Explorando o Minecraft na Simulação de Sistemas Autoadaptativos: Um Estudo Baseado em Missões de Drones

Gabryella Rodrigues, Kalil Rodrigues, Lucas Holanda, Vinicius dos Santos, Luis Gustavo Benevides, Matheus Chagas, Lucas Alves, Paulo Henrique Maia

Centro de Ciências e Tecnologia - Universidade Estadual do Ceará (UECE)
Fortaleza, CE – Brasil

pauloh.maia@uece.br, gesaduece@gmail.com

Abstract. *This paper explores the use of Minecraft as an alternative platform for simulating adaptive autonomous drones. A custom mod was developed to replicate critical mission scenarios in a smart city context. Using the GQM model, we defined metrics to assess key aspects such as adaptive behavior, execution stability, reproducibility, and environmental realism. The results indicate that, despite inherent physical limitations, Minecraft presents a reliable and low-cost environment for prototyping and educational purposes, providing flexibility and consistent behavior across multiple simulation runs.*

Resumo. *Este artigo investiga o uso do Minecraft como uma plataforma alternativa para simulação de drones autônomos adaptativos. A proposta se baseia no desenvolvimento de um mod personalizado capaz de representar cenários críticos em um contexto de smart city. Utilizando o modelo GQM, foram definidas métricas para avaliar critérios como comportamento adaptativo, estabilidade e determinismo de execução, reprodutibilidade e realismo ambiental. Os resultados indicam que o Minecraft, mesmo com limitações no realismo físico, apresenta bom desempenho como ambiente de prototipação e ensino, oferecendo flexibilidade de uso e estabilidade em múltiplas simulações.*

1. Introdução

O uso de simuladores é essencial no desenvolvimento e avaliação de sistemas autônomos, principalmente na área da robótica. Sua relevância é dada pela necessidade de testar sistemas minimizando custos ao explorar situações de tempo de execução imitando a realidade [Choi et al. 2021]. Em alguns contextos modernos, como o uso de drones autônomos em cidades inteligentes, isso significa que é possível diminuir custos em testes de voo, além de preservá-los de possíveis danos nos equipamentos desnecessariamente, permitindo a realização de experimentos seguros e controlados. Dentre outras ferramentas, simuladores foram desenvolvidos para atender a esse propósito, podendo incluir desde soluções de fácil acesso e flexíveis, até ambientes mais realistas, com física detalhada e complexa.

Contudo, embora existam alternativas para simulação, tais ferramentas podem impor barreiras de entrada para iniciantes por conta de suas inerentes curvas de aprendizado para utilização. Entre as principais limitações, destacam-se a necessidade de hardware robusto, configurações de ambientes complexos, e as dificuldades acidentais como documentação imprecisa, inadequada ou ausente [Afzal et al. 2021].

Diante de tais limitações, pesquisadores e educadores vêm explorando o potencial no desenvolvimento de jogos digitais, e principalmente, de engines de jogos como ambientes alternativos para simulações. Motores como Unreal Engine[1] e Unity[2] demonstram potencial para simulações imersivas e experimentos no desenvolvimento de gêmeos digitais [Singh et al. 2025, Sørensen et al. 2022].

Nesse contexto, o jogo Minecraft[3] se mostra uma plataforma promissora devido à sua natureza representada por um ambiente aberto, customizável e sustentado por uma comunidade ativa e colaborativa. Este jogo permite, de maneira acessível, a criação e modelagem de ambientes, por meio de *mods*, que são modificações realizadas em softwares, especialmente jogos digitais, com o intuito de alterar, ampliar ou personalizar suas funcionalidades ou conteúdos. Essas modificações de regras internas e integração de comportamentos personalizados abrem espaço para vários tipos de aplicações, incluindo simulações de sistemas autônomos.

Este trabalho visa explorar e avaliar o uso do Minecraft como um ambiente para simulação de comportamentos adaptativos de drones autônomos em cenários críticos, a partir do desenvolvimento de um *mod* customizado. Como referência para os cenários e os comportamentos avaliados, tomou-se como base o simulador Dragonfly [Maia et al. 2019b, Maia et al. 2019a], que simula drones com comportamentos adaptativos em situações excepcionais, por meio de *wrappers* implementados utilizando a técnica de programação orientada a aspectos (AOP) e a linguagem DRESS-ML [Alves et al. 2022]. Para guiar a avaliação, utilizamos o modelo GQM (*Goal-Question-Metric)*, para construir a metodologia de coleta de métricas adequadas para o objetivo do estudo.

Os resultados obtidos demonstraram que o Minecraft, com o *mod* desenvolvido, teve capacidade de simular com sucesso missões de drones adaptativos em diferentes cenários críticos, apresentando comportamento consistente ao longo das execuções repetidas. A análise quantitativa baseada em métricas como o desvio padrão das posições finais e níveis de bateria, indicou estabilidade no sistema e baixa variabilidade entre os testes. Apesar de algumas limitações técnicas observadas, a plataforma se mostrou promissora como alternativa de baixa complexidade para prototipação, ensino e experimentação de lógica adaptativa.

As principais contribuições apresentadas nesse estudo são: (1) A implementação de uma simulação de translado realizado por drones no Minecraft, explorando possíveis interrupções num contexto de entrega de pacotes, e integrando mecanismos de tomada de decisão que respondem a mudanças no ambiente; (2) A proposta de uma abordagem metodológica para observar e analisar comportamentos adaptativos na simulação; E (3) uma demonstração que o Minecraft pode servir como uma plataforma de simulação de sistemas autoadaptativos flexível e de baixa complexidade, destacando vantagens e limitações.

2. Contexto e Trabalhos Relacionados

Drones são usados em diversas aplicações, desde vigilância e logística até monitoramento ambiental e resposta a desastres [Singh and Kumar 2025]. Em cenários críticos, drones

[1]https://www.unrealengine.com/pt-BR

[2]https://unity.com/pt

[3]https://www.minecraft.net/pt-br/about-minecraft

precisam se adaptar de forma autônoma, considerando fatores como bateria, obstáculos e prioridades de missão. Nesse contexto, o uso de simuladores mostra-se essencial por viabilizar testes em um ambiente controlado, seguro e econômico, antes do uso de drones em aplicação real. Isso otimiza o processo de desenvolvimento e contribui para a eficiência e segurança do sistema final.

Como exemplo desses simuladores, pode-se citar o Dragonfly [Maia et al. 2019b], desenvolvido para simular missões com drones autônomos em um cenário de cidades inteligentes. Nesse estudo é explorada a "adaptação cautelosa", proposta para lidar com componentes resistentes a mudanças, sistemas que, por segurança ou criticidade, não podem ser adaptados de forma direta ou desordenada [Maia et al. 2019a]. Esse conceito foi posteriormente explorado no domínio de Internet das Coisas (IoT) [Fonseca et al. 2021] e instanciado em uma linguagem específica de domínio (DSL), proposta para facilitar a modelagem de cenários excepcionais e comportamentos adaptativos [Alves et al. 2022].

2.1. Simuladores Tradicionais para Drones

Ferramentas como Gazebo[4], AirSim[5] ou CoppeliaSim[6] são empregadas para a simulação de sistemas autônomos. Tais ferramentas buscam oferecer ambientes com física realista e suporte à integração com frameworks como o ROS[7]. Alguns simuladores aproveitam motores de jogos para proporcionar simulações visuais e físicas mais imersivas. O Microsoft AirSim, por exemplo, utiliza a Unreal Engine, enquanto ferramentas como Flightmare e FlightGoggles[8] utilizam a Unity para renderização das simulações [Dimmig et al. 2024]. Apesar disso, parte desses simuladores foca na avaliação de funcionalidades com física realista e algoritmos de controle, o que pode aumentar a curva de aprendizado e limitar seu uso em ambientes educacionais ou em experimentação inicial [**?**].

2.2. Uso de Jogos como Alternativa de Simulação

Jogos digitais como o Minecraft podem ser usados tanto na educação quanto na pesquisa científica, sendo valorizados por sua capacidade de apoiar atividades experimentais e promover a aprendizagem ativa [Nebel et al. 2016]. Trabalhos recentes têm mostrado como o Minecraft pode ser usado para a geração procedural de ambientes tridimensionais e para o treinamento de agentes em múltiplas tarefas em diferentes domínios como *smart homes* [Wang et al. 2024] e de planejamento urbano [Christiansen and Scirea 2022]. Isso sugere que jogos podem servir como plataforma para experimentação de comportamentos adaptativos e colaborativos. Entretanto, apesar desses avanços, ainda são escassos os estudos que avaliem sistematicamente o Minecraft como uma plataforma de simulação de comportamentos adaptativos de sistemas autônomos em missões estruturadas. Ao propor uma avaliação baseada em métricas da capacidade do Minecraft de prover suporte a missões autônomas adaptativas, este trabalho contribui para preencher essa lacuna e ampliar as perspectivas sobre o uso de jogos como ferramentas de simulação para sistemas autônomos.

[4]https://gazebosim.org/home
[5]https://github.com/microsoft/AirSim
[6]https://www.coppeliarobotics.com/
[7]https://www.ros.org/
[8]https://github.com/uzh-rpg/flightmare e https://flightgoggles.mit.edu/

3. Metodologia

Este estudo adotou o modelo GQM *(Goal-Question-Metric)*, uma abordagem utilizada para estruturar objetivos de análise, formular perguntas de pesquisa e estabelecer métricas relevantes. A partir da aplicação dessa abordagem, perguntas foram categorizadas em seis critérios: **(C1) Configuração do Ambiente e Cenários:** recursos de configuração de ambiente virtual e criação dos cenários; **(C2) Monitoramento e Telemetria:** recursos de acompanhamento de informações em tempo real sobre o drone e o ambiente; **(C3) Lógica Autônoma e Comportamento do Drone:** recursos para desenvolvimento e simulações de comportamento e tomadas de decisão; **(C4) Realismo e Fatores Ambientais:** nível de realismo no ambiente simulado, considerando fatores como vento, clima e obstáculos; **(C5) Registro, Análise e Reprodutibilidade:** capacidade de registrar e analisar os dados dos testes e de repetir experimentos com as mesmas configurações; e **(C6) Requisitos Técnicos e Compatibilidade:** condições mínimas para executar o simulador e informações sobre integração com ferramentas externas.

A Tabela 1 apresenta a lista de perguntas definidas, acompanhadas da indicação de qual critério pertencem e suas respectivas métricas para respondê-las. Para cada pergunta, as métricas definidas quantificam ou qualificam as funcionalidades e limitações do ambiente avaliado.

Tabela 1. Critérios numerados com suas respectivas perguntas e métricas

[Critério] Pergunta	Métricas
[C1] O ambiente permite configurar os cenários propostos?	Quantidade de Cenários implementados
[C1] É possível configurar cenários com parâmetros como bateria, obstáculos e clima?	Quantidade de Parâmetros configuráveis
[C1] Quantas ferramentas e etapas são necessárias para configurar um cenário completo?	Nº médio de ferramentas/etapas necessárias
[C2] É possível visualizar dados de posição, velocidade, bateria e sensores em tempo real?	Lista de dados disponíveis em tempo real
[C2] O ambiente oferece visualizações gráficas em tempo real?	Existência de visualização gráfica
[C2] Existe suporte para exportação de dados de telemetria?	Formatos de exportação disponíveis
[C3] É possível programar lógica baseada em sensores, regras e algoritmos?	Tipos de lógica suportados
[C3] O ambiente possui ferramentas internas ou precisa de integração externa?	Necessidade de ferramentas externas
[C3] As ações do drone são previsíveis e consistentes em execuções repetidas?	Taxa de desvio padrão entre execuções
[C4] O ambiente simula efeitos como vento, turbulência, atrito e colisões?	Lista de efeitos físicos simulados
[C4] É possível alterar condições ambientais em tempo real?	Capacidade de alterar condições dinâmicas
[C4] Existe suporte para cenários imprevisíveis automáticos?	Existência de geração automática de cenários
[C5] O ambiente gera *logs*, gravação de vídeo e exportação de métricas?	Tipos de registros gerados
[C5] É possível configurar e repetir testes controlados?	Suporte a testes reprodutíveis
[C5] Existe suporte para execução automatizada de múltiplos testes?	Suporte a automação de testes em lote
[C6] O ambiente exige hardware específico ou de alto desempenho?	Requisitos mínimos de CPU, RAM e GPU
[C6] É compatível com Linux, Windows e macOS?	Lista de sistemas suportados
[C6] Existem limitações ou recursos ausentes em algum sistema?	Lista de limitações por sistema operacional
[C6] Quais são as dependências externas?	Lista de dependências externas
[C6] O ambiente oferece instalação via Docker ou instaladores?	Disponibilidade de Docker ou instaladores

3.1. Cenários de Avaliação

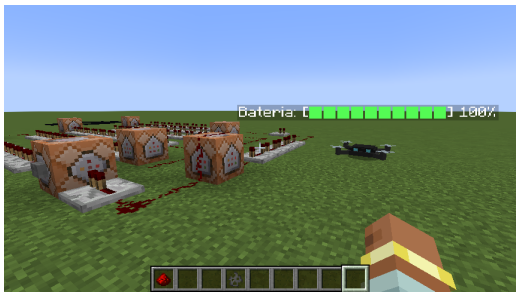
Para a realização do estudo, foram definidas quatro situações distintas, utilizando um subconjunto dos cenários avaliados com o Dragonfly [Maia et al. 2019a]. Esses cenários de exemplo cobrem diferentes níveis de complexidade e têm como objetivo avaliar tanto comportamentos provenientes do padrão de fábrica quanto a capacidade de adaptação em certas condições excepcionais. Os cenários e suas respectivas finalidades são:

E1. Missão Básica: Navegação simples entre dois pontos fixos, sem interferência de variáveis externas ou eventos inesperados, visando avaliar a execução padrão da lógica de navegação e o comportamento base do drone.
E2. Pouso Programado por Baixa Bateria: Realização de pouso de segurança controlado ao detectar que a bateria atingiu um nível crítico mínimo, para verificar capacidade do sistema em reconhecer uma condição de risco e executar uma ação predefinida.
E3. Pouso de Emergência com Checagem de Terreno: Antes de realizar a operação de pouso, caso esteja sobrevoando sobre a água, o drone realiza uma ação adicional de deslocamento para uma região segura, como a margem do rio ou lago, tendo como objetivo avaliar a integração entre sensores e lógica condicional adaptativa.
E4. Exceção por Fatores Ambientais: Em condição crítica de bateria, caso outras variáveis externas favoreçam a continuidade da missão, o drone continua sua rota e finaliza a missão, para investigar a capacidade do sistema de reavaliar prioridades e alterar seu comportamento com base em múltiplos fatores em tempo de execução.

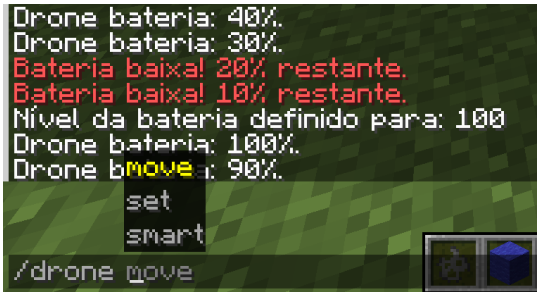
3.2. Ambiente de Desenvolvimento

A estruturação do ambiente de desenvolvimento foi feita com foco na criação de um *mod* para o Minecraft Java Edition que fosse capaz de simular os cenários propostos. Como IDE, foi utilizado o IntelliJ IDEA Community Edition[9], configurado para a linguagem Java (versão 21). Por ser suportada por uma comunidade ativa e compatível com as ferramentas utilizadas, a versão 1.21.1 do Minecraft foi escolhida.

A estrutura do projeto de desenvolvimento seguiu o padrão tradicional usado para a criação de *mods* no ecossistema do Minecraft, utilizando o sistema de automação Gradle disponibilizado pelo ambiente de desenvolvimento do *modloader*, programa que facilita a instalação e o uso de modificações, *mods*. Para execução e carregamento do *mod*, o *modloader* adotado foi o NeoForge[10], versão 21.1.77. O NeoForge foi selecionado para a realização desse estudo por sua estabilidade, suporte ativo e compatibilidade com as ferramentas utilizadas no projeto.



(a) Entidade do drone dentro do jogo no cenário de testes.



(b) Lista de comando dentro do Minecraft para realizar as ações do drone.

Figura 1. Imagens da simulação no Minecraft.

[9]https://www.jetbrains.com/pt-br/idea/
[10]https://github.com/neoforged/NeoForge

3.3. Implementação do *Mod*

Utilizou-se o NeoForge e o Blockbench[11] para criar o modelo gráfico do drone. No código-fonte, foram adicionadas nove classes para importar o modelo, animações e implementar a lógica do drone e do sistema de vento. As classes `DroneModel`, `DroneAnimations`, `DroneRenderer`, são responsáveis por criar a renderização e aparência do drone dentro do ambiente. A classe `DroneEntity` implementa a classe nativa do jogo `Mob`, garantindo comportamentos básicos como entidade. Foram desenvolvidas classes específicas para o sistema de bateria, `DroneBattery`, geração de *logs*, `DroneLog`, e comandos, `DroneCommand`. Por fim, foi criada a classe `DroneAutomataDecorator`, que funciona como um *wrapper* para o drone, responsável por adicionar os comportamentos autônomos de forma separada. Utilizando o padrão Decorator, permitindo incluir novas funcionalidades sem precisar mexer diretamente no código principal do drone. Isso torna o sistema mais organizado e flexível, já que o modo autônomo pode ser ativado ou desativado sem complicar a estrutura da entidade.

O sistema de vento foi modelado com duas classes: `WindSystem`, que define os parâmetros como direção e intensidade, e `WindCommand`, que permite controlar a configuração por comandos. O vento afeta diretamente o consumo de bateria, assim como outros fatores ambientais, como a chuva e tempestades.

A interface de comandos disponibilizada no jogo foi usada para criação de comandos em que o usuário pode indicar coordenadas para o drone, permitindo controlar seu deslocamento dentro do ambiente simulado. O comando de configurar o comportamento do drone é definido entre básico e autômato, e comandos que permitem definir o nível de bateria total ou o nível atual, facilitando a criação de cenários de teste ou simulações específicas. Utilizou-se de uma estrutura com blocos de comando e circuítos lógicos nativos do ambiente para a realização das simulações dos cenários de forma automatizada. Além disso, o sistema envia mensagens no *chat* do jogo, com atualizações da bateria.

3.4. Procedimento Experimental

Para avaliar o comportamento do drone em execuções repetidas, foi conduzido um experimento controlado. Nesse experimento, o *mod* implementado foi utilizado para montar um cenário de execução dentro do jogo, no qual o drone foi programado para percorrer um trajeto entre dois pontos fixos. Todo o cenário foi construído dentro do Minecraft, garantindo que cada execução ocorresse de maneira idêntica, sem intervenção humana. O ambiente está disponível para acesso em um repositório público para replicação dos cenários[12]. Foram realizadas e registradas 50 execuções por cenário de exemplo, o que permite a observação de possíveis variações causadas pela dinamicidade inerente ao mundo simulado ou por imprecisões ou limitações do mecanismo de controle implementado. A partir desses dados, foi calculado o desvio padrão para medir, de forma objetiva, a estabilidade e a previsibilidade do comportamento ao longo das múltiplas iterações.

4. Resultados

Os resultados foram obtidos a partir da execução dos cenários de exemplo e da coleta das métricas definidas. A capacidade e experiência do uso do Minecraft como plataforma para

[11] https://www.blockbench.net/

[12] https://github.com/gabryellaro/DroneMod

simulação de drones autônomos adaptativos foi analisada de forma qualitativa e quantitativa, considerando as categorias estabelecidas. A seguir, esses resultados são explorados por categoria.

4.1. Configuração do Ambiente e Cenários (C1)

Foi possível configurar os quatro cenários propostos, incluindo a parametrização de movimentação, níveis de bateria, vento e condições climáticas, como tempo limpo, chuvoso e com tempestades. O processo seguido para configurar o projeto e executar a simulação de um cenário completo consiste em quatro etapas: (I) Instalação das dependências (Java 21, IntelliJ IDEA, Gradle, Git); (II) Clonagem de repositório; (III) Realização do *build* do projeto utilizando os *scripts* de Gradle disponibilizados no projeto; e (IV) Execução do Cliente com o IntelliJ.

4.2. Monitoramento e Telemetria (C2)

Durante as simulações, foi possível monitorar por *logs*, em tempo real, as informações relevantes como posição e nível de bateria do drone. Além disso, o Minecraft oferece visualização gráfica que permite acompanhar as missões em tempo de execução, proporcionando uma perspectiva adicional para acompanhar o comportamento do drone em cada cenário. A funcionalidade de exportação de dados de telemetria implementada disponibiliza os registros internos referentes aos rastros da execução em arquivo de texto (`.txt`).

4.3. Lógica Autônoma e Comportamento do Drone (C3)

Foi possível implementar, utilizando lógicas baseadas em sensores, regras condicionais e algoritmos personalizados, o que atendeu de forma satisfatória aos requisitos definidos para os cenários de exemplo. As ferramentas internas oferecidas tanto pelo NeoForge quanto pelo Minecraft foram suficientes para essa implementação. No entanto, integrações externas podem ser utilizadas para ampliar as capacidades e facilitar o desenvolvimento, como o *mod* SmartBrainLib[13] que retrabalha a lógica de entidades do Minecraft.

Após as 50 execuções de teste em cada cenário, analisamos o comportamento do drone com base no cálculo de desvio padrão das posições (coordenadas X, Y e Z) e do nível de bateria ao término de cada percurso. Os resultados apontaram para um comportamento estável, mas, mesmo que baixas, variações ainda foram identificadas. Destaca-se a execução de dois cenários: E1, por possuir a maior taxa de variação em um único atributo, e o E3, que apresentou maior quantidade de atributos com variância.

O cenário E1 apresentou resultados idênticos nas posições ao final do pouso, mas teve a maior variação registrada em nível de bateria (aproximadamente 1%). Isso pode estar relacionado ao tempo de resposta variável, devido ao funcionamento interno do Minecraft. O cenário E3 apresentou variações nos eixos X e Z, que são as coordenadas do jogo referentes à posição no plano, como latitude e longitude. Fora isso, ocorreu um desvio na bateria de aproximadamente 0,5%, indicando que situações de maior complexidade lógica ou ambiental podem gerar mais inconsistências no final da trajetória.

Apesar disso, todas as variações observadas foram consideradas insignificativas por não afetarem o sucesso das missões e o resultado final da simulação. Essas oscilações

[13] https://github.com/Tslat/SmartBrainLib

podem decorrer de fatores internos do próprio jogo, como o gerenciamento de *ticks* ou limitações do motor de física implementado. *Ticks* são as funções que definem o que deve acontecer a cada quadro exibido no jogo, o que faz com que seja considerado como o tempo interno do jogo. Por fim, não foi possível determinar precisamente com os dados coletados se as causas são estruturais, relacionadas ao ambiente de simulação, ou lógicas, relacionadas à programação dos controladores.

4.4. Realismo e Fatores Ambientais (C4)

O sistema oferece suporte para alterar dinamicamente variáveis ambientais, como condições climáticas e presença de obstáculos, o que contribui para simulações mais complexas. Efeitos adicionados como vento também puderam ser simulados, ainda que com realismo físico limitado quando comparado a simuladores tradicionais. Observou-se que o realismo visual do Minecraft, por conta de sua estética simplificada, não teve impacto significativo na compreensão. Adicionalmente, o ambiente permite a geração automática de cenários imprevisíveis, criando condições variadas sem a necessidade de reconfiguração manual constante. Além disso, é possível utilizar comandos para alterar arbitrariamente as condições do ambiente em tempo de execução.

4.5. Registro, Análise e Reprodutibilidade (C5)

Durante as simulações, foi possível gerar *logs* de posição e bateria do drone, além de registrar eventos da execução. Foi observada a possibilidade de configurar e repetir testes controlados com condições idênticas, o que garante reprodutibilidade para experimentos comparáveis. Pela experiência de desenvolvimento e execução do experimento, foi possível inferir que é viável também a execução automatizada de múltiplos testes em lote.

4.6. Requisitos Técnicos e Compatibilidade (C6)

O ambiente apresentou compatibilidade com Windows, Linux e macOS. Limitações foram identificadas em ambientes macOS, onde pode ser necessário utilizar recursos auxiliares para garantir pleno funcionamento. Além disso, o jogo opera de forma estável em máquinas que cumpram os requisitos mínimos: processador Intel Core I3, 4GB de RAM e placa gráfica integrada. Demonstrando uma boa compatibilidade e portabilidade em diferentes tipos de máquinas e configurações de hardware. Por outro lado, o processo de instalação ainda demanda grau de complexidade, exigindo a configuração manual de Java, IntelliJ IDEA, Git, Gradle e o modloader NeoForge. O Minecraft também fornece suporte a contêineres Docker[14], visando o desenvolvimento de servidores em jogo.

5. Discussão

A partir da aplicação dos cenários propostos, foi demonstrado que o jogo Minecraft, apesar de não ser originalmente desenvolvido como uma plataforma de simulação, oferece uma base viável para simulações envolvendo drones adaptativos. A estrutura aberta e versátil da plataforma, junto ao suporte ativo da grande comunidade de *mods*, colabora na criação de ambientes simulados personalizáveis e acessíveis do ponto de vista computacional. Isso foi suficiente para a replicação dos comportamentos definidos, incluindo as lógicas de adaptação, a tomada de decisão em cenários excepcionais e a reavaliação

[14]https://github.com/TheSudoYT/Running-Minecraft-In-Production

dinâmica de prioridades em tempo de execução. A possibilidade de integração com outros mods já existentes pode ampliar funcionalidades e cenários possíveis. Entretanto, essa integração demanda adaptações na implementação, uma vez que cada mod pode apresentar estruturas, APIs e requisitos próprios, exigindo ajustes específicos para compatibilidade e funcionamento consistente. Outras vantagens podem ser mencionadas, como a possibilidade de visualização 3D em tempo de execução e requisitos mínimos acessíveis. A estabilidade observada nessas configurações sugere que o Minecraft tem potencial para ser usado em ambientes com infraestrutura limitada, além de possibilitar seu uso em contextos educacionais.

5.1. Limitações e Ameaças à Validade

Nas limitações desse estudo, observa-se possíveis riscos relacionados a imprecisão e realismo físico. Efeitos como atrito, turbulência e resposta a colisões podem ser inacurados ou insuficientes. Isso pode limitar o uso da plataforma em testes que dependam de simulações para sistemas críticos detalhadas de ambientes do mundo real. Além disso, a preparação do simulador criado pode ser complicada para usuários inexperientes, devido a configuração de dependências e uso de ferramentas de *build* e *modloaders* específicos.

Quanto a validade do estudo, considera-se as ameaças (de): **(i) Internas:** A variabilidade entre execuções pode comprometer a consistência de resultados, portanto, para mitigar esse risco, o experimento foi realizado com 50 execuções repetidas por cenário, sob condições controladas e idênticas; **(ii) Externas:** a generalização dos resultados pode ser limitada, uma vez que o *mod* criado foi desenvolvido especificamente para este estudo, então foram representados diferentes níveis de complexidade e adaptação nos exemplos selecionados; **(iii) Construção:** O uso do Minecraft pode não representar situações fisicamente realísticas e, por isso, o estudo restringiu o foco na análise da lógica de decisão e comportamentos adaptativos da simulação; **(iv) Conclusão:** Devido à natureza subjetiva do estudo, parte das avaliações realizadas foi qualitativa e, para reduzir esse risco, utilizou-se o modelo GQM para guiar a formulação de perguntas e métricas objetivas.

6. Conclusão

Este estudo investigou o uso do Minecraft como alternativa para a simulação de drones adaptativos, com foco na adaptação em tempo de execução para cenários excepcionais com reavaliação dinâmica de prioridades. Os resultados revelaram que o Minecraft oferece a possibilidade de realizar simulações consistentes e versáteis, com experiência intuitiva e facilidade de uso, inclusive para dispositivos com infraestrutura limitada. Apesar das limitações identificadas, a execução controlada de múltiplos testes contribuiu para validar a estabilidade dos comportamentos simulados.

Com isso, é reforçada a viabilidade do uso de jogos como plataformas de simulação, como alternativa aos simuladores tradicionais, sobretudo em estágios iniciais de desenvolvimento ou contextos educacionais. **Trabalhos Futuros:** Dentre as lacunas a serem preenchidas, existe a extensão dos cenários de avaliação cobrindo mais casos de possíveis falhas, bem como o desenvolvimento de funcionalidades complementares, como automatização de testes em lote. Também pretende-se replicar o experimento com outros simuladores do estado da arte, e observar o comparativo dos resultados a fim de entender melhor o diferencial do uso de jogos como ferramentas de simulação para drones autônomos.

Referências

Afzal, A., Katz, D. S., Le Goues, C., and Timperley, C. S. (2021). Simulation for robotics test automation: Developer perspectives. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 263–274.

Alves, L., Pereira, J. D., Aragão, N., Chagas, M., and Maia, P. H. (2022). Dress-ml: a domain-specific language for modelling exceptional scenarios and self-adaptive behaviours for drone-based applications. In *ICSE 2022*, pages 56–66.

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., et al. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1):e1907856118.

Christiansen, S. S. and Scirea, M. (2022). Space segmentation and multiple autonomous agents: a minecraft settlement generator. In *2022 IEEE Conference on Games (CoG)*, pages 135–142. IEEE.

Dimmig, C. A., Silano, G., McGuire, K., Gabellieri, C., Hšnig, W., Moore, J., and Kobilarov, M. (2024). Survey of simulators for aerial robots: An overview and in-depth systematic comparisons. *IEEE Robotics & Automation Magazine*.

Fonseca, A., Sousa, D., Chagas, M., Maia, P. H. M., Alves, L., Praxedes, V., and Junior, E. (2021). Dealing with iot defiant components. In *SESoS/WDES 2021*, pages 24–31. IEEE.

Maia, P. H., Vieira, L., Chagas, M., Yu, Y., Zisman, A., and Nuseibeh, B. (2019a). Cautious adaptation of defiant components. In *ASE 2019*, pages 974–985. IEEE.

Maia, P. H., Vieira, L., Chagas, M., Yu, Y., Zisman, A., and Nuseibeh, B. (2019b). Dragonfly: a tool for simulating self-adaptive drone behaviours. In *SEAMS 2019*, pages 107–113. IEEE.

Nebel, S., Schneider, S., and Rey, G. D. (2016). Mining learning and crafting scientific experiments: a literature review on the use of minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355–366.

Singh, M., Kapukotuwa, J., Gouveia, E. L. S., Fuenmayor, E., Qiao, Y., Murray, N., and Devine, D. (2025). Comparative study of digital twin developed in unity and gazebo. *Electronics*, 14(2):276.

Singh, R. and Kumar, S. (2025). A comprehensive insights into drones: History, classification, architecture, navigation, applications, challenges, and future trends.

Sørensen, J. V., Ma, Z., and Jørgensen, B. N. (2022). Potentials of game engines for wind power digital twin development: an investigation of the unreal engine. *Energy Informatics*, 5(Suppl 4):39.

Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., et al. (2024). Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.