MowerCard — Um Simulador de Cortador de Grama com Trajetos Programados por Cartões QR Code

Raul César Alves Magalhães, Deiviston S Aguena

Universidade Federal de Mato Grosso do Sul – Campus Coxim
Curso Sistemas de Informação

1 de dezembro de 2025

Resumo

Em diversas instituições de ensino, nota-se que parte dos acadêmicos, apresentam dificuldades no aprendizado de lógica de programação, que é uma das bases fundamentais em cursos voltados a área de Computação e tecnologia da informação. A partir desse problema, notou-se que um dos principais fatores relacionados é a dificuldade de abstração de conceitos de programação, como variáveis e funções. Como forma de contornar este problema, diversas metodologias tem sido utilizadas no ensino, desde as que trazem abordagens lúdicas até as que utilizam a lógica tangível, envolvem robótica computacional ou jogos digitais interativos, buscando tornar o processo de aprendizado mais engajador. Seguindo o mesmo sentido, apresentamos o *MowerCard*, um aplicativo para dispositivos móveis que oferece um cenário virtual onde um robô cortador de grama é programado por meio da leitura de cartões *QR Code*, uma abstração do conceito de programação com cartões perfurados. No aplicativo, também são utilizados recursos de Realidade Aumentada para aumentar o engajamento dos estudantes. Espera-se que o uso do aplicativo possa auxiliar o aprendizado de lógica de programação como uma abordagem alternativa.

1 Introdução

A lógica de programação constitui a base fundamental para o aprendizado em cursos voltados para a área de Computação e áreas tecnológicas. A partir dela, o estudante desenvolve o raciocínio lógico e estruturado, fundamental para compreender e construir algoritmos capazes de solucionar problemas de maneira sistemática.

Contudo, nas instituições de ensino, em diferentes níveis acadêmicos, observa-se um desafio recorrente: os alunos que estão no início da graduação têm dificuldades no aprendizado dessa matéria, essencial para o desenvolvimento acadêmico e profissional. De acordo com Netto *et al.* (2017), entre os principais fatores que contribuem para isso estão a dificuldade de abstração e o predomínio de metodologias tradicionalistas, que tornam o processo de aprendizagem rígido e pouco estimulante.

1.1 Uso de metodologias lúdicas no processo de aprendizagem

O uso de metodologias lúdicas, como jogos digitais, tem se mostrado eficaz para reduzir a abstração no ensino de lógica de programação e aumentar o engajamento dos alunos (SILVA *et al.*, 2021).

Em paralelo as metodologias lúdicas, a robótica educativa tem se apresentado como uma estratégia eficaz para transformar o aprendizado, aumentando sua motivação com o uso de lógica tangível. Neste contexto, a lógica tangível permite que o estudante manipule objetos físicos e visualize os efeitos diretos de suas ações (CAMBRUZZI; SOUZA, 2015). A partir disso, podemos determinar que o uso de recursos visuais e interativos pode transformar a forma de aprendizado, deixando o ensino mais concreto e engajador.

Em uma abordagem complementar à robótica educativa, mas focada em conceitos puramente digitais, Rodrigues (2022) propõe um simulador tridimensional para o problema do Jantar dos Filósofos. O autor aplica a ludicidade para auxiliar no aprendizado de sincronização de processos, um dos tópicos classicamente abstratos da disciplina de Sistemas Operacionais.

Em um exemplo de software lúdico que combina Realidade Aumentada com interação tangível, SUKEYOSI, W. A. P. e Aguena (2015) apresentam jogos educacionais para crianças. A solução explora a resposta a questionários de múltipla escolha sem o tradicional uso de cliques em uma tela, mas por meio de marcadores físicos, onde a criança seleciona a resposta ao realizar a oclusão de um marcador correspondente à alternativa correta, conectando assim uma ação tangível a um resultado no ambiente digital.

1.2 A Realidade Aumentada

A Realidade Aumentada (RA), impulsionada pelos avanços da multimídia e da Realidade Virtual, enriquece o ambiente físico com objetos do mundo virtual, levando a implementação de interações lúdicas para outro nível. Diferente da Realidade Virtual, que transporta o usuário para um ambiente virtual, a Realidade Aumentada mantém o usuário em seu ambiente físico, trazendo mais facilidade para a interação com objetos

do mundo virtual de maneira mais natural (KIRNER *et al.*, 2006).

Diante do desafio do aprendizado da lógica e da programação, neste trabalho propomos o *MowerCard - Um Simulador de Cortador de Grama com Trajetos Programados por Cartões QR Code*, com o objetivo de aplicar conceitos de lógica de programação de maneira prática, visual e lúdica.

2 Organização do Texto

Além da Seção 2, este artigo é composto por outras quatro seções. Na Seção 1 é contextualizado o problema no ensino de lógica de programação para acadêmicos no início da graduação, e propomos a implementação do *MowerCard*. Na Seção 3 são apresentadas as ferramentas usadas na implementação deste trabalho, além de relatar como aconteceu o processo de desenvolvimento. Na Seção 4 apresenta-se o produto resultante, desde as telas criadas, o fluxo de funcionamento da aplicação, os dados registrados, até suas limitações. Por fim, na Seção 5 é apresentada conclusão do trabalho.

3 Metodologia

Até meados da década de 1970, a principal forma de acesso a computadores era realizada por meio de cartões perfurados — cartões físicos de papel que continham furos inseridos manualmente por programadores, a fim de serem interpretados como instruções ou dados pelos computadores por meio de um leitor de cartões (CRUZ, 2001).

Inspirado neste conceito, no primeiro semestre do ano de 2022, durante a disciplina Introdução à Computação, ministrada à época pelo professor Deiviston S Aguena, do curso de Sistemas de Informação da Universidade Federal de Mato Grosso do Sul (UFMS), do campus de Coxim-MS, foi proposta uma atividade com a intenção de introduzir conceitos de algoritmos e lógica de programação aos calouros do curso, sem a necessidade de envolver conceitos de variáveis, funções ou qualquer sintaxe de linguagem de programação. A atividade consistia em apresentar ao aluno um tabuleiro e a figura do robô cortador de grama, que ocupava uma posição inicial dentro desse tabuleiro. Além disso, algumas posições no tabuleiro continham gramados que deveriam ser cortados pelo robô cortador de grama, outras posições do tabuleiro continham obstáculos que precisavam ser evitados. Para resolver a atividade, o aluno consultava uma tabela de comandos de ação para o cortador de grama, como mover-se adiante, virar à esquerda, virar à direita, ligar ou desligar a hélice; então, ele deveria preencher um gabarito com os comandos selecionados, na mesma ordem em que deveriam ser executados pelo robô cortador de grama, simulando assim o funcionamento da programação com o uso de cartões perfurados (AGUENA, 2022).

Apesar de a tarefa motivar o aluno a completar a atividade utilizando os princípios da lógica de forma lúdica, na prática, envolvia um processo manual que se mostrava demorado e trabalhoso. Igualmente, o processo de correção dos gabaritos exigia um tempo considerável, além de estar sujeito a erros por ser realizado de maneira manual. Outra observação era que os gabaritos utilizados nesta atividade sempre eram descartados após a utilização, não podendo ser reutilizados.

Visando aprimorar esta atividade e melhorar o processo de aprendizado, foi proposta a implementação de um aplicativo para dispositivos móveis, em que a entrada dos comandos por "cartões perfurados" (ou "gabarito de comandos") fosse substituída pela leitura de cartões em *QR Code*, e o trajeto programado pelo usuário fosse simulado em um ambiente virtual de maneira automática. Com este aplicativo, utilizaríamos a lógica tangível para aumentar o engajamento e eliminaríamos tanto a utilização de material impresso descartável para anotar as respostas quanto a etapa de correção manual, demorada e sujeita a erros.

3.1 Tecnologias Pesquisadas

Para a implementação do *MowerCard*, foram considerados três aspectos principais: desenvolvimento direcionado a dispositivos móveis; ferramentas para desenvolvimento em computação gráfica; e integração com RA, especialmente para a leitura de marcadores. Para selecionar a tecnologia usada para o desenvolvimento, foram realizadas pesquisas nas páginas oficiais da documentação de cada tecnologia sugerida, e a implementação de protótipos simples para garantir que essas tecnologias seriam capazes de suprir as necessidades que o projeto demanda.

3.1.1 Python e OpenCV(Marcadores ArUco)

No desenvolvimento do primeiro protótipo implementado para a lógica de leitura de cartões, foi escolhida a linguagem Python por sua simplicidade e por ter suporte nativo à *OpenCV*, biblioteca *open source* para soluções envolvendo visão computacional (OPENCV, 2025) que também conta com um módulo para lidar com marcadores *ArUco*, uma biblioteca desenvolvida para detecção de marcadores fiduciais quadrados (ARUCO, 2025).

O protótipo foi concluído com um funcionamento simples que envolvia ler um marcador *ArUco* pela câmera do computador — como podemos notar na Figura 1 — e, a partir de um dicionário (estrutura de dados) definido no código, traduzir para uma determinada ação, que representaria um dos comandos para o robô cortador.

Apesar de o protótipo estar funcional, foram identificados alguns problemas com esta abordagem, o principal deles sendo a dificuldade em portar a aplicação para dispositivos móveis, já que não há muitas abordagens convencionais e otimizadas para empacotar o software em Python e *OpenCV* de maneira direta para Android. Além de os resultados não serem muito eficientes, a biblioteca de interface gráfica utilizada neste protótipo (*TKinter*) é bastante limitada para imple-

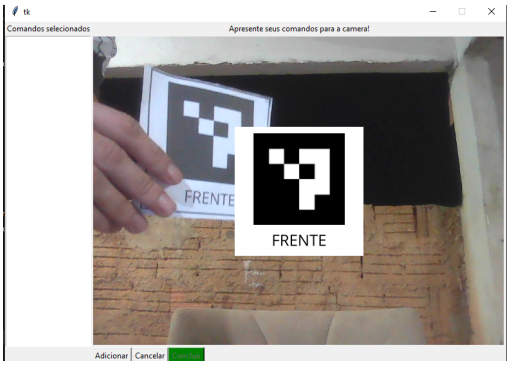

Figura 1: Demonstração do protótipo desenvolvido com Python e *OpenCV* identificando um marcador *ArUco* através da *webcam*.

mentar a simulação do robô cortador em computação gráfica. Uma solução considerada seria escolher outra biblioteca mais voltada para o desenvolvimento de jogos, como a *PyGame*, porém, ainda seria mais limitada que as demais plataformas a serem apresentadas a seguir.

3.1.2 Unity e Vuforia

Já pensando em suprir a necessidade de desenvolver para Android e com mais facilidade para implementar soluções de computação gráfica, a próxima ferramenta a ser considerada foi a Unity, uma das principais *game engines* — ferramenta de software para desenvolvimento de ambientes gráficos e interativos, especialmente para jogos digitais. A Unity apresenta consigo suporte para desenvolvimento multiplataforma, incluindo o Android SDK (*Software Development Kit*) para publicar diretamente em dispositivos móveis, além de ser uma ferramenta consolidada para desenvolvimento de jogos no mercado com um vasto portfólio (UNITY, 2025e).

Para integrar com a Unity na parte de RA, foi considerada a *Vuforia*, ferramenta proprietária para desenvolvimento de RA, que possui um SDK para Unity (VUFORIA, 2025).

Para testar a ferramenta e como ela poderia ser implementada no projeto, um protótipo semelhante ao desenvolvido em Python foi proposto. Mas, além de ler os marcadores, o protótipo também deveria possibilitar instanciar objetos virtuais na visão da câmera do usuário.

Contudo, o protótipo não foi bem sucedido, devido a barreiras impostas pela licença de uso da *Vuforia*, que neste caso foi utilizada uma licença de uso gratuita que apresentava marca-d'água e limitações de funcionalidades que barravam o progresso na implementação.

A falta de flexibilidade com a *Vuforia* fez com que o projeto abandonasse soluções envolvendo SDKs proprietários, seguindo com a próxima plataforma que se mostrou mais flexível.

3.1.3 Unity e AR Foundation

Buscando mais flexibilidade para trabalhar dentro da Unity com tecnologias de RA, foi sugerido o uso da *AR Foundation* — biblioteca para desenvolvimento de aplicativos em RA com suporte à multiplataforma, através de *AR SDKs* nativos (UNITY, 2025a) — por ser desenvolvida e mantida pela própria Unity. Com ela, teríamos todos os aspectos considerados para a implementação do projeto: com a Unity, ter a possibilidade de desenvolver uma interação gráfica para simular o trajeto do robô cortador e exportar o projeto para dispositivos móveis de maneira direta; E com a *AR Foundation* ter suporte nativo a plataformas de RA.

Mais uma vez, utilizando o recurso de *image tracking* da *AR Foundation*, foi implementado um protótipo para ler e identificar cartões através da câmera do dispositivo, que trouxe resultados positivos tanto no processo de desenvolvimento, quanto no funcionamento da aplicação.

3.2 Tecnologia selecionada

A partir do resultado do último protótipo desenvolvido na Unity com a *AR Foundation*, se iniciou o desenvolvimento do *MowerCard* na *engine* Unity, versão 2022.3.58f1, com o auxílio da biblioteca *AR Foundation*, versão 5.1.5, na linguagem de programação C#.

3.3 Terminologia da Unity

Antes de prosseguir com o relato do desenvolvimento do aplicativo, apresentamos algumas terminologias, a fim de trazer contextualização ao leitor nas próximas subseções.

***Scene* (cena)** São os níveis ou telas de um jogo. Nela estão presentes *GameObjects* organizados de maneira que determinam o funcionamento da mesma.

***GameObject* (objeto)** Todo ator dentro de uma cena. Funciona como um contêiner de funcionalidades (componentes).

***Component* (componente)** Atribuem funcionalidade (comportamento) nativas aos *GameObjects*.

Script Um tipo específico de *Component*. Atribuem funcionalidades criadas pelo desenvolvedor aos *GameObjects*.

Transform Um *Component* padrão que todo *GameObject* possui. É ele que atribui valor de posicionamento de um objeto num espaço, além de suas dimensões e rotação.

Grid Um componente da Unity fundamental que provê um *layout* estruturado de linhas e pontos espaçadas uniformemente. Ideal para a criação de *Tilemaps*.

Tilemap Sistema projetado para simplificar a criação de mapas, níveis 2D, e ambientes baseados em *grid*.

Prefab Um *GameObject*, completo com seus *Components*, salvo em forma de modelo que pode ser clonado e instanciado dentro de uma cena.

***Hierarchy* (hierarquia)** Uma estrutura em forma de árvore que representa tudo que está instanciado dentro de uma cena. Define relações pai/filho entre objetos, crucial para organização e definir comportamento em grupo.

Canvas Componente do sistema de *User Interface* (UI) que tem como objetivo abstrair o espaço onde os componentes de UI, como botões, caixas de texto, etc. serão posicionados na cena.

Asset Todo arquivo que faz parte do projeto, podendo representar um *Script*, um *Sprite*, um áudio, um modelo 3D, etc.

Sprite Qualquer imagem 2D (bidimensional) para representar algum objeto no jogo, seja um personagem, obstáculo ou algum componente de UI.

Tile Um *sprite* usado como peça repetível na construção de níveis.

3.4 O Simulador *MowerCard*

Já com as tecnologias a serem usadas devidamente conhecidas, o primeiro passo para a implementação do projeto foi fazer o simulador. Este módulo tem como objetivo ser o equivalente virtual do tabuleiro anteriormente impresso em papel, onde está disposta a figura do robô cortador e os componentes do cenário, como a grama e os obstáculos.

Neste estágio de desenvolvimento, foi criada a cena "*SimulatorTester*", para ser usada como um ambiente para desenvolver e testar os objetos relacionados somente ao funcionamento do simulador. Note a importância de cada objeto ser contido, de modo que cada um tenha responsabilidades individuais e funcione de forma independente.

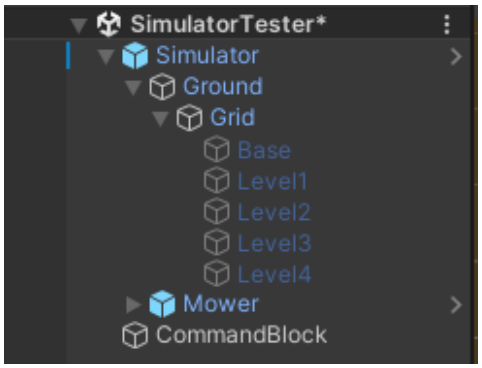

Figura 2: Hierarquia da cena *SimulatorTester*

Na hierarquia da cena, como podemos ver na Figura 2, temos o objeto *Simulator*, que funciona como contêiner onde temos contido um objeto *Ground* — que representa o chão onde os outros objetos serão dispostos — que tem como filho um objeto *Grid* (Grade), com o componente homônimo, apresentando uma estrutura em grade para posicionar os elementos do tabuleiro em posições fixas, e assim construir os níveis que os jogadores deverão completar. Filiado a essa grade, foram criados *Tilemaps*, cada um representando um nível diferente.

Também contido em *Simulator*, temos o objeto *Mower*, que representa o robô cortador. Nele foi atribuído o *Script* "*MowerController.cs*" que controla a lógica do comportamento do cortador, desde a execução de suas ações de movimento, como a rotação e a verificação de colisão com obstáculos, até a ação de cortar a grama.

Por fim, temos, no mesmo nível do objeto *Simulator*, o objeto *CommandBlock*, que contem um *Script* de mesmo nome, onde está implementada a estrutura de fila em que os comandos a serem lidos serão enfileirados e executados pelo cortador.

A fim de testar o funcionamento do *Mower*, foi criado um *Canvas* com botões que representam cada ação que o cortador realiza, com o intuito de testar manualmente as ações do cortador e verificar o resultado das interações entre os objetos na cena. Paralelamente, também foi testado o funcionamento do *CommandBlock*, usando uma fila de comandos com valores fixos a fim de testar o processamento e a execução de comandos sequencialmente.

3.4.1 Desafios encontrados

Usar os recursos de *Grid* e *Tilemaps*, trouxe um desafio para o desenvolvimento, já que essas ferramentas fazem parte do pacote de desenvolvimento 2D, e este projeto estava sendo desenvolvido em um ambiente 3D. Com isso, os *Tilemaps* que normalmente são usados para construir níveis usando *assets* bidimensionais, como *Sprites*, poderiam não ser apropriados para um mapa tridimensional. Para contornar este imprevisto, foi utilizado o recurso de *Rule Tiles*, com ele foi possível usar *Tiles* 2D para desenhar os níveis, e usando um *Script*, varrer o *grid* como se fosse um vetor bidimensional, e traduzir cada *tile* para um objeto 3D, que poderia ser ou um bloco de grama ou uma pedra (obstáculo) (UNITY, 2025c).

Outro desafio foi durante a movimentação do robô cortador, que deveria ser relativo ao tabuleiro. Para manter isso correspondente, foi atribuído ao objeto *Ground* um componente de *Mesh Renderer* — componente para renderizar malhas 3D — para renderizar um modelo de plano 3D. Isso proporcionou uma visualização do chão além de agora ter dimensões mais explícitas. A partir disso o plano pôde ser referenciado pelo *script* controlador do *Mower*, que assim pode atualizar a posição e direção do robô cortador relativo a suas direções *forward* e *right*.

Por fim, no *CommandBlock*, foi necessário implementar uma maneira de executar os comandos assincronamente. Para isso, foi projetado um sistema de disparo e espera de eventos, onde o bloco de comandos esperava pelo evento *OnMovementFinished*, que era disparado sempre que o robô cortador finalizava a execução de um comando. Deste modo, o bloco de

comandos executava um comando por vez, esperando o fim do comando anterior para executar o próximo.

Com estes testes finalizados e assegurado de que o simulador está funcional, o próximo passo é implementar o enfileiramento de comandos através da leitura de cartões, via *AR Foundation*.

3.5 Leitura dos Cartões

Para começar a implementação da mecânica de leitura de cartões, foi utilizada como base a cena utilizada para o protótipo feito na fase de seleção de tecnologias. Nesta cena, além dos objetos que foram construídos na cena do simulador, há alguns objetos específicos para o funcionamento de uma cena RA na Unity. *XR Origin* (ou *AR Rig*), que serve para representar o dispositivo do usuário no mundo virtual; e o *AR Session*, que serve como o cérebro que processa toda lógica de RA dentro da cena, desde detecção de planos até rastreamento de imagens. Para resolver este problema, o componente que buscamos é o *ARTrackedImageManager*, presente no *XR Origin*.

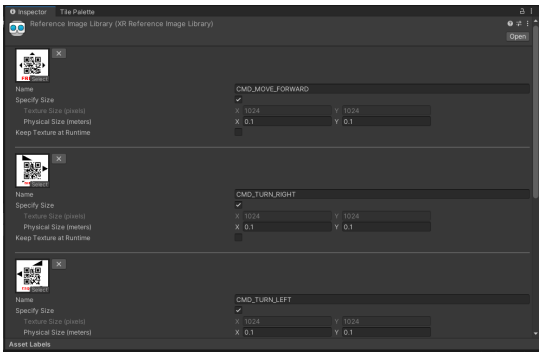

Figura 3: Referenciando imagens a serem lidas no *Reference Image Library*

Com o *ARTrackedImageManager*, o primeiro passo é criar um *Reference Image Library* (biblioteca de referência de imagens) — uma espécie de dicionário, onde será referenciado cada imagem que a *AR Foundation* será capaz de detectar no mundo real. Como se pode notar na Figura 3, foram registradas 3 imagens na biblioteca, e para cada uma foi especificado um nome (campo *Name*), além de um tamanho em metros para especificar as dimensões físicas que esperamos que a imagem tenha no mundo real, que neste caso foi atribuído 0,1 m (10 cm).

Para lidar com a lógica de leitura de imagens, atribuímos um *Script* "*ArCardTrackingManager.cs*", que referencia o componente *ARTrackedImageManager*. Nele, inscrevemos o método *OnImageChanged* para escutar o evento *trackedImagesChanged*, que é disparado sempre que *tracked image manager* realiza a leitura de alguma imagem (UNITY, 2025b). O método *OnImageChanged*, recebe como parâmetro 3 listas: *added*, contendo as imagens que foram lidas pela primeira vez; *updated*, contendo imagens que já haviam sido lidas, atualizando seu estado atual; e *removed* que, em teoria, contém as imagens que não estão mais sendo lidas, mas isso não acontece na prática então sua existência pode ser ignorada. Quando uma imagem é lida, o primeiro passo é identificar qual comando ela representa a partir do nome que lhe foi dado na biblioteca de referência. Para isso, foi declarado um *Dictionary* que mapeia cada nome ao seu respectivo comando.

Recapitulando o funcionamento da leitura de cartões: o *ARTrackedImageManager* realiza a leitura de um cartão e dispara o evento *trackedImagesChanged*. Com isso, o *Script ArCardTrackingManager.cs*, que estava esperando esse disparo, identifica qual cartão foi lido por meio do nome que lhe foi atribuído na biblioteca de referência. A partir dessa informação, um *Dictionary* é consultado e o comando é finalmente enviado para a fila (*CommandBlock*).

3.5.1 Desafios encontrados

Durante o desenvolvimento dessa funcionalidade, o principal desafio foi implementar uma lógica de tempo de espera para que não fossem lidos múltiplos comandos de uma só vez, já que o evento de leitura poderia ser disparado várias vezes enquanto o cartão estivesse visível pela câmera. Para isso, foi criado o *Dictionary commandLastReadTime*, que associava a identificação do comando com um valor numérico que representa quantos segundos fazem que o cartão está sendo lido, sendo este zerado sempre que não estivesse mais sendo lido. Com isso, foi estabelecido um intervalo mínimo em que um cartão deveria ser lido até efetivamente registrar o comando na fila, que no projeto está definido o valor padrão de 1,5 segundos.

Após finalizar a implementação da leitura de cartões com RA, o próximo passo foi definir a estrutura dos comandos em si, para serem enfim executados pelo *CommandBlock*.

3.6 Processamento dos Comandos

Durante a implementação da fila de comandos foi utilizado o recurso *ScriptableObjects*, da Unity, para representar os comandos, determinando um modelo de dados e comportamento que pode ser compartilhado entre vários objetos e componentes no projeto de maneira desacoplada (UNITY, 2025d). Neste contexto, um *ScriptableObject* "Comando" foi definido com atributos para identificar o comando por nome e imagem, além de um método *Execute*, onde será implementada a ação executada por cada comando. Herdando a classe deste modelo, foram criados outros *ScriptableObjects*, um para cada comando que o Cortador pode executar, com seus respectivos atributos e comportamento. Desta maneira, o processo de modificação ou adição de novas ações para o robô cortador se torna mais prático e escalável.

3.7 Registrando resultados dos jogadores

Para auxiliar com possíveis futuros experimentos com a aplicação, foi proposto que o *app* fosse capaz de re-

gistrar os resultados de cada jogador que testasse a aplicação. Para isso, foi idealizado que cada jogador deveria iniciar uma sessão, que guardaria informações do jogador, como faixa etária e sexo, além de salvar o resultado do jogador em cada tentativa no simulador.

Para que isso fosse possível, foi criado um esquema de telas para que o jogador pudesse navegar entre: menu inicial — onde é apresentado a tela de título; tela de cadastro do usuário — onde o jogador inicia sua sessão; e tela de jogo — onde acontece a simulação. Neste sentido, a aplicação segue um fluxo em que o jogador inicia a sessão informando os dados necessários, realiza quantas tentativas lhe for conveniente no simulador, das quais cada uma terá o resultado registrado e por fim, quando o jogador preferir, encerrar a sessão e ter seus resultados salvos em um arquivo no dispositivo. Uma vez que a sessão é encerrada, ela não pode ser aberta novamente e para cada sessão, um arquivo é criado.

Para registrar os resultados, foi feito uso de *JSON* (*Javascript Object Notation*), um formato leve e de fácil leitura, utilizado para armazenamento e transmissão de dados estruturados. O uso desse formato permitiu organizar as informações de cada sessão — como *id* do jogador, sexo, faixa etária e os resultados de seus testes — em uma estrutura hierárquica simples e fácil de ser interpretada, possibilitando que sejam usados para análise posterior, mantendo a integridade e a legibilidade das informações coletadas durante os testes.

Devido às novas políticas de segurança implementadas nas versões mais recentes do *Android*, que não permitem ao aplicativo acesso direto ao armazenamento interno dos dispositivos, foi necessário implementar um *plugin* para lidar com *Storage Access Framework* (SAF) — permite aplicativos acessarem arquivos locais no dispositivo de acordo com a preferência do usuário (ANDROID, 2025) — para persistir os arquivos *JSON* referentes aos resultados das sessões no dispositivo do jogador. O *plugin*, implementado nativamente em *Java*, atua como uma ponte entre a *Unity* e as APIs do sistema operacional, permitindo solicitar ao usuário acesso a um diretório específico, e posteriormente manter esse acesso de forma persistente. Por meio do SAF, o *plugin* obtém um URI (*Uniform Resource Identifier*) autorizado pelo usuário e o converte em uma instância de *DocumentFile*, possibilitando operações de leitura e escrita sem violar as políticas de segurança do *Android*.

Dessa forma, os dados registrados possibilitam que, futuramente, se realizem estudos de caso com a ferramenta e, a partir dos resultados armazenados, possa-se fazer uma análise detalhada das informações.

3.8 Visualização em Realidade Aumentada

Com o objetivo de melhorar a experiência do jogador, foi implementada a funcionalidade de visualização da leitura do cartão com RA. Para isso, foi utilizado o mecanismo de leitura de imagens com *ARTrackedImageManager*, o mesmo utilizado na leitura dos cartões. Com o disparar do evento *trackedImagesChanged*, um objeto é instanciado na posição do cartão e mostrado na visão em RA na câmera do usuário. Este objeto apresentado ao usuário torna mais explícito o processo de leitura do cartão, fornecendo um retorno visual do progresso da leitura do cartão, que, quando concluído, apresenta uma animação — também no ambiente de RA — ao usuário.

A implementação desta funcionalidade oportunizou um melhor aproveitamento das tecnologias de RA da *AR Foundation*, trazendo maior integração entre o mundo físico e o mundo virtual através dos olhos do usuário. Além disso, melhorou-se a usabilidade do sistema de leitura de cartões, uma vez que o usuário terá uma noção melhor se a leitura do cartão está realmente acontecendo ou se o cartão sequer está sendo reconhecido pelo sistema RA, além de tomar conhecimento do intervalo de tempo que ocorre entre as leituras de cada cartão.

3.9 Uso de *assets* externos

Para a composição visual da aplicação, foram utilizados *assets* gráficos de terceiros licenciados sob uso *royalty-free* e *Creative Commons (2025)* (CC). As artes utilizadas para a composição da *UI* foram produzidas por Penzilla DESIGN (2025) e são utilizadas de acordo com os termos da licença do autor.

O modelo 3d utilizado para representar o cortador de grama, o *"Lawn Mower"*, foi produzido por 2025668 (2025). E o modelo 3d utilizado para representar os obstáculos, neste caso, *as pedras*, foi o modelo *"Piedra"*, produzido por Dacoher (2025). Ambos os modelos são utilizados de acordo com os termos da licença CC.

4 Resultados e Discussão

Nesta seção, será apresentada a versão do produto resultante do desenvolvimento final deste trabalho, o *MowerCard*. Antes, porém, destacamos que resultados parciais foram previamente apresentados no 1º Simpósio Internacional de Conexões Pantaneiras: Tecnologia, Saúde, Direito, Cultura e Linguagens (AGUENA *et al.*, 2025).

O *MowerCard* consiste em um aplicativo para dispositivos móveis *Android* desenvolvido com *Unity*. A seguir, todos os testes relatados foram realizados em um dispositivo *Samsung* modelo SM-S911B, rodando no sistema operacional *Android 14*.

4.1 Fluxo da aplicação

Ao abrir o aplicativo, a primeira tela apresentada ao usuário é o seletor de arquivos nativo do sistema operacional, mostrada na Figura 4. Invocado pela interface de armazenamento, permite ao usuário escolher em que diretório o aplicativo irá registrar os resultados de cada sessão.

Figura 4: Seletor de arquivos nativos do Android, invocado pelo SAF.

Com o diretório selecionado, o jogador é direcionado à tela inicial de título *MowerCard - v1*, apresentado no primeiro quadrante da Figura 5. Aqui temos uma imagem renderizada do modelo do cortador de grama para fins estéticos. Compondo a UI, temos o titulo da aplicação, um botão com o rótulo "jogar" e, no rodapé, um informativo sobre as licenças de uso de *assets* de terceiros.

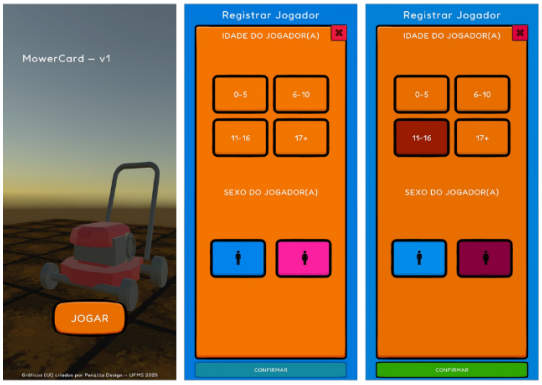

Figura 5: Telas da cena menu principal.

O botão "jogar" leva o jogador à tela de cadastro, apresentada nos demais quadrantes da Figura 5. Na ocasião apresentada, o usuário em questão está se cadastrando como um indivíduo do sexo feminino, com idade entre 11 e 16 anos. Ao informar os dados, o usuário habilita o botão "confirmar", apresentado no rodapé da tela. Este botão tem o intuito de confirmar as informações inseridas e dar início a sessão do jogador, levando-o à tela de jogo apresentada na Figura 6.

Na tela de jogo são apresentados três componentes principais: o cabeçalho, a visão da câmera, e a visualização do mapa de jogo.

O cabeçalho está dividido em duas partes principais: o contêiner de botões, localizado na parte de cima, e abaixo o *carousel* que apresenta a fila de comandos lidos pelo usuário.

No contêiner de botões está presente um componente de texto, identificando qual o nível que o jogador está jogando atualmente. Ademais, outros três botões que possibilitam o jogador a gerenciar a fila de comandos.

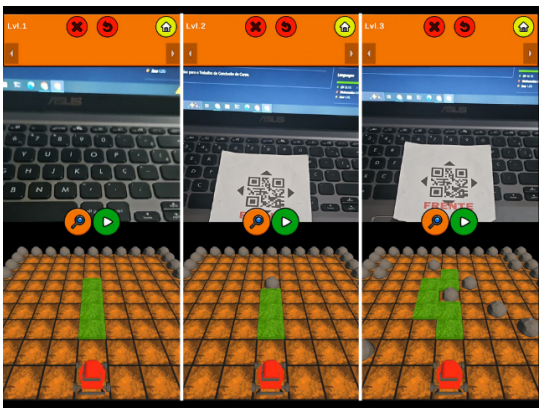

Figura 6: Da esquerda para a direita: Nível 1, nível 2 e nível 3.

Em vermelho com um ícone em forma de 'x', o botão "limpar", que possibilita o jogador a limpar a fila de comandos e começar a leitura novamente, caso necessário; também em vermelho e com o ícone de meia-volta, o botão "desfazer", que como o nome indica, serve para remover a última entrada da fila de comandos, desfazendo as últimas leituras do jogador; Em amarelo, o botão "*home*" (início), que possibilita ao jogador encerrar sua sessão e voltar à tela inicial. Uma caixa de diálogo é apresentada, confirmando a intenção do jogador.

Já o *carousel* — componente interativo que exibe múltiplos conteúdos em sequência, permitindo a navegação do usuário — é apresentado junto a dois botões em forma de seta, um em cada extremidade, permitindo ao usuário navegar entre os itens ali dispostos.

Ao fundo, é mostrada a visão da câmera do usuário. A partir dali o usuário pode capturar os cartões e visualizar o processo de leitura em RA. Pode-se dizer que este é o elo entre o mundo físico e o mundo virtual, nesta aplicação.

Na parte inferior da tela é apresentada a visualização do mapa de jogo, em forma de um menu flexível. Este componente pode ser apresentado em dois estados: aberto ou fechado. Junto a ele, temos dois botões: um verde com ícone de reprodução para, uma vez lidos os comandos, iniciar a simulação; e o laranja com simbolo de lupa, que tem como objetivo alternar entre os estados aberto e fechado do mapa, como apresentado na Figura 7.

Quando no estado aberto — seu estado inicial — o mapa de jogo se faz presente, cobrindo cerca de 50% da tela (observe no quadrante da direita da Figura 7). Aqui, o jogador poderá analisar o desafio e assim elaborar sua sequência de comandos que deverão ser lidos para resolvê-lo, ou seja, cortar toda a grama do mapa sem colidir com nenhum obstáculo. Enquanto neste estado, o jogador não pode realizar a captura dos comandos, mas apenas dispor-los em sequência para realizar a leitura, como é exemplificado na Figura 8. Uma vez concluída a sequenciação dos comandos, o jogador deve pressionar o botão da lupa que altera o estado do mapa para fechado. Desta forma, a visão da câmera será ampliada ao ocultar a visualização do mapa (observe no

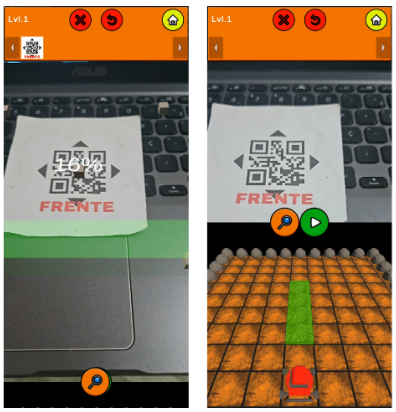

Figura 7: A visualização do mapa de jogo pode alternar entre aberto e fechado.

quadrante da esquerda da Figura 7).

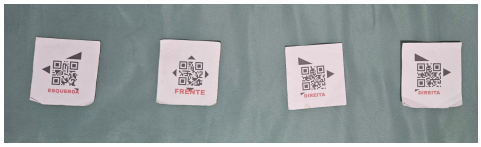

Figura 8: Exemplo de cartões dispostos para serem lidos, respeitando certa distância entre eles.

4.2 Realizando leitura dos cartões

Uma vez que os cartões foram sequencialmente dispostos pelo jogador, ele pode iniciar com o processo de leitura dos comandos. Para isso, ele deve colocar a visualização do mapa no estado fechado, e apontar a câmera para cada comando, respeitando a sequência e o tempo de leitura de 1,5 segundo entre cada comando. Para auxiliá-lo neste processo, um *feedback* animado irá ser mostrado em ambiente de RA, como é demonstrado na Figura 9. Assim que a leitura de um cartão é concluída, além do *feedback* em RA, o ícone correspondente ao comando lido será anexado ao *carousel* presente no cabeçalho.

Figura 9: Alguns quadros demonstrando a visualização da leitura de um cartão.

Após terminar de ler toda a sequência, o jogador pode consultar o *carousel* para confirmar se sua leitura foi correta e, se necessário realizar as devidas correções.

4.3 Simulação

Assim que o jogador finaliza a leitura da sequência de cartões. Ele pode pressionar o botão com a lupa para retornar a exibição do mapa. Com a aplicação neste estado, ele pode pressionar o botão reproduzir para iniciar a simulação.

Uma vez que a simulação é iniciada, a visualização do mapa é bloqueada no estado aberto, e o cortador de grama inicia seu trajeto com base nos comandos lidos, como podemos observar na Figura 10.

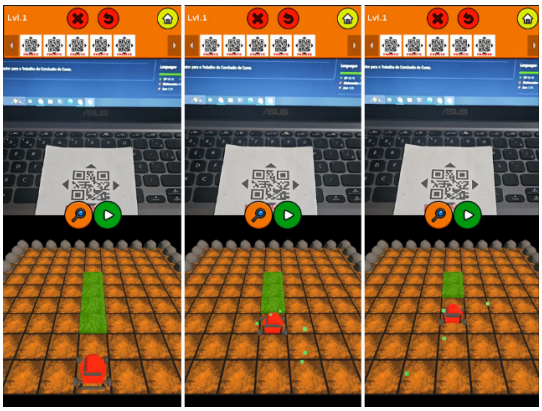

Figura 10: Primeiros passos da execução da solução do nível 1.

A simulação segue até que o último comando seja executado, ou até que o cortador encontre um obstáculo em seu caminho, o que resultará em falha no objetivo. Ao terminar de executar todos os comandos da fila, o simulador verifica se há alguma grama restante, caso que também resultará na falha do objetivo.

Se o jogador obtiver êxito em cortar toda a grama do mapa ao fim da execução dos comandos, uma tela será exibida para notificar seu sucesso. Nesta tela, o jogador pode escolher entre tentar o próximo nível ou encerrar sua sessão, e voltar ao menu inicial. A Figura 11 demonstra uma simulação com o resultado da solução bem sucedido.

Em caso de falha, o jogador é apresentado a uma mensagem de fim de jogo, o notificando que o objetivo não foi alcançado, como pode ser visto na Figura 12. Neste ponto, o jogador pode escolher entre tentar o nível novamente ou encerrar a sessão e voltar ao menu inicial.

4.4 Dados registrados

Ao final de cada jogo, independente se seu resultado foi positivo ou negativo, a aplicação registra o resultado como uma *run*, contendo informações de sua execução como o tempo decorrido na simulação, a quantidade de comandos que foram executados e o resultado final, se foi ou não bem sucedida. Cada *run*, é salva dentro do perfil de cada jogador, que foi cadastrado no início

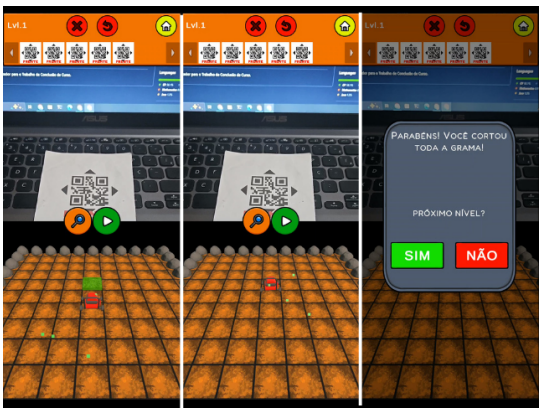

Figura 11: Conclusão da execução da solução do nível 1.

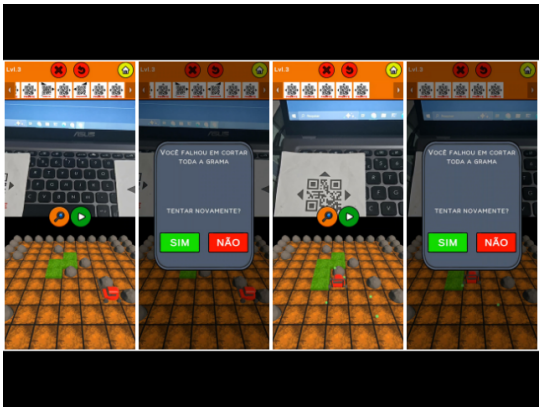

Figura 12: Situações de falha na conclusão do objetivo.

da execução do aplicativo, no menu de cadastro, junto com o identificador numérico, o sexo e a faixa etária do jogador. A Figura 13 exemplifica um desses perfis com usa série de *runs* registradas com ele.


```
{
    "id":1763942379593,
    "age":"6-10",
    "playerSex":"Female",
    "runs":[
        {"elapsedTime":5.139984130859375, "commandsExecuted":5,"success":true},
        {"elapsedTime":4.1051788330078129, "commandsExecuted":4,"success":true},
        {"elapsedTime":11.013458251953125, "commandsExecuted":13,"success":false},
        {"elapsedTime":3.1709932617187S, "commandsExecuted":3,"success":false},
        {"elapsedTime":7.2419738769531125, "commandsExecuted":8,"success":false},
        {"elapsedTime":11.414703369140625, "commandsExecuted":13,"success":true}
    ]
}
```


Figura 13: Exemplo de um arquivo *JSON* contendo os resultados de uma sessão.

4.5 Limitações

A principal limitação encontrada na tela de jogo, ocorre durante a ação de leitura de cartões. A facilidade de leitura dos cartões depende de como eles estão dispostos, além da qualidade da impressão e da iluminação do ambiente externo onde eles estão. Estes são os principais fatores que influenciam no processo de leitura. Para garantir uma boa leitura, deve-se certificar que o ambiente esteja bem iluminado e que os cartões estejam em bom estado de conservação, sem rasgos, amassos ou rasuras. Também é recomendável que os cartões sejam impressos com as dimensões esperadas de 10 cm de comprimento por 10 cm de largura, como recomendado pelas *api's* de leitura.

Figura 14: Dois comandos sendo lidos ao mesmo tempo.

Durante o posicionamento dos cartões para a leitura, é recomendado que se respeite um espaço mínimo entre 5 cm e 10 cm de distância entre eles. Durante a leitura, deixar os cartões muito próximos uns dos outros, pode resultar na leitura acidental de um cartão fora da sua ordem planejada, ou na leitura de múltiplos cartões simultaneamente, o que resultará em falha da leitura da construção da solução. A Figura 14 ilustra uma situação em que dois comandos são indevidamente lidos ao mesmo tempo.

5 Conclusão

O aprendizado de lógica de programação é, sem dúvidas, a base na construção do pensamento analítico e do raciocínio além de ser um dos principais pilares de aprendizagem no decorrer de um curso voltado para a área de Computação. Entretanto, a evasão de acadêmicos da área quando têm seu primeiro contato com a disciplina, tem se mostrado um desafio a ser superado.

Com o intuito de apresentar uma alternativa lúdica e tangível de ensino, foi implementado o *MowerCard*, que propõe estimular de forma lúdica o aprendizado do acadêmico, simulando a lógica de programação com cartões perfurados, utilizando comandos físicos de *QR Code* e tecnologia de Realidade Aumentada. A partir dele, seria possível ensinar, de forma prática e intuitiva, princípios de sequência lógica e controle de fluxo através da movimentação de um robô cortador de grama em um ambiente virtual simulado, estimulando o estudante a superar desafios visuais, sem a necessidade de apresentar conceitos formais e abstratos, como variáveis e funções. A implementação deste projeto, permite a experimentação em sala de aula, com alunos sem experiência prévia com programação e a partir dos resultados coletados conseguir determinar a eficácia desta ferramenta para o ensino.

Apesar da presença de tecnologias de Realidade Aumentada (RA) neste projeto, esta limita-se apenas à visualização da leitura dos cartões físicos, como uma

forma de *feedback* visual. Expandir o uso de RA para fins mais lúdicos, como apresentar a simulação virtual do trajeto do cortador num ambiente físico através das câmeras dos dispositivos, pode ser um acréscimo enriquecedor para a experiência, tornando-a mais marcante e tangível para os envolvidos na experiência.

Referências

2025668. *Lawn Mower*. 2025. Disponível em: <https://sketchfab.com/3d-models/lawn-mower-0833a211f0ab485188e56911c476a0fe>. Acesso em: 23 nov. 2025.

AGUENA, D. S. O uso da programação com cartões perfurados como background para o aprendizado de algoritmos e programação. In: **PROGRAMA VEM PRA UFMS**. [S.l.: s.n.], 2022.

AGUENA, D. S.; MAGALHÃES, R. C. A.; RODRIGUES, E. G. Ensino lúdico de lógica de programação por meio de realidade aumentada e comandos físicos baseados em QR-Code. In: **Anais da 5ª Semana da Enfermagem UFMS-CPCX e do 1 Simpósio Internacional de Conexões Pantaneiras: Tecnologia, Saúde, Direito, Cultura e Linguagens**. Coxim, MS: Even3, 2025. Evento ocorrido em Coxim, MS. Disponível em: ⟨https://www.even3.com.br/anais/conexoes-pantaneiras/⟩.

ANDROID. **Abrir arquivos com o Framework de acesso ao armazenamento**. 2025. Disponível em: ⟨https://developer.android.com/guide/topics/providers/document-provider?hl=pt-br⟩. Acesso em: 23 nov. 2025.

ARUCO. **OpenCV - Detection of ArUco markers**. 2025. Disponível em: ⟨https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html⟩. Acesso em: 14 nov. 2025.

CAMBRUZZI, E.; SOUZA, R. M. de. Robotica educativa na aprendizagem de lóogica de programação: Aplicação e análise. **CBIE-LACLO**, 2015.

COMMONS, C. **Deed — Attribution 4.0 International**. 2025. Disponível em: ⟨http://creativecommons.org/licenses/by/4.0/⟩. Acesso em: 23 nov. 2025.

CRUZ, F. da. 2001. Disponível em: ⟨https://www.columbia.edu/cu/computinghistory/cards.html⟩. Acesso em: 10 nov. 2025.

DACOHER. **Piedra**. 2025. Disponível em: ⟨https://sketchfab.com/3d-models/piedra-a1d7d54d4c5842f48ffd7dbdfea23b80⟩. Acesso em: 23 nov. 2025.

DESIGN, P. **Graphics created by Penzilla Design. Asset pack digital.** 2025. Disponível em: ⟨https://penzilla.itch.io/basic-gui-bundle⟩. Acesso em: 19 nov. 2025.

KIRNER, C.; TORI, R.; SISCOUTTO, R. **Fundamentos de realidade aumentada.** [S.l.]: Pré-Simpósio VIII Symposium on Virtual Reality, 2006.

NETTO, D.; MEDEIROS, L. M.; PONTES, D. de; MORAIS, E. de. Game logic: Um jogo para auxiliar na aprendizagem de lógica de programação. **WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI)**, 2017.

OPENCV. **OpenCV - Open Computer Vision Library**. 2025. Disponível em: ⟨https://opencv.org/⟩. Acesso em: 14 nov. 2025.

RODRIGUES, L. G. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação), **Um simulador tridimensional para o Jantar dos Filósofos**. [UFMS/CPCX]: [s.n.], 2022. Orientador: Deiviston da Silva Aguena.

SILVA, R. R.; RIVERO, L.; SANTOS, R. P. dos. Programse: Um jogo para aprendizagem de conceitos de lógica de programação. **RBIE**, 2021.

SUKEYOSI, W. A. P.; AGUENA, D. S. LeituRA: ferramenta de Apoio Pedagógico com Realidade Aumentada. In: AGUENA, D. d. S. (Ed.). **Aplicações Móveis e Realidade Aumentada Despertando Talentos**. 1. ed. Campo Grande: Life, 2015. p. 17–43.

UNITY. **AR Foundation**. 2025. Disponível em: ⟨https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.3/manual/index.html⟩. Acesso em: 15 nov. 2025.

UNITY. **AR tracked image manager — AR Foundation**. 2025. Disponível em: ⟨https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/tracked-image-manager.html⟩. Acesso em: 17 nov. 2025.

UNITY. **Rule Tile — 2D Tilemap Extras**. 2025. Disponível em: ⟨https://docs.unity3d.com/Packages/com.unity.2d.tilemap.extras@1.6/manual/RuleTile.html⟩. Acesso em: 16 nov. 2025.

UNITY. **Unity - Manual: ScriptableObject**. 2025. Disponível em: ⟨https://docs.unity3d.com/6000.2/Documentation/Manual/class-ScriptableObject.html⟩. Acesso em: 19 nov. 2025.

UNITY. **Unity Engine**. 2025. Disponível em: ⟨https://unity.com/⟩. Acesso em: 14 nov. 2025.

VUFORIA. **VuForia Engine**. 2025. Disponível em: ⟨https://developer.vuforia.com/home⟩. Acesso em: 14 nov. 2025.