SoftwareX 26 (2024) 101665

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

GAGI: Game engine for Artificial General Intelligence experimentation

Juan Carlos de la Torre *, José M. Aragón-Jurado, Abdón Crespo-Álvarez, Guillermo Bárcena-González

School of Engineering, Universidad de Cádiz, Spain

ARTICLE INFO

Keywords:
Game engine
Artificial Intelligence
AGI experimentation
AI agent
Real-time interaction

ABSTRACT

Video games have been in the focus of the research and academic community for the last few years, with the study and experimentation of Artificial General Intelligence (AGI) standing out. AGI experimentation platforms allow to analyze and study, in a visual way, the behavior of different AI agents previously defined. In this work a novel game engine, called GAGI, capable of serving as an AGI experimentation platform is presented. As a game engine, GAGI is able to design and create novel 2D and 3D video games using C++ programming language. Moreover, GAGI provides the user with a unique environment for simulating and studying AI agents inside the created game. Users can deploy multiple AI agents while interacting with them in real time, improving the understanding of their interactions and behaviors. The features of the proposed software is compared against others widely-used game engines in the video games industry as well as in the research community, highlighting the advantages in terms of design capability and AI support. GAGI also offers the possibility to reproduce the experiments, opening up multiple possibilities for the research community.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00741
Permanent link to Reproducible Capsule	https://github.com/goal-group-uca/GAGI/releases/tag/Latest
Legal Code License	GPL-3.0
Code versioning system used	git
Software code languages, tools, and services used	C++, OpenGL, ArrayFire, GLM, GLEW, GLFW, stb_image
Compilation requirements, operating environments & dependencies	OpenGL, Arrayfire, GLM, GLEW, GLFW and stb_image installed according to the provided installation manual, at least C++20 and at least Visual Studio 2019.
If available Link to developer documentation/manual	https://github.com/goal-group-uca/GAGI/blob/main/docs/
Support email for questions	abdon.crespoalvarez@mail.uca.es, juan.detorre@uca.es

1. Motivation and significance

1.1. Introduction

Nowadays, video games are one of the most relevant entertainment industries. In 2022, video games industry generated worldwide profits of 182.9 billions dollars [1]. Therefore, more companies become interested in this sector, developing new products.

In order to increase their profit, companies should develop video games for multiple platforms like the actual generation of video consoles (f.e. PS5 and Xbox Series X) or personal computers with different hardware settings. In consequence, game engines have been growing in popularity, thanks to its usability [2]. First game engine appears in 1993 with the release of the video game Doom, where Id Software allows developers to easily create first-person shooter games like Doom o Quake.

Furthermore, in the last few years, the research and academic community has put the focus on video games [3]. The application of video games in these fields, particularly in the areas of serious games [4,5] and Artificial Intelligence (AI) [6], has been significant. The development of new game engines has played an important role in the progress of these areas, making advances faster and more efficient [7,8].

* Corresponding author.
E-mail addresses: juan.detorre@uca.es (Juan Carlos de la Torre), abdon.crespoalvarez@mail.uca.es (Abdón Crespo-Álvarez).

https://doi.org/10.1016/j.softx.2024.101665
Received 1 November 2023; Received in revised form 13 February 2024; Accepted 15 February 2024
Available online 21 February 2024

In the domain of AI, the use of Deep Reinforcement Learning (RL) approaches has increased dramatically [6,9], evidenced by their achievements in multiple popular genres of video games like Multiplayer Online Battle Arena (MOBA) [10], Real Time Strategy (RTS) [11], and First Person Shooter (FPS) [12]. This progress has caused the development of new Deep RL training environments, with a growing focus on those centered around Artificial General Intelligence (AGI) as is the case of Malmo Project [13], ViZDoom [14], and DeepMindLab [15]. Unlike current approaches to AI, AGI aims to develop AI systems capable of learning to perform well across a wide range of tasks, resembling the flexible learning observed in humans and other animals.

In this context, with the aim to build a game engine capable of serving as a AGI experimentation platform, we present GAGI, a 3D game engine with the ability to reproduce simulated scenarios, leaving the source code documented and easy to understand for other researchers and developers. Furthermore, it serves as a AGI testing ground, obtaining significant information about how the AI agent behaves when it tries to complete its task. Throughout this work, we have compared our proposal with the most relevant game engines of the moment.

GAGI is available for download at:https://github.com/goal-group-uca/GAGI/releases/tag/Latest.

1.2. Related work

Over the past five years, there has been a notable increase in the development of Deep RL training platforms as documented in the literature. A range of environments for training Deep RL agents has emerged, beginning with classic video games like the Atari-57 benchmark [16] and extending to well-known 3D video games including Doom [14] and Minecraft [13].

One of the most relevant Deep RL training environment is the Malmo Project developed by Microsoft [13], an AGI experimentation platform built on top of the game Minecraft. Malmo leaves agents in a 3D environment with complex dynamics, allowing researchers to exploit their properties inside a Minecraft world.

Currently, game engines are increasingly gaining popularity among both independent developers and larger companies for video game creation. A segment of the study in [8] focuses on identifying the most commonly used game engines within the industry. This is achieved by analyzing data from two major digital video game platforms for PC: Steam [17], which features both large and small-scale projects, and Itch.io [18], known for its independent developer community. The findings reveal that Unity [19] and Unreal Engine [20] dominate the market, with Unreal Engine being more prevalent in Steam-based projects and Unity being the preferred choice on Itch.io.

Both engines have a Graphical User Interface (GUI), enabling users to efficiently manage video game projects with minimal effort. Moreover, they offer a graphical scripting language, allowing to program the game logic without extensive programming knowledge.

Another relevant game engine, although not as popular as the previous ones, is Godot Engine [21]which stands out for its scripting language and performance [22]. Godot is notable thanks to being an open source project with an user-friendly GUI and a easy integration with a multitude of programming languages available in the industry such as C++, Python or Rust among others. This versatility has inspired the community to develop tools like Godot RL Agents [23]. Godot RL Agents enriches the engine with a Python interface that facilitates communication between the Godot environment and various Deep RL training frameworks, such as RLib.

1.3. Game engines comparison

1.3.1. Methodology

We compared our game engine against three widely used game engines, Unity, Unreal Engine and Godot, as well as a Deep RL training platform that specializes in Minecraft, Project Malmo. In this way, we highlighted both its advantages as a game engine and as an AGI experimentation platform. There are various works in the literature that compare different game engines, though they are focused from the perspective of the typical developer [24,25]. However, we need a comparison from the point of view of Deep RL research. From this perspective, we considered five key points to qualitatively assess each game engine, based on the features presented in [22]:

- Usability. The usability of a game engine is a crucial factor which involves learning, efficiency of use, as well as overall user satisfaction. Usability is splitted into two features: the design of a user-friendly GUI and the availability of a good language to define AI agents.
- Design capability. It focuses on the engine's versatility in creating diverse video games, essential for AI experiments. We assessed it by looking at supported programming languages, the dimensionality of games, and the ability to extend the engine's functionality.
- AI agents support. Having support for AI Agents is a key point to be able to study their behavior inside video games. Therefore, we considered two features such as support for AI algorithms in the game engine as well as the possibility for the agent to interact with the player and with other agents.
- Game reproducibility. For researchers, ensuring the replication of the game conditions is crucial to study the behavior of the agents. For this purpose, we studied two important features are studied : the replicability of experiments and the possibility of reproducing a previously played game.
- User accessibility. The accessibility and cost of a game engine are significant criteria in determining its suitability for the AI agents research.

We used "Poor", "Good" and "Excellent" ratings for the evaluation of game engine features for AI research, except for user accessibility (assessed by software license type), dimensionality (types of games that can be created), and agent interaction (detailing possible interactions).

1.3.2. Results

Usability should consider different types of users, especially when more and more researchers and academics are interested in the field of video game development [26–31]. All of the considered game engines have a GUI to manage the game design process but they differ in functionality. Godot Engine and Unity offers really user-friendly GUI, allowing novice users to learn quickly, while the GUI offered by Unreal Engine is more complex. Project Malmo offers a Minecraft insight of the different AI agents previously designed, while, the GUI of GAGI offers this also in conjunction with the possibility to coexist with users. Users can modify the world while the agents are running, altering their behavior.

Considered game engines have dedicated AI agent extensions: Unreal Engine Learning Agents [32] (with limited models and requiring C++ and Python 3), Godot RL Agents [23] (offering multiple AI models and integrating with its scripting language), and Unity Machine Learning Agents [33] (with a visual block language for easier AI agent definition). In contrast, Project Malmo is fully implemented in C++ and uses XML for defining scenarios, while GAGI, also in C++, provides neural network models and allows adding new AI models via a well-documented API.

Another relevant aspect is the design capability of each game engine. Project Malmo is built on top of the Minecraft, restricting it to only design AI agents in the Minecraft game. Unity, Unreal Engine and Godot are totally versatile game engines, allowing the development of all types of games in 2D and 3D. Users in GAGI can build Minecraft type games visually using the edition mode. However, it is a full 2D and 3D graphics game engine and with some C++ programming skills, users can use GAGI to develop other types of 2D and 3D games with support for AI agents.

Table 1
Comparison between different game engines.

	Features	Unreal Engine	Unity	Godot Engine	Project Malmo	GAGI
Usability	User-friendly GUI	Good	Excellent	Excellent	Poor	Good
	Language to define AI agents	Good	Excellent	Excellent	Good	Excellent
Design capability	Programming languages	Good	Good	Excellent	Poor	Good
	Extensibility	Poor	Poor	Good	Poor	Excellent
	Dimensionality	2D and 3D	2D and 3D	2D and 3D	Minecraft	2D and 3D
AI support	AI agents	Poor	Good	Excellent	Good	Excellent
	Interaction	No interaction	Multi-agent	No interaction	Multi-agent and player	Multi-agent and player
Game reproducibility	Replicability	Poor	Good	Good	Good	Excellent
	Game state replayer	Poor	Poor	Poor	Poor	Excellent
User accessibility		Commercial license (Source code access)	Commercial license	Full free open source	Full free open source	Full free open source

Understanding the behavior behind the AI algorithm is a hard task, and even more so when trying to understand how it learns. In this field, Project Malmo stands out for enabling player and AI agent interaction, a feature not present in Godot and Unreal Engine. Unity, on the other hand, only supports interactions among multiple agents, without direct player involvement. Our game engine, GAGI, offers the possibility to play with multiple AI agents while studying their interactions. In addition, multiple AI configurations and models are available in GAGI, which users can use without programming skills.

Furthermore, when we talk about AI experimentation, a crucial point is the reproducibility of experiments. As was the case for AI support, Unreal Engine, Godot and Unity do not offer direct gameplay reproducibility. Users need to use their programming skills to record gameplays and reproduce them. On the other hand, Project Malmo allows recording and replaying the behavior of specific agents. Nevertheless, our game engine GAGI also record the behavior of multiple agents at the same time along with the player's action. Moreover, GAGI lets the user rewind and fast-forward the gameplay to properly analyze each frame.

Open source code is a key feature in order to researchers and developers be able to understand well a game engine and, in consequence, extend its functionality. Project Malmo, Godot and our proposal GAGI follow a open source code license, which allows the code to be freely accessed and used. However, this is no the case for Unreal Engine and Unity. Unreal Engine source code is open for all users, although it is ruled by specific license model known a Unreal Engine End User License Agreement (EULA), limiting what users can do with the code. For Unity, source code is not available and there are commercial paid licenses to access to some functionalities.

Table 1 presents a brief comparison between the different game engines considered.

All of the above form the novelty of GAGI, a open source game engine capable of building 2D and 3D games that in turn serve as an AGI testing field. GAGI offers a editing mode to build and study multiple AI agents inside a Minecraft like game along with the player interactions. In addition, users can replay, rewind, and fast-forward reproducible scenarios under the same conditions according to their needs.

2. Software description

2.1. Software architecture

Our game engine provides the users with a API of six levels of abstraction. Fig. 1 shows briefly the purpose of each layer and Table 2 gives a description of the classes of each of them.

Fig. 2 presents the different interactions among the different layers of the API. As evident, each layer interacts with the immediately preceding it. Moreover, Programming Layer interacts with both the Graphics Independent Layer and the Core Game Layer, abstracting all that functionality to the programmer. A more detailed interaction diagram between classes can be found in the GitHub repository.[1]

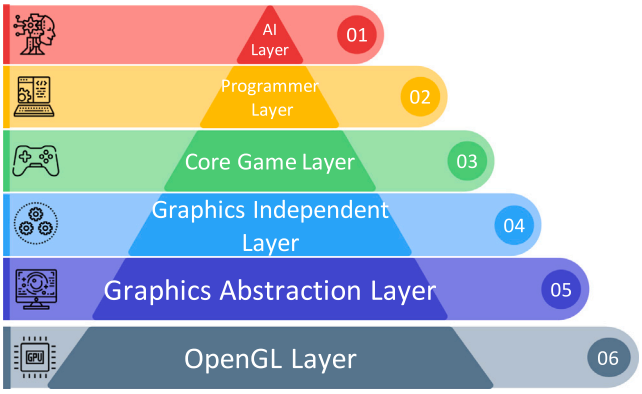

Fig. 1. Different layers of the game engine API.

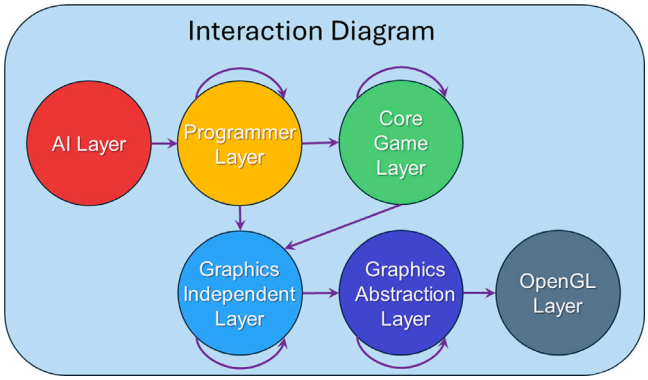

Fig. 2. Interactions among the different API layers.

2.2. Software functionalities

GAGI consists of two fundamental parts: the game engine and the AI API. In the first part, users can create and manipulate levels created with voxel-like terrain. Moreover, in case additional functionality is required, users can extend the engine functionality using the API provided through the C++ programming language.

In case some dynamism is required, GAGI API offers namespaces and managers for developers to add custom functions and methods. For example, if the user wanted to add a sheep that walks and eats grass,

[1] https://github.com/goal-group-uca/GAGI/tree/main/docs

Table 2
Available classes in the different layers of the API.

Layer	Description
Layer 6: OpenGL Layer	– Renderers: Draw call types. – Shaders: GLSL abstractions for GPU vertex processing. – Textures: Detailing images for models. – Vertices: Points in space with extra data. – Buffers: Storage for graphics data.
Layer 5: Graphical Abstraction Layer	– Graphics: OpenGL API encapsulation. – Color: Color format abstraction.
Layer 4: Graphics Independent Layer	– Controls: User inputs. – Window: Game engine window specs. – Camera: Viewpoint with cone/direction. – Models: Graphics shapes API. – Batch: Meshes collection. – ChunkRenderingData: Terrain chunk render data. – Noise: Simplex/Perlin functions.
Layer 3: Core Game Layer	– Player: User capabilities and attributes. – Chunk: World's cubic optimization. – WorldGen: Terrain generation API. – Entity: Non-terrain objects.
Layer 2: Programmmer Layer	– Game: Engine and cycle management. – World: World info methods. – AIAgentAction: AI agent actions. – GUI: Interface elements and functions. – InputsFunctions: GUI-bound functions. – Managers: Chunks, entities, and GUI management- – Time: Monitoring functions. – Threading: Thread pools. – Definitions: Common instructions/constants. – Utilities: Logging and debugging tools.
Layer 1: AI Layer	– AIGame: AI game base class.

user can load the sheep model into an entity object and bind the entity to a function that simulated the behavior of the sheep.

For the second part, GAGI provides base classes for developers to extend using derived classes in order to create custom AI games with their rules. In addition, developers can define game-ending conditions and the AIs that will control the players, along with specifying how to save these AIs in auxiliary memory.

AI models are defined through programming and can be as simple or as complex as required. To enhance user accessibility, our API incorporates implementations of both the multi-layer perceptron and genetic algorithms. This facilitates the genetic training of defined AI agents. Additionally, the API is designed with extensibility in mind, allowing users to integrate new models by extending the existing classes within the API framework. This design choice significantly expands the scope for user customization and experimentation in AI agent development. In the end, the engine will provide information from the level to the user and the user will decide what to give to the AIs. In consequence, the agents will modify the state of the game based on the defined AI actions.

Furthermore, there is the possibility to create a separate copy of the same level for each AI agent participating in the game. This feature can be used to improve performance when the user needs to train a large number of agents.

Every AI game can be recorded and replayed in any moment, providing the user with the reproducibility of their experiments. Reproducibility is a key feature for researchers, offering them not only the possibility to create games with AI agents easily but also a safe and replicable environment.

3. Illustrative example

To illustrate a real-time example and help users understand the development process, we recorded an AI game.[2] In this game, we place an AI agent in a randomly generated level with a limited number of action points. Each defined AI action incurs a certain cost in action points, and the agent aims to achieve the highest score by mining blocks in the generated level. The score varies based on the block type, with deeper blocks yielding higher points. Fig. 3 displays a frame from the recorded AI game.

The AI agent has limited actions, able to see only what is within a cone three blocks away and a radius of three blocks. Additionally, the agent can only pick up the block in front of it and move forward. Although the agent can rotate on its axis, this action costs fewer action points than moving. In the example game, all AI actions, except rotation, cost 1 action point. Rotation around itself incurs a cost of 0.10 action points for each rotation. Each AI agent starts with an initial 500 action points.

The defined AI agent consists of a five-layer neural network, with three hidden layers containing 50 neurons each. No additional layers exist between the input and output layers in order to find the balance between avoiding hypertraining and allowing the AI agent to learn complex behaviors. To speed up the training process, users can define and train multiple AI agents simultaneously, with each using its own copy of the same level to avoid interference with the training of other agents. An example of how a trained AI performs in this AI game can be found in the video provided below at minute 0:47.[3]

To analyze the performance of the game engine, we have taken multiple empirical measurements of time in milliseconds for different aspects of the recorded example game. The computer used for the experiments is an Intel(R) Core(TM) i7-9700F CPU @ 3.00 GHz (8 cores - no hyper-threading) with 16 GB RAM, a 1TB SATA SSD and a NVIDIA GeForce GTX 1660. All the experiments were executed over Windows 10 64-bit operating system.

Table 3 shows the average time in milliseconds of the chunk management system in the example game. A chunk is a division of terrain made of 16 × 16x16 blocks. The user can modify the number of chunks loaded in each axis. However, in the example the number of chunks in the Y axis is set to 24 to avoid restricting the agent's search space to only the deepest areas of the level.

[2] https://youtu.be/hQgiJ9GGmxs

Fig. 3. Screenshot of the AI game.

Table 3
Average management times of the chunks in ms in the example AI game.

Chunks	Generation time	Loading time	Saving time
10	1656	743	646
20	6646	2898	2409
30	14 956	6541	5593
40	26 329	11 608	9922
50	41 253	17 649	15 586

Table 4
Average training related times in milliseconds of the AI agents.

Individuals	One epoch	Saving data	Loading data
10	36 514	212	2547
20	64 615	283	2465
30	117 562	458	2802
40	137 706	579	2736
50	183 082	710	2881

If we talk about the training process of the AI agent, Table 4 illustrates the average epoch time in milliseconds, as well as the average loading and saving times of the data in the system, also in milliseconds. Typically, around epoch 500, AI agents start to mimic the strategy of Minecraft players to collect their resources. The most valuable resources are scarce and found in the deepest layers, causing players to mine at a specific depth. Nevertheless, as the stone blocks give a point to the agents, once the agents are at the right depth, they mine all the blocks, contrary to the players. AI agents score approximately 250 points following this strategy. Considering that the agent has to pay 2 action points to move forward and collect the block in front of it, we can conclude that the AI has maximized the score to an acceptable degree with the set number of initial action points.

4. Impact

We expect that features of GAGI make it wide usable by researchers and academics as an AGI experimentation platform capable of design 2D and 3D video games with AI agents. On the one hand, academics can use it to visually explain different AI models to students, aiding in the understanding of their algorithms. In addition, the reproducibility of GAGI offers researchers a platform for benchmarking games alongside AGI experimentation, allowing them to use meta-models and metaheuristic techniques with the game.

The possibility of building completely new 2D and 3D games with support to multiple AI agents will be really appreciated both in the research and academic communities. GAGI not only offers a game engine capable of replaying previously played games under the same conditions, but users can also easily define new AI algorithms to interact with the world. The API provided by GAGI allows interaction with intelligent systems of any kind, including neuromorphic systems designed for use with simulators such as NesimRT [34]. Its extensibility also applies to the field of game design, allowing users to extend the capabilities of the game engine using the C++ programming language.

GAGI is a fully open source project distributed under GPLv3 license. We generated a Doxygen documentation for each method in the game engine, ensuring that each user can use them and extend their functionality. Each release of the game engine comes with an installation guide, containing all the steps to properly install GAGI and its dependencies.

5. Conclusions

In this work, we presented a novel 2D and 3D game engine capable of serving as an AGI experimentation platform called GAGI. The developed tool integrates game engines and AGI platforms together, becoming a pioneer in doing so without relying in a video game. A recording tool has been developed that allow replaying previously played games with the option to rewind or fast-forward frames, facilitating performance measurement or analysis of the behavior of AI agents. GAGI modularity offers the possibility to easily define new AI algorithms in the engine, as well as to extend the functionality of the game engine.

CRediT authorship contribution statement

Juan Carlos de la Torre: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Funding acquisition, Conceptualization. **José M. Aragón-Jurado:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Conceptualization. **Abdón Crespo-Álvarez:** Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Guillermo Bárcena-González:** Writing – review & editing, Writing – original draft, Validation, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

Jose M. Aragón-Jurado is funded by Spanish MCIU, Spain under the FPU grant FPU21/02026. This publication is supported by projects eFracWare (TED2021-131880B-I00) funded by Spanish MCIN, Spain and the European Union "NextGenerationEU", and project eMob (PID2022-137858OB-I00) funded by Spanish MCIN, Spain, the AEI, Spain and the ERDF, and project NEMOVISION (PID2019-109465RB-I00) funded by Spanish MCIU, Spain and the ERDF.

References

[1] NewZoo. Games Market Reports and Forecasts. 2024, https://newzoo.com/games-market-reports-forecasts. [Accessed 12 February 2024].
[2] Andrade A. Game engines: A survey. EAI Endorsed Trans Serious Games 2015;2(6).
[3] Xia B, Ye X, Abuassba AO. Recent research on AI in games. In: 2020 international wireless communications and mobile computing. IEEE; 2020, p. 505–10.
[4] Cowan B, Kapralos B. A survey of frameworks and game engines for serious game development. In: 2014 IEEE 14th international conference on advanced learning technologies. IEEE; 2014, p. 662–4.
[5] Checa D, Bustillo A. A review of immersive virtual reality serious games to enhance learning and training. Multimedia Tools Appl 2020;79:5501–27.
[6] Skinner G, Walmsley T. Artificial intelligence and deep learning in video games a brief review. In: 2019 IEEE 4th international conference on computer and communication systems. ICCCS, IEEE; 2019, p. 404–8.
[7] Gazis A, Katsiri E. Serious games in digital gaming: A comprehensive review of applications, game engines and advancements. WSEAS Trans Comput Res 2023;11:10–22.
[8] Toftedahl M, Engström H. A taxonomy of game engines and the tools that drive the industry. In: DiGRA 2019, the 12th digital games research association conference. Digital Games Research Association (DiGRA); 2019.
[9] Souchleris K, Sidiropoulos GK, Papakostas GA. Reinforcement learning in game industry—Review, prospects and challenges. Appl Sci 2023;13(4):2443.
[10] Ye D, Liu Z, Sun M, Shi B, Zhao P, Wu H, et al. Mastering complex control in MOBA games with deep reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence, vol. 34, (no. 04):2020, p. 6672–9.
[11] Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 2019;575(7782):350–4.
[12] Lample G, Chaplot DS. Playing FPS games with deep reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence, vol. 31, (no. 1). 2017.
[13] Johnson M, Hofmann K, Hutton T, Bignell D. The Malmo platform for artificial intelligence experimentation. In: Proceedings of the twenty-fifth international joint conference on artificial intelligence. IJCAI '16, AAAI Press; 2016, p. 4246–7.
[14] Schulze C, Schulze M. ViZDoom: DRQN with prioritized experience replay, double-Q learning and snapshot ensembling. In: Intelligent systems and applications: proceedings of the 2018 intelligent systems conference (intelliSys) volume 1. Springer; 2019, p. 1–17.
[15] Beattie C, Leibo JZ, Teplyashin D, Ward T, Wainwright M, Küttler H, et al. Deepmind lab. 2016, arXiv preprint arXiv:1612.03801.
[16] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. OpenAI Gym. 2016, arXiv preprint arXiv:1606.01540.
[17] Steam store. 2024, https://store.steampowered.com/. [Accessed 13 February 2024].
[18] Download the latest indie games. 2024, https://itch.io/. [Accessed 13 February 2024].
[19] Unity real-time development platform | 3D, 2D, VR & AR engine. 2024, https://unity.com/. [Accessed 13 February 2024].
[20] Unreal engine | The most powerful real-time 3D creation tool. 2024, https://www.unrealengine.com/. [Accessed 13 February 2024].
[21] Godot Engine. Godot engine - free and open source 2D and 3D game engine. 2024, https://godotengine.org/. [Accessed 13 February 2024].
[22] Sharif KH, Ameen SY. Game engines evaluation for serious game development in education. In: 2021 international conference on software, telecommunications and computer networks. IEEE; 2021, p. 1–6.
[23] Beeching E, Debangoye J, Simonin O, Wolf C. Godot reinforcement learning agents. 2021, arXiv preprint arXiv:2112.03636.
[24] Christopoulou E, Xinogalos S. Overview and comparative analysis of game engines for desktop and mobile devices. Int J Serious Games 2017;4(4). http://dx.doi.org/10.17083/ijsg.v4i4.194.
[25] Vohera C, Chheda H, Chouhan D, Desai A, Jain V. Game engine architecture and comparative study of different game engines. In: 2021 12th international conference on computing communication and networking technologies. IEEE; 2021, p. 1–6.
[26] Ullah M, Amin SU, Munsif M, Safaev U, Khan H, Khan S, et al. Serious games in science education. a systematic literature review. Virt Real Intell Hard 2022;4(3):189–209.
[27] González Sánchez JL, Padilla Zea N, Gutiérrez FL. From usability to playability: Introduction to player-centred video game development process. In: Kurosu M, editor. Human centered design. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009, p. 65–74.
[28] Murphy-Hill E, Zimmermann T, Nagappan N. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In: Proceedings of the 36th international conference on software engineering. 2014, p. 1–11.
[29] Politowski C, Petrillo F, Ullmann GC, de Andrade Werly J, Guéhéneuc Y-G. Dataset of video game development problems. In: Proceedings of the 17th international conference on mining software repositories. 2020, p. 553–7.
[30] Politowski C, Guéhéneuc Y-G, Petrillo F. Towards automated video game testing: Still a long way to go. In: Proceedings of the 6th international ICSE workshop on games and software engineering: engineering fun, inspiration, and motivation. 2022, p. 37–43.
[31] Dewan R, Polishetty R, Jagadam N, Goyal MK, Ravulakollu KK, Sharan B. Significance of state-of-art search engine in game development. In: 2023 10th international conference on computing for sustainable global development. IEEE; 2023, p. 1134–9.
[32] Learning agents. 2024, https://docs.unrealengine.com/5.3/en-US/BlueprintAPI/LearningAgents/. [Accessed 13 February 2024].
[33] Juliani A, Berges V-P, Teng E, Cohen A, Harper J, Elion C, et al. Unity: A general platform for intelligent agents. 2020, arXiv preprint arXiv:1809.02627.
[34] Rosa-Gallardo DJ, de la Torre JC, Quintana FM, Dominguez-Morales JP, Perez-Peña F. NESIM-RT: A real-time distributed spiking neural network simulator. SoftwareX 2023;22:101349. http://dx.doi.org/10.1016/j.softx.2023.101349.