

INTEGRAÇÃO DIRETA DE MODELOS BIM EM MOTOR DE JOGOS OPEN SOURCE: UM LEITOR IFC PARA GODOT

Bruno Muniz⁽¹⁾, José Granja⁽²⁾, Miguel Azenha⁽³⁾

(1) Universidade do Minho, ISISE, ARISE, Departamento de Engenharia Civil, Guimarães, Portugal, ORCID:0000-0002-3183-9779

(2) Universidade do Minho, ISISE, ARISE, Departamento de Engenharia Civil, Guimarães, Portugal, ORCID: 0000-0002-0858-4990

(3) Universidade do Minho, ISISE, ARISE, Departamento de Engenharia Civil, Guimarães, Portugal, ORCID:0000-0003-1374-9427

Resumo

A indústria dos videojogos tem desenvolvido, ao longo de décadas, experiências 3D que se destacam pela sua elevada fidelidade visual e interatividade. Na base da criação dos videojogos estão os motores de jogos, que permitem a gestão de diversos recursos a nível de interação e simulação. A indústria da construção tem incorporado motores de jogos nos últimos anos mediante integrações com modelos BIM, especialmente para a exploração de usos como a visualização de projetos em realidade virtual e aumentada, identificação de situações de risco e simulações. Contudo, os fluxos de trabalho atuais baseiam-se maioritariamente na utilização de motores e *softwares* proprietários e/ou de fluxos fragmentados de conversão de dados, que envolvem exportação das informações gráficas e metadados separadamente para outros formatos padronizados, como *Wavefront* e JSON, para depois permitir a sua inserção nestes motores. Este trabalho apresenta o desenvolvimento de um leitor de modelos BIM, baseado no formato aberto IFC, para o motor de jogos de código aberto Godot. O leitor permite o processamento direto das informações geométricas e alfanuméricas do IFC de maneira integrada para o motor, permitindo a sua utilização para a criação de ferramentas openBIM. Finalmente, este trabalho demonstra alguns exemplos de criação de ferramentas BIM a partir do leitor desenvolvido, para validar o potencial de utilização da ferramenta.

1. Introdução

A visualização e interação com os modelos de informação de edifícios é uma das atividades centrais na metodologia BIM [1]. Paralelamente, a indústria dos jogos digitais também tem como um dos fatores centrais a visualização e interação com modelos que carregam informações 3D e metadados, apesar de possuir propósitos significativamente distintos dos da construção.

A indústria dos videojogos tem crescido ano após ano, com receita global próxima de 260 mil milhões de dólares no ano de 2025 [3]. Desde os seus passos iniciais, em meados do século XX, os videojogos tiveram uma grande evolução na sua fidelidade gráfica e capacidade de proporcionar simulações do mundo real através do fotorrealismo e representação de fenómenos naturais [4]. Os motores de jogos são os *softwares* centrais nesta evolução, pois agregam diversas capacidades, tanto a nível gráfico, como de coordenação de informação geométrica e alfanumérica.

Para exemplificar a coordenação destas informações, videojogos de automobilismo precisam de agregar a informação geométrica dos carros a atributos como aceleração, capacidade de receber danos, capacidade de fazer curvas, entre outros. Assim, os modelos utilizados para representar ativos nos jogos possuem paralelo com os modelos de informação utilizados no setor da construção. Videojogos como SimCity ou Cities: Skylines tornam este paralelo ainda mais evidente, ao fazer a representação de ativos num ambiente urbano e utilizar metadados para determinar a densidade populacional, tráfego [5], entre outros parâmetros.

Partindo deste princípio, os motores de jogos já foram previamente utilizados em iniciativas na indústria e ao nível académico, de maneira a recorrer às capacidades gráficas, de simulação e de interação com modelos e gerar experiências fiéis e imersivas. Estas iniciativas focam-se maioritariamente em aspetos de visualização do modelo e interação mediante recurso à realidade virtual (VR) e à realidade aumentada (AR). Por outro lado, já existem estudos que preveem uso de motores de jogos para a simulação e a criação de aplicações com usos variados, a exemplo da gestão de construção [6], havendo ainda fatores que impedem uma maior integração de modelos e dos motores.

Dentre os fatores impeditivos, a interoperabilidade ainda é central. Isto se deve maioritariamente à necessidade de fragmentação das informações contidas no modelo (geometria, metadados e relações) em formatos específicos para poderem ser utilizadas nos motores [7]. Algumas soluções, como o Datasmith [8] na Unreal Engine e IFC Editor Plugin para o Unity [9], permitem integrar diretamente estas informações nos respetivos motores a partir do processamento de modelos IFC, entretanto ainda possuem limitações na obtenção de algumas informações, a exemplo dos espaços no Datasmith e da flexibilidade para a criação de colisões no Unity.

Outro aspeto importante na utilização de ambos motores (Unity e Unreal), está no fato de serem proprietários, o que pode sujeitar utilizadores a mudanças de licença e alteração nos modelos de pagamento, com potencial de inviabilizar o seu uso. Isto pode ser exemplificado pela mudança do modelo de pagamentos da Unity em 2023, que trocou o plano anual de pagamentos por um pagamento por cada cópia do jogo criado com o motor, diminuindo o potencial de lucro por replicabilidade [10]. Em resposta a este último caso, muitos desenvolvedores partiram em busca de soluções confiáveis de código aberto, o que acabou por impulsionar motores de jogos como o Godot.

Godot é um motor de jogos 2D e 3D, de uso geral, com capacidade de criar videojogos e aplicações para *desktop*, *mobile* e *web* [11]. O seu desenvolvimento começou em 2001 em um estúdio argentino, mas o projeto tornou-se de código aberto em 2014. No relatório “The Big Game Engine Report of 2025” o Godot figura como um dos três principais motores de jogos, com a Unreal engine e o Unity [12]. Entretanto, diferentemente destes outros motores, o Godot possui código aberto. Além disso, o motor conta com atualizações e melhorias constantes, estando; à data de escrita deste artigo, na sua versão 4.6.

O desenvolvimento no motor Godot é feito maioritariamente com a linguagem nativa do motor, GDScript. Entretanto, os desenvolvedores publicam oficialmente uma versão .NET, com capacidade de codificar em C# além de dar suporte à codificação em C++ através do recurso GDExtensions [13]. Complementarmente, *plugins* criados pela comunidade têm permitido a integração de outras linguagens como Python e Rust.

Este trabalho então apresenta um leitor de IFC (GDIFC) para o motor de jogos de código aberto Godot, que permite a leitura de ficheiros IFC de forma direta e com informação geométrica e alfanumérica integradas, de tal modo que isto permite a criação de aplicações BIM para diversos fins. Este leitor visa proporcionar uma alternativa ao uso dos motores proprietários ao mesmo tempo que visa colmatar algumas das lacunas que existem nas soluções proprietárias. Finalmente, o trabalho apresenta exemplos de aplicação, demonstrando a utilidade e incorporação do leitor para a criação de aplicações e de experiências interativas a partir de modelos BIM.

As próximas secções do trabalho apresentam respetivamente: desenvolvimento do GDIFC, com descrição pormenorizada das funcionalidades pretendidas e de como o modelo IFC é processado; uso e configurações, onde são apresentadas as formas de utilizar o GDIFC e quais as configurações possíveis; exemplos de aplicação, com a apresentação de protótipos que preveem a utilização do GDIFC para usos BIM; conclusões e desenvolvimentos futuros.

2. Desenvolvimento do GDIFC

O desenvolvimento do GDIFC partiu da busca pela exploração do potencial do motor de jogos Godot para o desenvolvimento de ferramentas BIM baseadas em IFC. A capacidade de atender a diferentes casos de uso, implicou no desenvolvimento de uma solução abrangente, sendo um dos fatores centrais que permeou o seu desenvolvimento. Esta abrangência implica na capacidade de satisfazer requisitos mais usuais, como permitir a criação de experiências em realidade virtual (VR), assim como permitir a criação de aplicações que historicamente não são desenvolvidas em motores de jogos, como visualizadores BIM. Desta maneira, as próximas subsecções apresentam aspetos do desenvolvimento que visaram garantir as capacidades básicas de leitura do IFC para obtenção de uma solução abrangente, enquanto se tentou dar uma resposta às lacunas deixadas por outras soluções existentes.

2.1. Capacidade de leitura direta e integrada

Uma das limitações encontradas em muitos fluxos de trabalho que visam integrar modelos BIM com motores de jogos é a fragmentação do processo e da informação [7]. Nestes cenários, os modelos originais precisam de ter a sua geometria convertida para outro formato, como *Wavefront* ou glTF, e sua informação alfanumérica, quando utilizada, é convertida para um formato como JSON. Após estas conversões, as informações então podem ser consumidas pelos motores e então associadas com utilização do identificador global de cada objeto (i.e., GUID). Por sua vez, estes processos de conversão intermediários são executados em outros *softwares* e/ou ferramentas (p. ex. Blender e IfcConvert), o que torna um processo pouco dinâmico e passível de erros.

Visto esta limitação, o requisito principal do GDIFC era permitir o carregamento direto das informações geométricas e alfanuméricas do IFC dentro do Godot, para evitar a fragmentação no processo.

2.2. Capacidade de leitura do modelo “in-game”

O desenvolvimento de *software* através do Godot ocorre através da criação de cenas, compostas de nós e *scripts*. Este processo é fundamentado no design direcionado a objetos, de forma que os nós representam desde elementos como objetos 3D a botões e abstrações como pontos de vida e são criados a partir de classes. Ao desenvolver aplicações no motor de jogos, as atividades passam por determinar os nós em cada cena, assim como o seu comportamento individual e de troca de informações entre si. Esta configuração é pensada para ser executada dentro do jogo, de tal forma que a configuração que ocorre no editor é maioritariamente estática. Para capacitar a criação de aplicações que permitam consumir IFCs, foi preciso desenvolver a capacidade do leitor de ser executado dentro do jogo (*in-game*). De forma prática, para desenvolver esta capacidade, foi preciso permitir a utilização de método de leitura e processamento de IFC com a utilização da linguagem GDScript, o que foi implementado via classes que serão apresentadas na secção 2.4.

2.3. Dependências tecnológicas

O principal recurso utilizado para o desenvolvimento do GDIFC foi sistema GDExtension, que permite ampliar as capacidades do motor de jogos através de extensões baseadas em C++. Desta forma, as funcionalidades adicionadas com este sistema possuem elevado desempenho, se comparado aos *plugins* desenvolvidos em GDScript ou mesmo em C#.

A utilização do C++ como linguagem principal favoreceu a integração de outras duas bibliotecas que serviram de base para o desenvolvimento das capacidades de processamento do IFC: *engine_web-ifc* e *IfcOpenShell*.

A Web IFC é uma biblioteca que possui núcleo da arquitetura em C++, sendo bastante eficaz no processo de conversão das representações geométricas do modelo IFC, como o “Brep”, “SweptSolid” e “CSG” em malhas poligonais. Apesar de também possuir capacidade de obter informações alfanuméricas, a API da Web IFC em C++ não é completa para este fim, tendo por defeito limitações na capacidade de percorrer as relações das entidades no IFC.

Já o *IfcOpenShell* é uma biblioteca que também possui núcleo em C++, apesar de ter uma API em Python que é mais popularmente utilizada. Dentre os módulos que compõem a biblioteca, o *IFCParser* é o responsável pelo processamento inicial do modelo IFC, permitindo o processamento do modelo e criação dos respetivos objetos através de um modelo de classes em C++. Assim, todas as informações do modelo estão disponíveis através da API do *IFCParser*, permitindo o acesso às informações segundo o modelo inicialmente definido em EXPRESS.

Apesar do *IfcOpenShell* também possuir um módulo para processamento geométrico, o *IfcGeom*, este depende da biblioteca *OpenCascade*, o que torna os projetos desenvolvidos a partir do *IfcGeom* mais pesados e com API mais complexa do que o da Web IFC.

A combinação dos componentes da Web IFC e do *IfcOpenShell*, então, permitiram criar uma solução simplificada, capaz de facilmente gerar malhas poligonais a partir da representação geométrica dos elementos em IFC e de obter todas as informações alfanuméricas a partir de API baseada em C++.

2.4. Fluxo de processamento do IFC e integração ao Godot

De maneira simplificada, o procedimento de leitura do IFC e integração com o Godot foi desenvolvido a partir dos três seguintes passos:

- Processamento da informação alfanumérica
- Processamento da geometria
- Integração das informações em objetos do Godot

Este fluxo ocorre de forma assíncrona no motor para a executar o processamento (a) e (b) num *thread* paralelo e não travar o motor de jogos de forma prolongada, havendo, entretanto, latência conquanto de (c). Na prática, isto permite que durante a execução de aplicações criadas com o GDIFC, seja possível continuar a interagir com um modelo enquanto outro é carregado.

Em termos da estrutura do projeto, duas classes principais foram criadas: GDIFCManager e IFCNode. O primeiro é responsável pela coordenação e processamento das informações do ficheiro, de tal forma que possui os principais métodos para ler e buscar informações ao IFC. O segundo é responsável por conter todas as informações alfanuméricas de cada um dos elementos do IFC, assim como conter as malhas poligonais e os elementos agrupados conforme as relações de agregação do IFC. Logo, a estrutura básica, como apresentada na Figura 1, possui o GDIFCManager como nó principal, sendo desagregada em IFCNodes que podem ou não conter geometria.

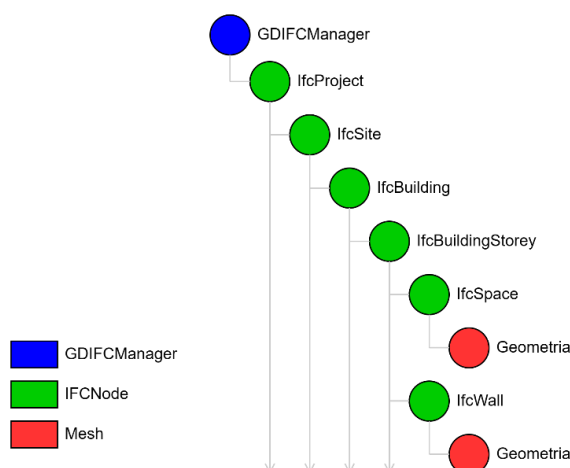


Figura 1: Exemplo de árvore parcial de IFC no Godot.

2.5. Instalação, uso e configurações

O código do GDIFC está completamente disponível em repositório aberto ¹, com indicações gerais para compilação e uso. Entretanto, a forma mais simples de obter o GDIFC é a partir da biblioteca de recursos do Godot, que pode ser acedida através do motor em qualquer projeto ².

¹ <https://github.com/Muniz1994/GDIFCpub>

² <https://godotengine.org/asset-library/asset/4212>

Há dois modos principais para utilização: através da interface e através de *scripts*. O primeiro envolve o carregamento do IFC na interface do Godot, e serve especialmente para quando se deseja fazer algum tipo de pós-processamento no modelo ou a adição de agentes. Neste caso, a aba GDIFC, localizada na área direita do editor, deve ser utilizada. Esta aba contém o botão para carregar o IFC, assim como configurações de carregamento, a exemplo da criação automática de colisões.

O segundo envolve o carregamento através da utilização do GDScript, e é recomendada para a criação de aplicações com o Godot baseadas em IFC. O carregamento do IFC pode ser feito com o método `read_ifc()`, da classe `GDIFCManager`. Após isto, as informações referentes aos outros elementos podem ser obtidas através do acesso a atributos e propriedades dos respetivos nós.

2.6. Limitações e Desempenho

De forma diminuir o tamanho dos ficheiros da biblioteca, as versões do IFC suportadas pelo GDIFC são 2.3.0.1 (2x3), 4.0.2.1 (4 ADD2 TC1) e 4.3.2.0 (4.3 ADD2). Desta maneira, o tamanho do ficheiro que é descarregado para instalação é de aproximadamente 65 MB.

Testes com modelos de testes disponíveis em repositórios geridos pela buildingSMART no Github foram utilizados para determinar o desempenho no carregamento dos modelos no Godot. Para isso, um código em GDScript foi elaborado para iniciar um cronómetro antes da função de carregamento com paragem logo após o carregamento de todos os objetos e suas componentes geométricas e alfanuméricas. O sistema utilizado para os testes é composto de um processador AMD Ryzen 5 7600X, uma placa gráfica Nvidia 4070 super e 32 GB de memória RAM. A Tabela 1 apresenta os modelos, suas versões IFC, tamanho e tempos de carregamento.

Tabela 1: Desempenho de carregamento do GDIFC

	Versão IFC	Tamanho (MB)	Tempo de carregamento (s)
Viadotto Acerno	4.3.2.0	31,5	3,52
Infra Landscaping	4.0.2.1	2,94	0,14
Schependomlaan	2.3.0.1	47,0	3,88

3. Exemplos de aplicação

Este trabalho apresenta exemplos de aplicação para demonstrar o potencial de utilização do GDIFC, tanto para ferramentas usuais quanto a nível de investigação. Desta forma, os exemplos apresentados nas próximas subsecções são compostos por protótipos com breve explicação de como o GDIFC foi integrado.

3.1. Visualizador

Este caso apresenta a capacidade de desenvolvimento de um visualizador de IFC baseado em Godot + GDIFC. As capacidades básicas definidas para o visualizador são o carregamento do modelo, apresentação da árvore de estrutura espacial e seleção dos elementos, assim como apresentação de informações alfanuméricas dos elementos seleccionados (Figura 2).

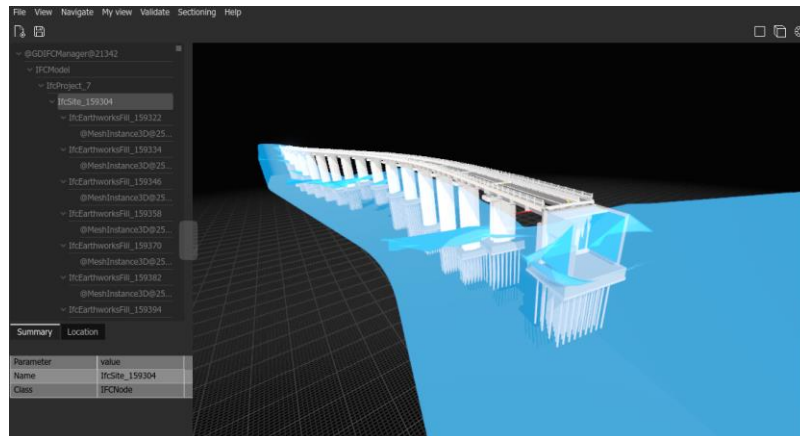


Figura 2: Exemplo de interface de utilização do visualizador.

A integração do GDIFC para a implementação do visualizador ocorre primeiramente pela capacidade de requisitar o GDIFCManager para inserir o IFC na cena 3D. No exemplo desenvolvido, um botão de abertura do ficheiro foi criado na interface de utilizador para a permitir abrir uma janela do sistema, criar uma instância do GDIFCManager na cena 3D e fornecer o caminho do ficheiro para que os elementos do IFC fossem também inseridos na cena. Após o carregamento do modelo, elementos da interface, como a árvore de elementos e as tabelas, são desbloqueados e fazem referência ao modelo e aos seus elementos. Além disso, todos os componentes da interface permitem interagir com as informações do modelo, para destacar visualmente elementos do ramo da árvore seleccionado.

3.2. Integração com 3D tiles

O ambiente do motor de jogos também propicia uma facilitada integração com informações vindas em formato 3D tiles, já que o formato emprega dados serializados em JSON e glTF, que de facto podem ser integrados aos motores. Assim, é possível coordenar a integração de informações do IFC lidas através do GDIFC e de tiles que permitam contextualizar a envolvente do edifício, como os tiles fotorrealistas da Google (Figura 3).

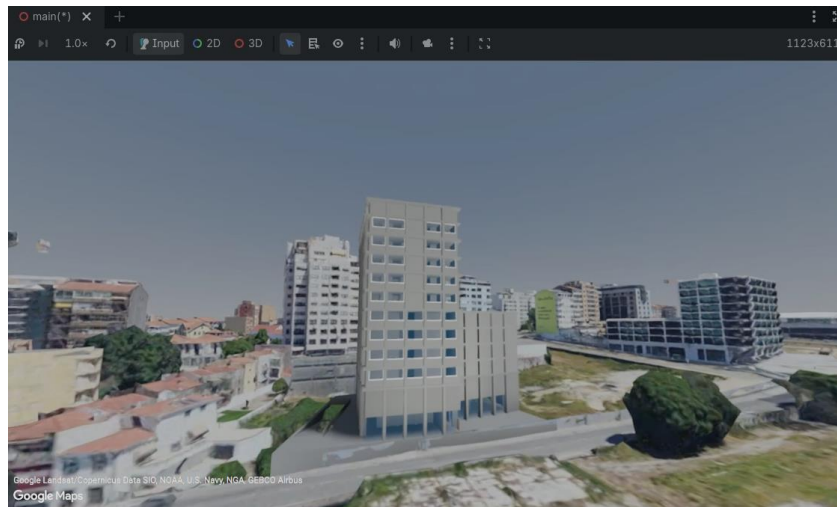


Figura 3: Exemplo de integração com *tiles* fotorrealistas da Google.

3.3. Aprendizagem por reforço (Reinforcement Learning)

Como apresentado em [14], a integração do Godot com o Python permitiu o desenvolvimento da extensão Godot-rl, que capacita o treino de agentes dentro do motor de jogos através de algoritmos disponíveis no *Stable Baselines*, *Sample Factory*, entre outros. Esta capacidade já foi explorada no âmbito da construção em trabalho desenvolvido na Universidade do Minho [15]. A utilização do GDIFC diminui a necessidade de pós-processamento dos modelos para o treino de agentes, já que o comportamento de colisão para elementos pode ser definido automaticamente no carregamento do modelo. Além disso, elementos podem ser desligados facilmente em grupos definidos pelas suas classes IFC.

Desta maneira, o trabalho passa a ser configurar o comportamento do agente, sua interação com o ambiente e as observações, que podem ser definidas, por exemplo, por sensores constituídos de raios identificadores da intersecção com objetos, como demonstrado na Figura 4.

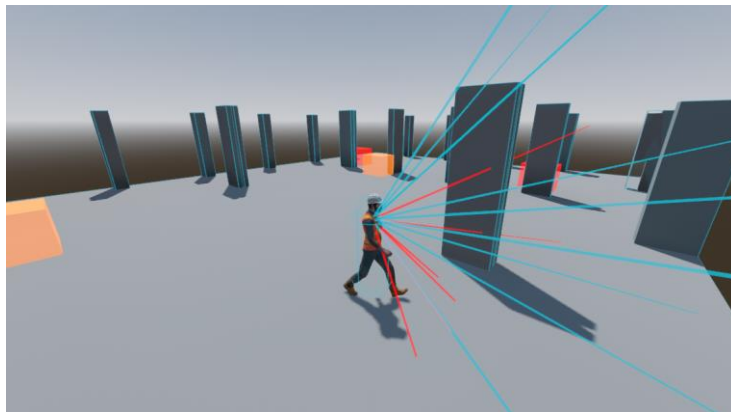


Figura 4: Agente a executar observações no ambiente através de sensores compostos por raios de colisão.

4. Conclusões e desenvolvimentos futuros

Este trabalho apresentou o desenvolvimento da extensão GDIFC para o motor de jogos Godot, que permite a integração direta de modelos IFC, com as informações geométricas e alfanuméricas. Com esta integração, o motor de jogos pode ser facilmente utilizado para o desenvolvimento de ferramentas BIM assim como para o desenvolvimento de simulações, com ou sem integração de aprendizagem automática.

As capacidades do GDIFC desenvolvidas até o momento já permitem usar o Godot para para variados fins, o que é demonstrado de forma simplificada com os exemplos desenvolvidos. Entretanto, capacidades de integração com modelos de cidade e de criação e edição de IFCs a partir da extensão ainda são previstos, o que permitirá acrescer os benefícios e capacidade das ferramentas desenvolvidas.

Agradecimentos

Este trabalho foi apoiado pela FCT/MCTES no âmbito da Unidade de I&D Instituto para a Sustentabilidade e Inovação em Estruturas de Engenharia (ISISE), com as referências UID/4029/2025 (<https://doi.org/10.54499/UID/04029/2025>) e UID/PRR/04029/2025 (<https://doi.org/10.54499/UID/PRR/04029/2025>), e no âmbito do Laboratório Associado Produção Avançada e Sistemas Inteligentes ARISE com a referência LA/P/0112/2020.

Realizado ao abrigo do Projeto “AIGENT - Gamificação baseada em IA para segurança na construção” (ref.: C744727449-00027455), financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER), através do Sistema de Incentivos à Investigação e Desenvolvimento, no âmbito do Programa Operacional para a Competitividade e Internacionalização do Portugal 2030.

Referências

- [1] C. M. Eastman, Ed., *BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors*, 2. ed. Hoboken, NJ: Wiley, 2011.
- [2] C. U. Aleke, W. O. Usang, A. Obi-obuoha, A. A. Jolaosho, and O. O. Michael, ‘Artificial intelligence as a tool for enhancing building information modeling (BIM)’, *World J. Adv. Res. Rev.*, vol. 24, no. 2, pp. 1833–1846, Nov. 2024, doi: 10.5281/zenodo.15104909.
- [3] ‘Video gaming report 2026: how platforms are colliding and why this will spark the next era of growth’, BCG Global. Accessed: Feb. 12, 2026. [Online]. Available: <https://www.bcg.com/publications/2025/video-gaming-report-2026-next-era-of-growth>
- [4] P. Amiri, ‘Review of rendering evolution of game engines in the 3D era’, 2024, Accessed: Feb. 15, 2026. [Online]. Available: <https://lutpub.lut.fi/handle/10024/167993>
- [5] A. Lehto, D. Morello, and K. Korppoo, ‘Game design deep dive: traffic systems in cities: skylines’, Game Developer. Accessed: Apr. 29, 2026. [Online]. Available: <https://www.gamedeveloper.com/design/game-design-deep-dive-traffic-systems-in-i-cities-skylines-i->

- [6] M. Assaf, S. Assaf, X. Li, and M. Al-Hussein, ‘A multi-user game-based system for planning modular construction activities’, *Expert Syst. Appl.*, vol. 299, p. 130050, Mar. 2026, doi: 10.1016/j.eswa.2025.130050.
- [7] S. U. Rehman, I. Kim, and K.-E. Hwang, ‘Advancing BIM and game engine integration in the AEC industry: innovations, challenges, and future directions’, *J. Comput. Des. Eng.*, vol. 12, no. 4, pp. 26–54, Apr. 2025, doi: 10.1093/jcde/qwaf030.
- [8] ‘Importing IFC files into unreal engine using datasmith | unreal engine 5.7 documentation | epic developer community’, Epic Games Developer. Accessed: Feb. 15, 2026. [Online]. Available: <https://dev.epicgames.com/documentation/en-us/unreal-engine/importing-ifc-files-into-unreal-engine-using-datasmith>
- [9] *Chair-intelligent-technical-design/IFC-unity-editor-plugin*. (Jan. 21, 2026). C#. Chair-Intelligent-Technical-Design. Accessed: Feb. 15, 2026. [Online]. Available: <https://github.com/Chair-Intelligent-Technical-Design/IFC-Unity-Editor-Plugin>
- [10] M. Isaac and K. Browning, ‘How a pricing change led to a revolt by unity’s video game developers’, *The New York Times*, Oct. 02, 2023. Accessed: Feb. 15, 2026. [Online]. Available: <https://www.nytimes.com/2023/10/02/technology/how-a-pricing-change-led-to-a-revolt-by-unitys-video-game-developers.html>
- [11] ‘Introduction’, Godot Engine documentation. Accessed: Feb. 15, 2026. [Online]. Available: <https://docs.godotengine.org/en/stable/about/introduction.html>
- [12] ‘The big game engines report of 2025’, Video Game Insights. Accessed: Feb. 15, 2026. [Online]. Available: <https://app.sensortower.com/vgi>
- [13] ‘What is GDExtension?’, Godot Engine documentation. Accessed: Feb. 15, 2026. [Online]. Available: https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/what_is_gdextension.html
- [14] E. Beeching, J. Debangoye, O. Simonin, and C. Wolf, ‘Godot reinforcement learning agents’, Dec. 07, 2021, *arXiv*: arXiv:2112.03636. doi: 10.48550/arXiv.2112.03636.
- [15] A. O. O. Baka’di, ‘Exploring agent-based models for construction planning using partial 4D BIM models’, Universidade do Minho, Guimarães, 2025.