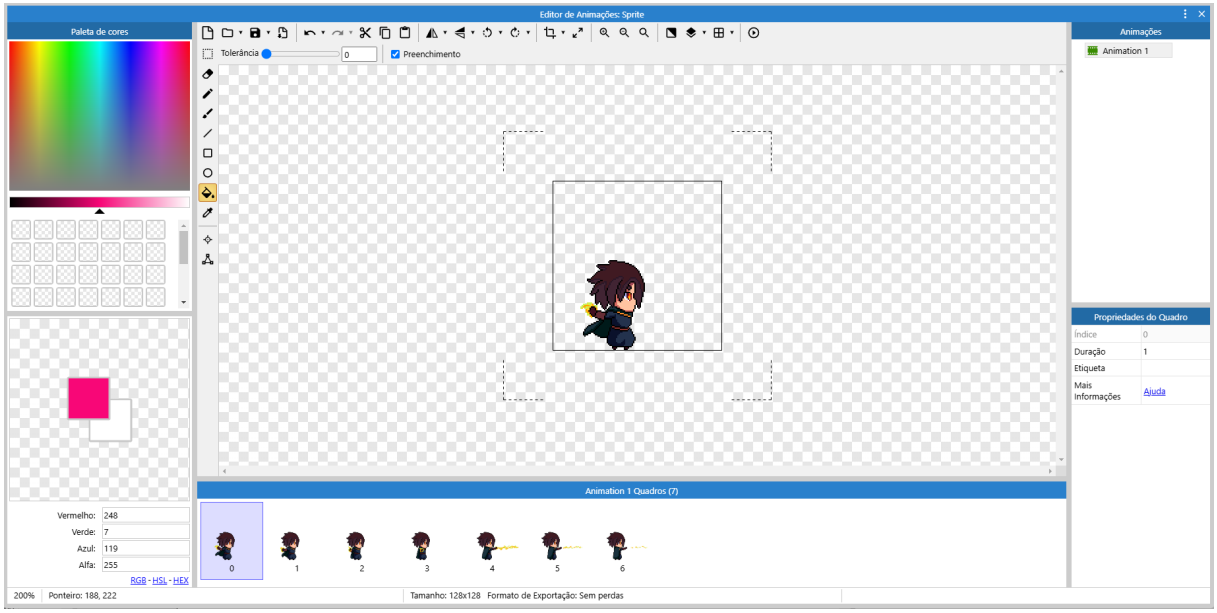
Figura 111: Janela de importação de tiras.



Fonte: Elaboração própria.

Agora, na aba de animações, você terá 8 quadros, sendo o primeiro quadro um espaço em branco. Para removê-lo, clique sobre o quadro e pressione a tecla `Delete`, obtendo o resultado da Figura 112 . Após excluir o primeiro quadro, você poderá ajustar a velocidade e a repetição da animação conforme necessário. Por fim, utilize a função de visualização para testar como a animação será exibida no jogo e faça ajustes adicionais, se necessário, até que o efeito final atenda às expectativas do projeto. Ao finalizar, clique em fechar e seu personagem estará pronto para ser usado no jogo.

Figura 112: Finalização da animação importada.



Fonte: Elaboração própria.

10.2 Fase 2: Função afim

A fase dois do jogo da joaninha apresenta uma proposta semelhante à da primeira etapa: introduzir e explorar um problema cuja modelagem é possível por meio de uma função. Nesta fase, a joaninha (representada por um ponto roxo) ainda precisa alcançar o alvo (representado pelo ponto ciano) no mapa seguindo uma reta (Figura 113). Contudo, ao contrário da fase 1, em que a joaninha sempre começava na origem, nesta fase ela inicia em um ponto aleatório no eixo Y, cuja ordenada é sempre um número inteiro. Esse novo cenário sugere uma modelagem por meio da função afim $f(x) = ax + b$.

Figura 113: Fase 2 do jogo da Joaninha.

Fonte: Elaboração própria.

Na seção 10.1 explicamos como o coeficiente a pode ser determinado a partir de dois pontos distintos, onde dado os pontos arbitrários $A(x_1, y_1)$ e $B(x_2, y_2)$ sobre a reta, o coeficiente a da função pode ser calculado pela expressão:

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$

Por outro lado, se tomarmos $x = 0$, teremos:

$$f(x) = ax + b$$

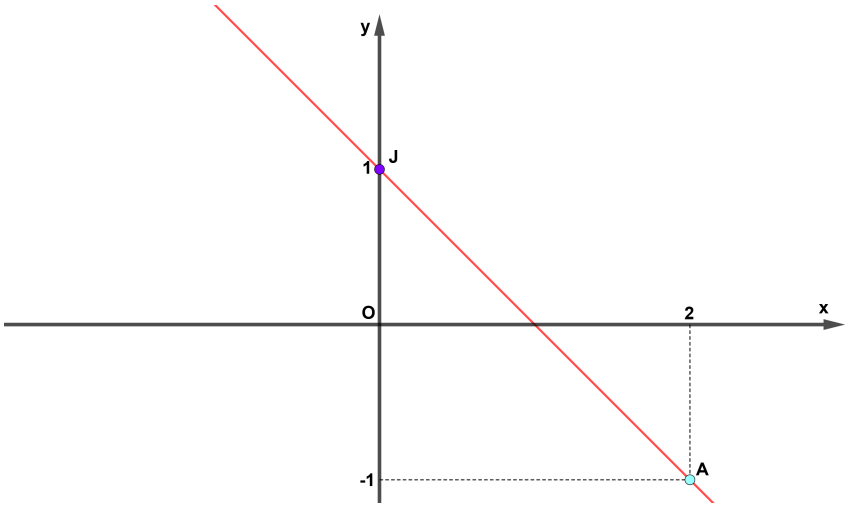
$$f(0) = a \cdot 0 + b$$

$$f(0) = b$$

Assim, o valor de b corresponde ao valor da função quando $x = 0$, ou seja, ao ponto onde o gráfico da função corta o eixo y.

A situação da (Figura 113) pode ser modelada matematicamente conforme a (Figura 114), onde o ponto J de coordenadas $(0, 1)$ representa a joaninha e o ponto A, de coordenadas $(2, -1)$ representa o alvo. O nosso objetivo é determinar os coeficientes da função $f(x) = ax + b$ que está representada pela linha vermelha.

Figura 114: Modelagem matemática da (Figura 113).



Fonte: Elaboração própria.

Para determinar b, basta observar onde a reta intercepta o eixo y, isto é, no ponto em que $x = 0$, pois aplicando $x = 0$ na função temos $f(0) = a \cdot 0 + b = b$. No exemplo em questão, como a joaninha está no ponto $(0, 1)$, o coeficiente b é diretamente igual a 1. Agora, para determinar o coeficiente a, podemos utilizar a fórmula da taxa de variação entre os dois pontos dados, que é dada por:

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$

Substituindo os valores:

$$a = \frac{-1 - (+1)}{2 - 0}$$

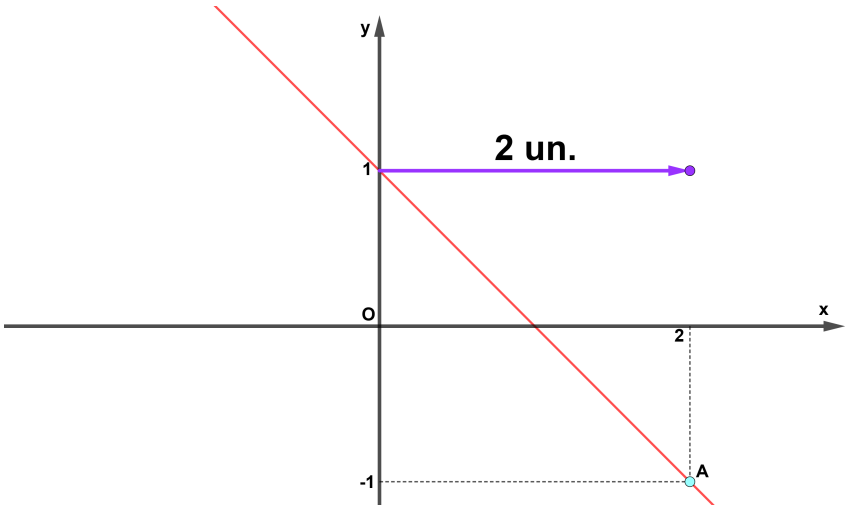
$$a = \frac{-2}{2}$$

$$a = -1$$

No entanto, antes de recorrer à fórmula, podemos interpretar geometricamente o valor de a. Como abordado na seção 10.1, a taxa de variação da reta corresponde à razão entre o deslocamento vertical e o deslocamento horizontal que a joaninha realiza para alcançar o alvo. No caso presente, a joaninha se desloca 2 unidades na direção x (de $x_1 = 0$ até $x_2 = 2$) (Figura 115) e em seguida 2 unidades na direção y (de $y_1 = 1$ até $y_2 = -1$) (Figura 116). Assim, a magnitude do coeficiente angular é dada por:

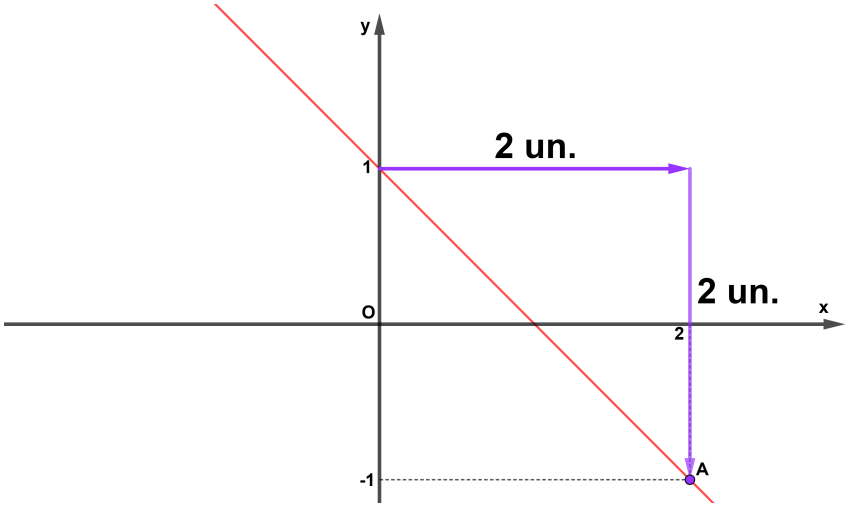
$$|a| = \frac{\text{Variação Vertical}}{\text{Variação Horizontal}} = \frac{2}{2} = 1$$

Figura 115: Deslocamento horizontal.



Fonte: Elaboração própria.

Figura 116: Deslocamento vertical.



Fonte: Elaboração própria.

Para determinar se a é positivo ou negativo, basta observar a disposição relativa dos dois pontos. A joaninha, que está à esquerda do alvo, possui uma ordenada maior (1 contra -1). Portanto, a reta é decrescente, o que implica que o coeficiente a será negativo. Assim, temos $a = -1$. Portanto os coeficientes da função afim que conecta a joaninha ao alvo são $a = -1$ e $b = 1$.

Com base nisso:

- O jogador deve inserir os valores de a e b nos campos correspondentes e confirmar a entrada pressionando o botão "OK"ou a tecla `ENTER` (Figura 117).

Figura 117: Inserindo os valores de a e b nos campos de entrada.

Fonte: Elaboração própria.

- Em seguida, uma linha roxa representando o percurso definido pelos coeficientes fornecidos será exibida no mapa.
- Se os valores estiverem corretos, a linha passará pelo ponto inicial da joaninha e pelo alvo, ativando seu movimento até o destino (Figura 118).

Figura 118: Joaninha em movimento pela reta formada com os valores de a e b.

Fonte: Elaboração própria.

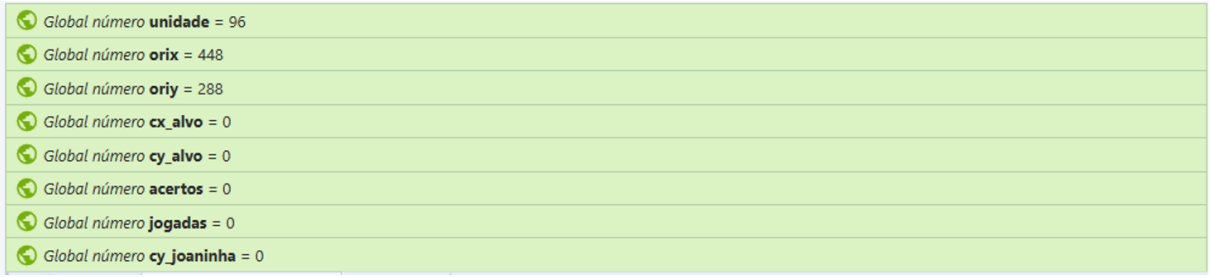
- Caso os valores de a ou b estejam incorretos, a linha será traçada no mapa indicando o caminho que a joaninha seguiria com esses valores, permitindo ao jogador corrigir os coeficientes informados.

Folha de Eventos da Fase Dois

A base para construir essa fase foi a folha de eventos da fase 1, com algumas adições e ajustes. Vamos detalhar essas modificações.

Variáveis Globais

Figura 119: Variáveis globais da fase 2.

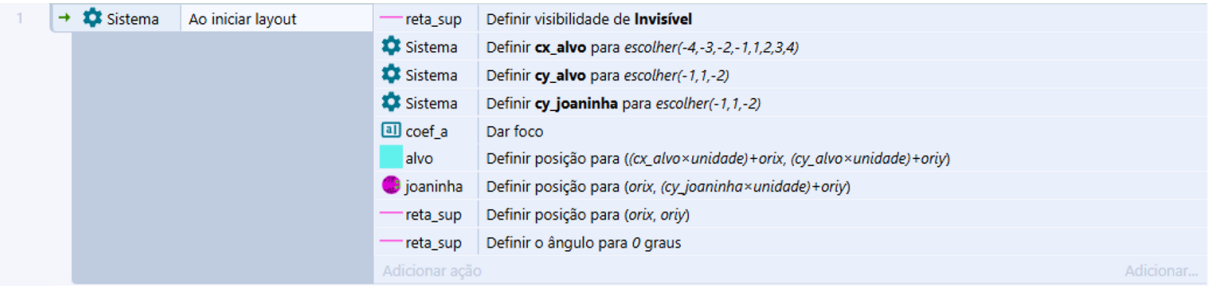


Fonte: Elaboração própria.

Foi adicionada a variável `cy_joaninha`, que armazena os valores do eixo Y onde a joaninha se posiciona (Figura 119).

Evento Número 1

Figura 120: Evento número 1 da folha eventos da fase 2.



Fonte: Elaboração própria.

Ao iniciar o layout, optou-se por deixar a reta suporte invisível, considerando questões estéticas, especialmente porque, neste jogo, a joaninha não está mais necessariamente posicionada na origem. Além disso, foi adicionado um valor aleatório à variável `cy_joaninha`, utilizando o comando `choose`. Após essa etapa, a joaninha e o alvo foram posicionados no mapa, enquanto a reta suporte permaneceu oculta, alinhada com o eixo X (Figura 120).

Evento Número 2

Figura 121: Evento número 2 da folha eventos da fase 2.

Fonte: Elaboração própria.

Este evento é acionado quando o jogador pressiona `ENTER` ou clica em `OK` (Figura 121). Nesse momento:

- O número de jogadas é incrementado em uma unidade.
- A reta suporte é ajustada conforme os coeficientes a e b inseridos pelo jogador e se torna visível.
- Os valores digitados são apagados, e o foco retorna à caixa de texto para a entrada do coeficiente a.
- O texto que exibe o número de jogadas é atualizado.

Eventos 3, 4 e 5

Figura 122: Eventos números 3, 4 e 5 da folha eventos da fase 2.

Fonte: Elaboração própria.

Se o jogador inserir os valores corretos, a joaninha e o alvo estarão alinhados na reta suporte, indicando que os coeficientes a e b foram acertados e ativando o evento número 3 (Figura 122). Primeiramente, a variável que registra o número de acertos é incrementada em uma unidade, e o texto que exibe o número de acertos é atualizado. Em seguida, a joaninha se desloca até o alvo seguindo a trajetória da reta suporte. Quando a joaninha alcança o alvo, o evento número 4 é acionado, ativando o sinalizador denominado `tocou`.

Este sinalizador é utilizado para controlar a sequência das ações programadas, garantindo que as etapas seguintes ocorram apenas após a joaninha ter alcançado o alvo. Sem o uso do sinalizador, as ações subsequentes poderiam ser disparadas de maneira desordenada ou antes do deslocamento da joaninha ser concluído, o que comprometeria tanto o funcionamento lógico quanto a experiência visual do jogo.

Após o sinalizador ser ativado, um intervalo de 0,2 segundos é aplicado para aprimorar o efeito visual. Em seguida, o valor da variável associada à posição no eixo y da joaninha é alterado aleatoriamente entre valores pré-determinados. Enquanto a joaninha retorna ao eixo y utilizando o novo valor da variável `cy_joaninha`, o alvo é temporariamente ocultado, reposicionado em novas coordenadas e, posteriormente, reaparece em um novo local. Quando a joaninha toca o eixo y, o evento número 5 é acionado junto com o sinalizador `voltou`. A partir desse ponto, as ações restantes do evento número 3

continuam, deixando o alvo visível novamente e ocultando a reta suporte, que é realinhada ao eixo x.

Assim, o jogo estará preparado para receber os novos valores da taxa de variação a e do coeficiente b, fechando um ciclo completo.

Conclusão

A partir de toda a trajetória descrita nesta dissertação, constatamos que a criação de jogos didáticos no Construct 3, focados no plano cartesiano e em funções, proporciona aos estudantes uma experiência de aprendizagem ativa e motivadora. Na disciplina eletiva em que este trabalho foi aplicado, observou-se um aumento significativo do interesse dos alunos em participar das aulas, sendo notória a procura deles durante os momentos livres para mostrar seus progressos nos jogos, tirar dúvidas e sugerir melhorias. Esse engajamento maior não se restringiu apenas aos estudantes com maior afinidade pela tecnologia: mesmo aqueles que, inicialmente, demonstravam pouco entusiasmo pela Matemática passaram a se envolver mais ativamente na elaboração de fases, na inserção de códigos e na discussão sobre como resolver problemas de posicionamento e escalas.

O envolvimento dos estudantes ao longo do projeto foi notável. A imersão na criação de jogos despertou um interesse inédito entre os participantes. Muitos passaram a buscar conhecimentos avançados em programação, dedicando-se ao aprendizado de novas linguagens, enquanto outros descobriram afinidades com áreas complementares, como robótica e desenvolvimento de sistemas. O projeto, assim, não apenas cumpriu seu objetivo de atrair alunos, mas também plantou sementes para futuras carreiras tecnológicas, mostrando como a prática criativa pode transformar curiosidade em motivação acadêmica.

As estratégias aqui apresentadas, exemplificadas pelos jogos Carteiro Aventureiro e Jogo da Joaninha, podem ser adaptadas a diversos outros conteúdos matemáticos. Essa proposta, por sua vez, alinha-se às perspectivas de Papert (1996) [20], que ressaltam a relevância do construcionismo e do protagonismo do aluno; e dialoga com as ideias de Prensky (2001) [22], no que se refere ao potencial dos recursos digitais para envolver os nativos digitais. Também corrobora a visão de Mattar (2013) [15], na medida em que a utilização de tecnologias e de jogos na escola fortalece a inovação curricular. Em consonância com a BNCC (Brasil, 2018) [3], tais abordagens incentivam a conexão entre teoria e prática, gerando oportunidades de aprendizagem mais contextualizadas e significativas.

Como continuidade deste trabalho, sugere-se a elaboração de um jogo único que possa abarcar tópicos matemáticos mais amplos (por exemplo, Geometria, Probabilidade, Estatística e Álgebra), enriquecendo a interface com maior diversidade de cenários, níveis graduais de dificuldade e elementos artísticos atraentes. Para alcançar resultados

ainda mais expressivos, recomenda-se aos professores que se aprofundem não apenas no Construct 3, mas também na composição artística (mosaicos, personagens, animações) e na inclusão de recursos de áudio (trilhas sonoras, efeitos sonoros). Essas possibilidades tornam a experiência mais envolvente e podem inspirar novas eletivas e projetos interdisciplinares, envolvendo Artes, Literatura, Língua Inglesa ou qualquer outra área que os alunos e docentes julguem pertinente integrar.

Apesar dos resultados positivos, algumas limitações se manifestaram ao longo da aplicação. Em primeiro lugar, embora seja possível criar e editar jogos diretamente no celular, o processo mostrou-se mais demorado e exige maior paciência tanto do professor quanto dos estudantes. Além disso, o Construct 3 necessita de acesso à internet para ser utilizado, o que pode restringir o uso em contextos com infraestrutura limitada. Por fim, embora o ideal seja trabalhar em computadores, é compreensível que algumas escolas tenham laboratórios pequenos e precisem compartilhar máquinas. Esse aspecto pode, de certo modo, estimular o trabalho em equipe, mas ainda assim demanda uma organização cuidadosa das turmas e dos tempos de uso.

Em síntese, a proposta de aliar jogos didáticos à educação matemática demonstra-se potente para tornar as aulas mais dinâmicas, aproximar os alunos dos conteúdos curriculares e incentivar a aprendizagem de forma prazerosa. A convergência entre tecnologia, arte e pedagogia revela-se, portanto, como um caminho fértil para o desenvolvimento de práticas inovadoras em sala de aula, reforçando a capacidade do professor de reinventar abordagens e de despertar o protagonismo estudantil — o que se reflete diretamente no engajamento dos discentes e na ampliação das oportunidades de aprendizagem, seja dentro ou fora do ambiente escolar.

Referências Bibliográficas

[1] ASEPRITE. Animated sprite editor & pixel art tool. https://www.aseprite.org/. Acesso em: 30 dez. 2024.

[2] E. BIANCHINI. *Matemática Bianchini 7º ano: manual do professor*. Moderna, São Paulo, 2022.

[3] BRASIL. Ministério da Educação. *Base Nacional Comum Curricular*. Brasília, DF, 2018. http://basenacionalcomum.mec.gov.br/. Acesso em: 26 out. 2024.

[4] N. G. CARR. *The shallows: what the internet is doing to our brains*. W. W. Norton & Company, New York, 2010.

[5] CONSTRUCT. Construct 3 manual: system expressions: choose. https://www.construct.net/en/make-games/manuals/construct-3/system-reference/system-expressions. Acesso em: 14 out. 2024.

[6] CRAFTPIX. Game assets store. https://craftpix.net/. Acesso em: 30 dez. 2024.

[7] L. ETCUBAN, J.; PANTINOPLE. The effects of mobile application in teaching high school mathematics. *International Electronic Journal of Mathematics Education*, 13, 2018.

[8] H. HATTIE, J.; TIMPERLEY. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007.

[9] J. F. e. a. HUGHES. *Computer graphics: principles and practice*. Addison-Wesley, Boston, 3. ed. edition, 2013.

[10] ITCH.IO. Game assets. https://itch.io/game-assets. Acesso em: 30 dez. 2024.

[11] D. KAFMAN, B.; MOSS. The role of technology in constructivist classrooms. *Journal of Technology and Teacher Education*, 17(4):327–346, 2009.

[12] KRITA. Free and open source digital painting software. https://krita.org/. Acesso em: 30 dez. 2024.

[13] E. L. LIMA. *Coordenadas no plano com as soluções dos exercícios: geometria analítica, vetores e transformações geométricas*. INEP, Brasília, 4. ed. edition, 2002. Colaboração de Paulo Cezar P. Carvalho. Publicado no âmbito do programa Publicações de Apoio à Formação Inicial e Continuada de Professores.

[14] E. L. LIMA. *Números e funções reais*. IMPA, Rio de Janeiro, 1. ed. edition, 2014.

[15] J. MATTAR. *Games em educação: como os nativos digitais aprendem*. Pearson, São Paulo, 2013.

[16] NYKNCK. City pack: top down - pixel art. https://nyknck.itch.io/citypackpixelart. Disponível gratuitamente. Acesso em: 30 dez. 2024.

[17] OPENGAMEART. Free game assets. https://opengameart.org/. Acesso em: 30 dez. 2024.

[18] S. PAPERT. *Mindstorms: children, computers, and powerful ideas*. Basic Books, New York, 1980.

[19] S. PAPERT. *The connected family: bridging the digital generation gap*. Longstreet Press, Atlanta, 1996.

[20] S. PAPERT. *Mindstorms: children, computers, and powerful ideas*. Basic Books, New York, 2. ed. edition, 1996.

[21] PISKEL. Online sprite editor. https://www.piskelapp.com/. Acesso em: 30 dez. 2024.

[22] M. PRENSKY. Digital natives, digital immigrants. *On the Horizon*, 9(5):1–6, 2001. Disponível em: https://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf. Acesso em: 26 out. 2024.

[23] Scirra Ltd. Objects - Construct 3 Manual. https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/objects, 2024. Acesso em: 13 out. 2024.

[24] A. SILVEIRA, R.; CALDEIRA. Resolução de problemas matemáticos nos anos iniciais do ensino fundamental: uma investigação com professores polivalentes. *Educação em Perspectiva*, 31(58):760–776, 2017. Disponível em: https://www.scielo.br/j/epec/a/GkqWNWzBv89F6wj3vcq9xsG/?format=pdf. Acesso em: 26 out. 2024.